

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

2020-021

Project Proposal Report

Saranga S.A.G

B.Sc. (Hons) in Information Technology Specializing 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) in Information Technology Specializing 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:

IT17016230	Saranga S. A. G	
------------	-----------------	---

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

Signature of the Supervisor: .....

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

## Abstract

With the growth of the information technology industry and the use cases of software solutions, the complexity of the implementations increased over time. After a series of improving Software architectures, Microservices have become one of the most popular development architectures for many software applications developed today. The main reason for this popularity is due to its efficient and effective way to decompose a large and complex system into their functional components and implement a loosely coupled, self-contained system that supports scalability and performance. Even though microservice architecture has its many benefits, management of hundreds of microservices which deployed in a various physical location is challenging.

Kubernetes comes to play to solve the challenges of maintaining a sizeable microservice structure and make life a little easier for a system engineer. Therefore, many organizations deploy their application embracing microservice architecture and make use of platforms such as Kubernetes to ensure the successful deployment of their application in production.

Even with the use of microservice orchestration tools such as Kubernetes, monitoring the whole application to identify the performance bottlenecks and risks is challenging. Monitoring the whole application is vital when it comes to a larger organization with a vast number of interconnected, highly dependent microservices.

This research carried out intentions to develop a model to find out the dependency level between each microservices to compose a better deployment strategy with the support for auto-deployment. Furthermore, provide a dynamic monitoring solution with a Business Intelligence dashboard.

**Keywords:** Service Mesh, Dependency, Container, Docker, Kubernetes, Machine Learning, Microservices, Time Series, Business Intelligence

# 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>4</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>13</b>
<b>3.2.3 Economic Feasibility .....</b>	<b>13</b>
<b>3.3 System Analysis.....</b>	<b>14</b>
<b>3.3.1 Software Solution .....</b>	<b>14</b>
<b>3.4 System Development and Implementation .....</b>	<b>15</b>
<b>3.4.1 Building the Dependency Network.....</b>	<b>17</b>
<b>3.5 Project Requirements .....</b>	<b>22</b>
<b>3.6 Testing.....</b>	<b>23</b>
<b>3.7 Time Line.....</b>	<b>24</b>
<b>4.0 PERSONNEL AND FACILITIES .....</b>	<b>25</b>
<b>5.0 COMMERCIALIZATION .....</b>	<b>26</b>

**6.0 BUDGET..... 27**

**7.0 SUMMARY ..... 28**

**References..... 29**

## List of Figures

Figure 1.0.0 Comparison Between Virtual machines and Containers .....	2
Figure 1.1.1 Monolithic Architecture.....	4
Figure 1.1.2 Microservice Architecture .....	5
Figure 1.1.3 Microservice Technologies Timeline. ....	6
Figure 1.2.1 Current Automated Process of Kubernetes.....	8
Figure 3.3.1 System Overview Diagram .....	14
Figure 3.4.1 Proposed Methodology .....	15
Figure 3.4.2 Service Mesh Architecture .....	17
Figure 3.4.3 Istio Architecture For A Single Cluster .....	18
Figure 3.4.4 Istio Architecture For A Multi Cluster .....	18
Figure 3.4.5 Prometheus Architecture .....	19
Figure 3.4.6 Sample Grafana Dashboard .....	20
Figure 3.4.7 Sample Kiali Dashboard .....	20
Figure 3.4.8 Sample Jaeger Dashboard.....	21
Figure 3.7.1 Gantt Chart.....	24

## List of Tables

Table 4.3.1 Tools and Languages to be Used in the Implementation .....	21
Table 4.0.1 Personnel and Resources .....	25
Table 6.0.1 Budget .....	27

## List of Abbreviations

AKS	Azure Kubernetes Services
API	Application Program Interface
APM	Application Performance Monitoring
AWS	Amazon Web Service
CD	Continuous Deployment
CI	Continuous Integration
CLI	Command Line Interface
DB	Database
DEVOP	Development and Operations
HTTP	Hypertext Transfer Protocol
IT	Information Technology
OS	Operating System
REST	Representational State Transfer
SOA	Service-Oriented Architecture
TLS	Transport Layer Security
TOSCA	Topology and Orchestration Specification for Cloud Applications
VM	Virtual Machine

## 1.0 INTRODUCTION

Microservices have grown to one of the most popular development structures for many software applications developed today [1] , mainly since microservices provide an effective way to develop a software application as a set of interconnected modular services that can be independently deployed and scaled. Prime companies such as Netflix, Uber, and Amazon moved from their traditional architectures to embrace the microservice architecture because of simplicity, support, and, most importantly, continuous deployment and integration.

With the growth of the industry and the usage of information technology, even microservices became too complicated. To solve this complexity of microservices, the technology "Containerization" [2] came in to play. The idea of containers was started way back in 1979 with **UNIX Chroot**. [3] It is a UNIX operating-system system call for transforming the root directory of a process, and it is children to a new position in the filesystem, which is only visible to a given process.

Most of the time, containers compared to virtual machines since they both allow the software to execute in a contained environment. However, they are different in the structure and the process of execution.

VMs are an abstraction of the hardware layer, meaning that each VM simulates a physical machine that can run the software. VM technology can use one physical server to run the equivalent of many servers, each of which is called a VM. So, while multiple VMs run on one physical machine, each VM has its copy of an Operating System, applications and their related files, libraries, and dependencies.

Containers are an abstraction of the application layer, meaning that each container simulates a different software application. Though each container runs isolated processes, multiple containers share a universal Operating System.

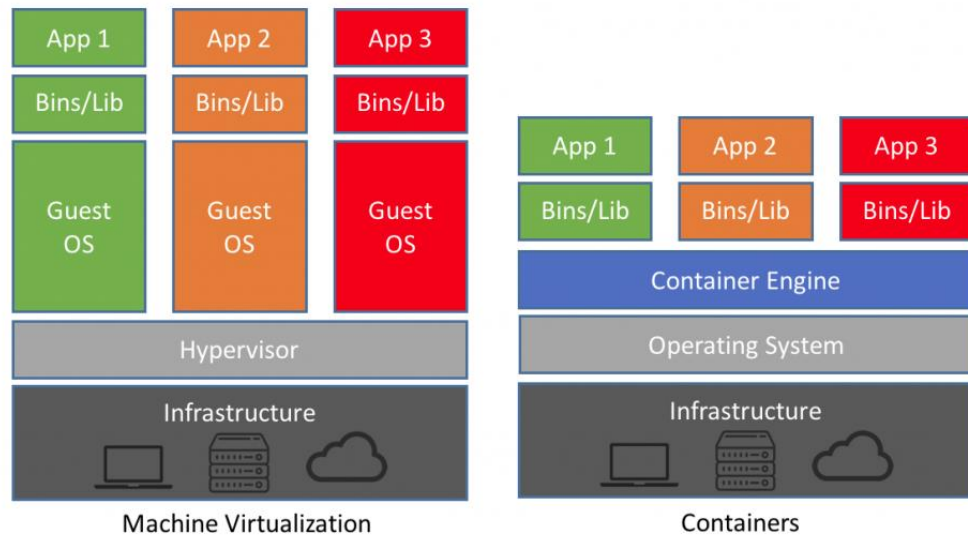


Figure 1.0.0: Comparison between Virtual Machines and Containers

With the popularity of containerization, there are several platforms to provide the services of containerization, including Docker, VirtualBox, Vagrant, and Wox [4]. All these platforms have their ups and downs, but docker holds the position of the leading platform when it comes to the containerization.

There are five main advantages of docker, which are speed, portability, scalability, rapid delivery, and density [5]. Furthermore, since all required dependencies, libraries, and all other components needed for the microservice packaged up in the container itself, the software is isolated from its environment, ensuring the performance and the functionality of the software remain the same regardless of the underlying platform [6].

Nevertheless, with the use of docker containers in production, practical problems become visible such as which container runs on which node, how to improve the performances, how to add more replications to add redundancy, and how to communicate with each node. Kubernetes designed to overcome all the above-mentioned practical problems and many more [7].

Kubernetes [8], which introduced in 2014, is a portable, extensible, open-source platform for managing containerized workloads and services that facilitates both declarative configuration and automation. It has a vast, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available [9]. Integrating Kubernetes into an application's deployment strategy, therefore, enables an organization to deploy quickly, manage, and handle an application, keeping the inherent benefits of container-based deployments while ensuring service availability. The existence of these unique characteristics in Kubernetes has therefore resulted in its widespread use in microservice deployments and increased popularity.

However, even though the use of deployment tools like Kubernetes, there are still some issues that need to address. Kubernetes can be an overkill for simple applications. Furthermore, even though Kubernetes introduced to simplify sophisticated containerizations, the Kubernetes itself can be complicated for developers not familiar with the infrastructure and cause low productivity.

The primary issue being that deployment tools like Kubernetes perform deployments without really taking into consideration about the interdependency among deployed microservices. Although this modular structure is what makes the microservices scalable, the downside is that there is no guarantee that interconnected services are deployed in the same node or nearby nodes unless configured otherwise. If those services deployed far away each, then issues such as network latency could be a problem.

This research aims to provide a solution to this significant problem by taking the initial step by aiming to develop a model that takes into consideration the overall dependency between each microservices. The expected output model will aid in reducing the network latency of the microservice system, and the users of the model will be able to get an idea about the overall dependency structure of the application in a network as a whole.

## 1.1 Background and Literature

Dr. Peter Rogers came up with the wording of "Micro web services" in 2005 during an initial conference of cloud computing. The term "Microservices" has revealed at an event for software architectures where the term used to define a new software architecture that many computer scientists were experimenting at that time [10]. Amazon, Uber, and Netflix were among the first few tech giants to embrace this new architecture in their businesses [11].

The concept of microservices came to the world to solve the challenges of the vastly growing software and systems, which were getting more and more complex rapidly [10]. The idea was to divide the complex applications to more manageable smaller applications and allow them to communicate with each other using an implementation language-independent interface.

Before the idea of microservices, which displayed in figure 1.1.2, almost all the companies were using monolithic architecture, as depicts in figure 1.1.1, which was running into problems in areas of managing, scaling, maintaining, and automation.

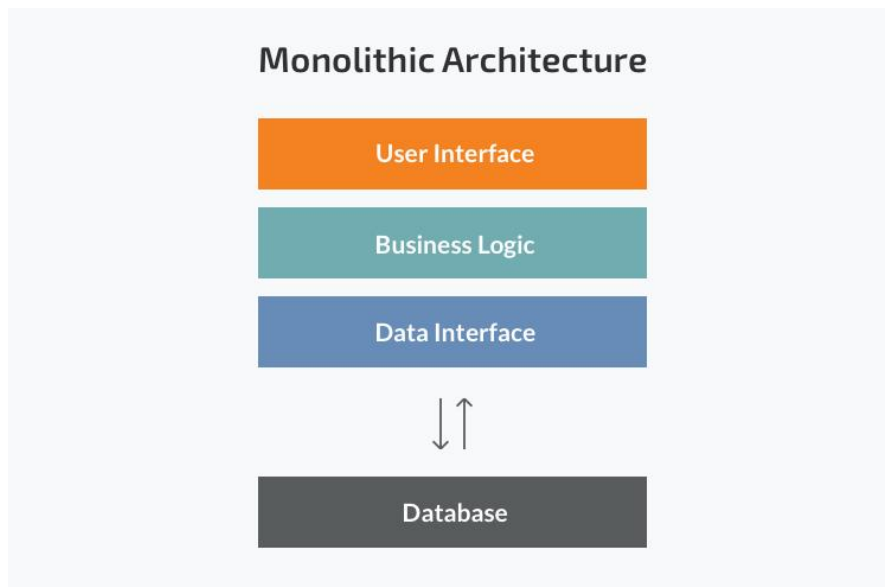


Figure 1.1.1: Monolithic Architecture

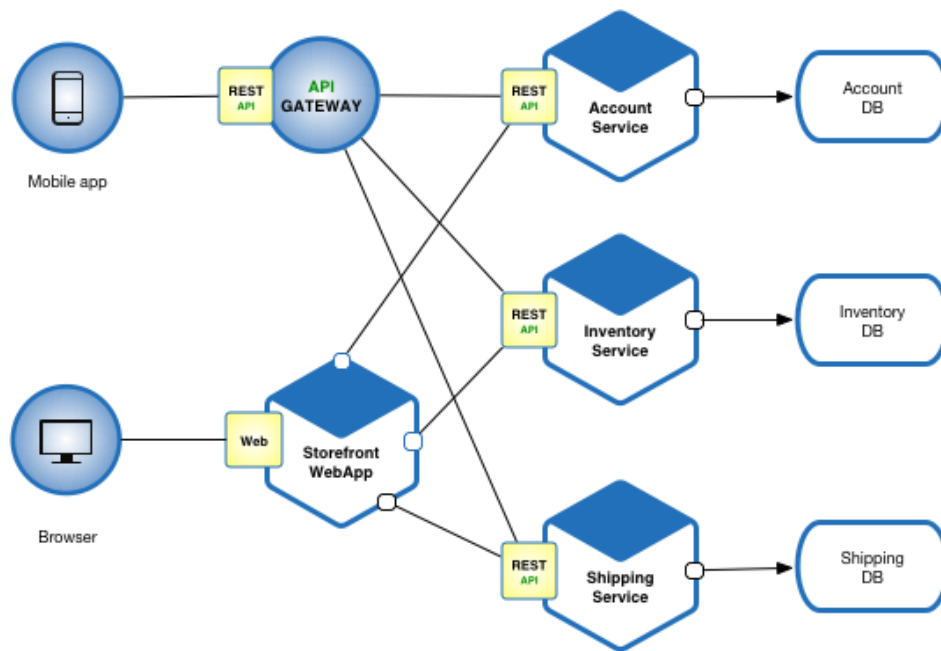


Figure 1.1.2: Microservice Architecture

Another key difference when between microservice and monolith architecture is when it comes to how they governed. This term is known as "Microservices Governance" concerning a microservice architecture, and it can define as a methodology or approach that establishes policies, standards, and best practices for the adoption of microservices to enable an enterprise agile IT environment [10]. Governance in monoliths is centralized, and decisions made "top-down" [10], whereas governance in microservices embraces a decentralized governance approach which, enables microservices to make use of a polyglot model technology stack in the development of applications.

However, the decentralized approach used in the governance of microservices should take more steps to ensure effective governance is maintained since typical applications require interconnections between a vast number of microservices where business process workflows introduced continuously. Therefore, in order to ensure an effective microservice governance is in place, organizations currently make use of a variety of tools that facilitate tasks such as monitoring, autoscaling, configuration management, fault tolerance, and many more. Over the years, with the increased popularity of

microservice architecture, new tools and technologies were introduced to support microservice-based technologies, as shown in Figure 1.1.3 to support improved scalability, monitoring, and deployments.

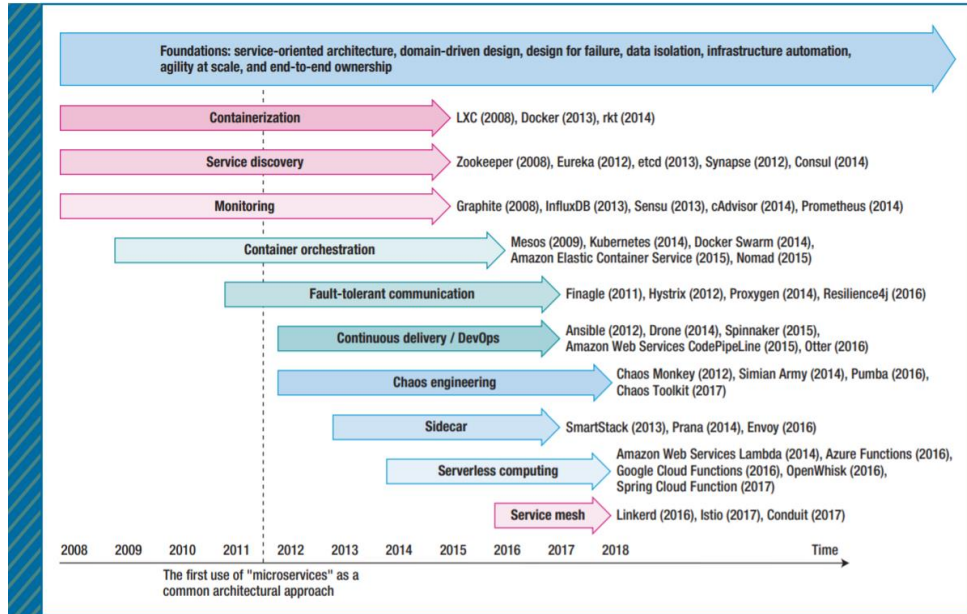


Figure 1.1.3: Microservice technologies timeline [30].

Even though Kubernetes and all other tools resolve various problems and improve the functionality of the microservices, there are still some problems and performance bottlenecks either those tools cannot solve, or the tools introduce. Past researches [12, 9, 13] describes the drawbacks and cons of Kubernetes when it comes to containerized microservices.

The past research [14] shows the importance of having a monitoring solution to monitor the workload and the performance of the cluster. Istio came into the play to solve this common problem. At a high level, Istio helps reduce the complexity of these deployments and eases the strain on the development teams. It is an entirely open-source service mesh that layers transparently onto existing distributed applications. It is also a platform, including APIs that let it integrate into any logging platform, or telemetry or policy system. Istio's diverse feature set lets the development team

successfully, and efficiently, run a distributed microservice architecture, and provides a uniform way to secure, connect, and monitor microservices [15].

There are lots of previous researches about building a service mesh and developing monitoring solutions. Nevertheless, there were no past publications have been done considering the quantified dependency level between microservices. It is quite clear that taking the dependency level into account will help when it comes to building a better monitoring solution.

## **1.2 Research Gap**

The primary research gap that this research aims at fulfilling is the gap that exists in current methodologies when considering the dependency level between microservices, particularly concerning Kubernetes. A thorough analysis of published research papers throughout the years has managed to highlight very low of the issues that are present in current microservice deployment methodologies and thereby enabled in the identification of the research gap, as mentioned in the previous statement.

The initial inspiration for the identification of this research problem and the objectives of this research can found in [16]. This publication clearly describes some of the key challenges faced in the deployment of microservices and the need for Application Performances Monitoring tools, especially those deployed in containers to include additional measures to monitor microservices such that they could use as input for resilience mechanisms and creation of auto-scaling policies.

Even though there are numerous researches about Kubernetes and service mesh, there were no publications considering the dependency between microservices as a whole. The available publications go on to describing the auto-deployment facilities of Kubernetes [17]; however, they do not describe how that can affect the network latency between microservice are deployed automatically in the available nodes.

However, when it comes to the deployment of the proposed optimal deployment strategy, generated at the end of the proposed solution, support for auto-deployment is crucial. Currently, the developers use the Yaml language [18] to write configurations as code to do the necessary deployments. Nevertheless, when it comes to handling the deployments of a vast number of microservices, it can be a headache for the development team.

According to publication [7, 8, 19], there is automated deployment support in Kubernetes, as depicted in figure 1.2.1. However, still, there is no consideration given towards the dependency level, resilience analysis, and load prediction, which are the key pillars of this proposed research.

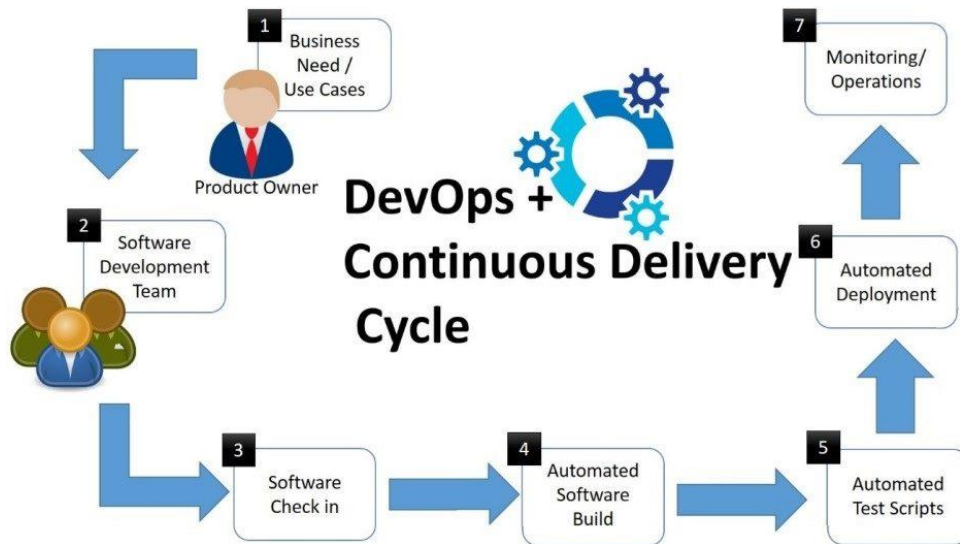


Figure 1.2.1: Current automated process of Kubernetes

### 1.3 Research Problem

Even though Kubernetes is quite successful and widely used nowadays, it is not without its unique challenges, which are especially true concerning some of the current processes in the governance of microservices through Kubernetes, particularly considering processes involving in monitoring, resource utilization, and performance optimization.

Some of the known problems to Kubernetes [20]:

- Complexity
  - Kubernetes itself is quite complex to install, configure and manage
- Unknowingly over or under-allocating the available resources with negative results
  - Pods might not start at all
  - Pods might crash at really bad times under high workload
- Using/configuring external load balancers to access the applications via the internet – yet another technology to learn and manage
- Creating health checks for every component
- Integration into the build pipelines
- Learning YAML
- Monitoring

The proposed research aims to address all the problems mentioned above by developing a model to consider the dependency levels between microservices to design a more improved deployment strategy and support automated deployment and provide a monitoring solution to get a better idea about the whole microservice architecture to get business-wise decisions.

## 2.0 OBJECTIVES

### 2.1 Main Objective

The main objective of this research to generate a dependency network based on the metrics gathered from the Kubernetes cluster to provide a better understanding of the whole microservice architecture. This aids in making decisions regarding the deployment strategy of the microservice architecture.

### 2.2 Specific Objectives

The following are the sub-objectives of conducting this research.

- To configure the Kubernetes cluster to get the metrics of the microservices and nodes
- To analyze the network and calculate the network traffic
- To support the auto deployment by generating **Yaml** files and applying them to the cluster.
- To provide a monitoring solution with the real-time metrics

## **3.0 METHODOLOGY**

### **3.1 Requirement Gathering**

Requirement gathering was through performing an extensive analysis of past research conducted throughout recent years, identification and analysis of the existing systems, as well as reading through a variety of online resources. Kubernetes' official documentation and the Istio documentation were the main resources when gathering the requirements.

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

When it comes to Past Research Analysis, there are quite a lot of research papers and publications under the topic of Service mesh. However, there was a smaller number of publications considering the dependency levels between microservices. Key topics of interest included Service Mesh, Load balancing, load prediction, and network monitoring.

During the past research analysis, the main focus was to identify the methodologies and the tools used to build the existing tools and platforms. Moreover, it helped to identify the problems that the past researchers faced.

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

There are several existing APM tools to monitor applications running on top of Kubernetes [21, 22]. However, those APMs do not provide any suggestions or support for an optimal deployment strategy. Moreover, most of the APMs are running independently, and it is harder to monitor every monitoring solution. There is a void in the current market for a monitoring solution to combine existing ones to provide a complete, more detailed platform and suggestions for a better deployment strategy and support for auto-deployment.

## **3.2 Feasibility Study**

### **3.2.1 Technical Feasibility**

#### **3.2.1.1 Knowledge on Kubernetes**

In order to develop the proposed optimization model, all members should have basic knowledge of Kubernetes and its relevant components. Members should be able to perform basic configuration and should have sufficient knowledge and practical experience on how to deploy microservices through Kubernetes.

#### **3.2.1.2 Knowledge on APM tools**

In order to develop the proposed optimization model, all members should have quite an in-depth understanding of the existing APM tools and the features and drawbacks present. The members should also have sufficient knowledge of configuring and APM tools selected for this research as well as knowledge on how to integrate the selected APM tools with the optimization model.

#### **3.2.1.3 Machine Learning Knowledge**

In order to develop the proposed optimization model, all members should have quite basic knowledge of machine learning basics as well as time series analysis. Members should be aware of the time series prediction models as well as knowledge on how to integrate the relevant models with machine learning and develop basic algorithms. Furthermore, members should also have a basic understanding of the Python programming language and related python machine learning and time series libraries.

#### **3.2.1.4 Knowledge in Microservices**

The topics like containerization, dockers, Kubernetes, and service mesh are all based on microservices. Therefore, having a good idea and understanding of the concepts and the techniques of the microservice architecture is highly essential.

#### **3.2.1.5 Knowledge in Yaml**

When it comes to auto-deployment in Kubernetes, the configuration language, Yaml [18], plays a vital role. Since one of the outcomes of the proposed research is to come up with an auto-deployment solution, knowing Yaml will be an added advantage.

#### **3.2.2 Schedule Feasibility**

As the output of this proposed research will act as an input for the researches in other team members, the proposed solution should implement within at least two months. The auto-deployment part of the proposed solution can implement at the final stage of the research.

#### **3.2.3 Economic Feasibility**

The cost of the proposed project should be as minimal as possible in order for it to be included and accepted in the existing APM tool market in Kubernetes because most APM tools and solutions offered currently concerning Kubernetes are often opensource.

### 3.3 System Analysis

#### 3.3.1 Software Solution

The overview of the proposed system is as follows. Its structure mainly composed of the following components.

- Create a Kubernetes cluster on Azure AKS [23]
- Configure Istio [15]
- Track and Monitor Network traffic and dependency
- Build the dependency network
- Support Auto deployment

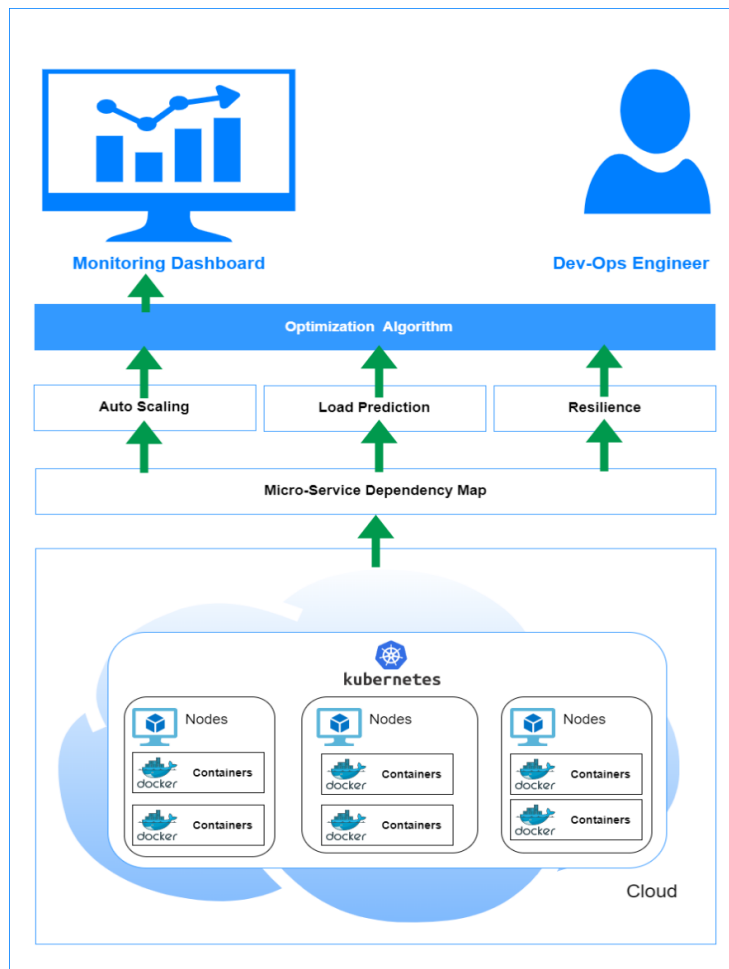


Figure 3.3.1: System Overview Diagram

### 3.3.1.1 Dependency Network

The microservice dependency network is a map that depicts the architecture of the microservice cluster based on the level of dependency among microservices. This model will use a series of tools to capture the network requests and responses and reverse engineer the metrics collected to build a dependency network based on the timestamps.

## 3.4 System Development and Implementation

The implementation process of the proposed model will perform as per Figure 3.2 given below.

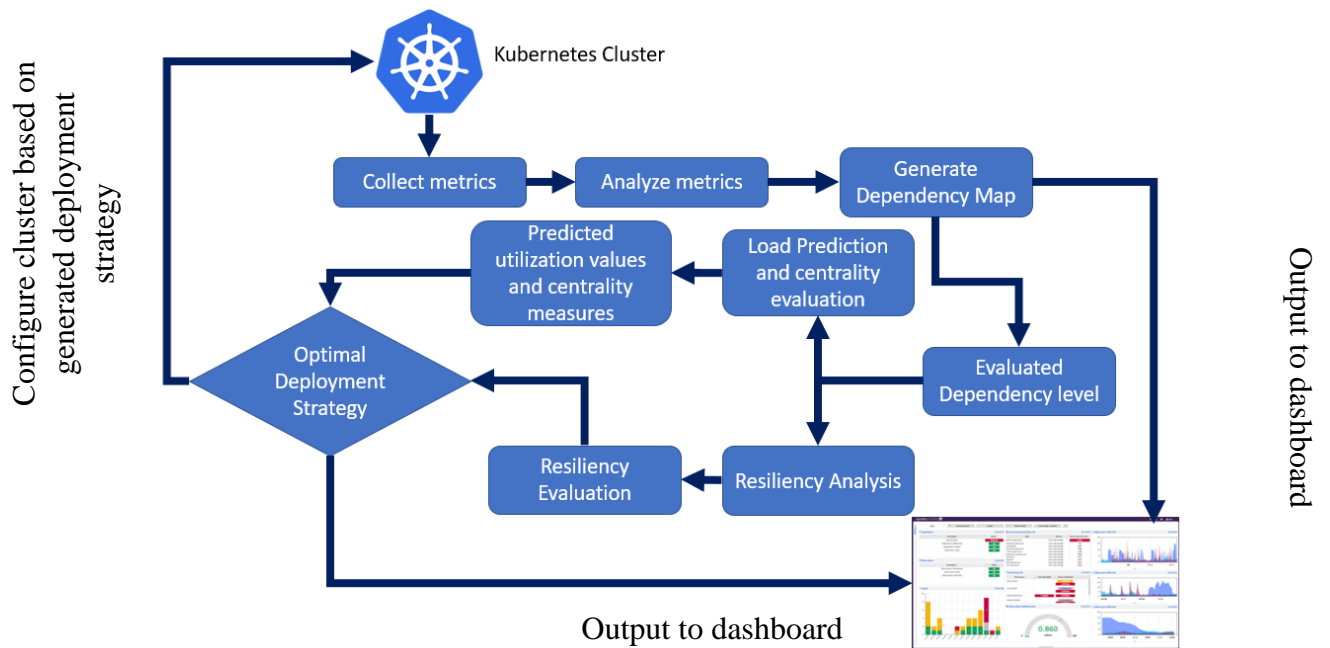


Figure 3.4.1: Proposed Methodology

The steps planned to follow in order to reach the proposed solution are as given below.

1. Create a Kubernetes cluster in Azure AKS
2. Configure Istio and deploy it to the created AKS cluster.
3. Create a proxy on top of each node to capture the network requests and responses in and out.
4. Come up with an algorithm, to backtrack the captured metrics with the timestamps to quantify the dependency levels between microservices of the cluster.
5. Create and optimize a dataset to use in the load prediction and resiliency analysis part of the proposed research.
6. Visualize the generated dependency network in the format of a chart and display it in the business intelligence dashboard.

### 3.4.1 Building the Dependency Network

The word **Service Mesh** is a frequently seen word when talking about the microservice architectures. A service mesh is a configurable, low-latency infrastructure layer designed to handle a high volume of network-based interprocess communication among application infrastructure services using application programming interfaces (APIs) [24]. A service mesh ensures that communication among containerized and often ephemeral application infrastructure services is fast, reliable, and secure. The mesh provides critical capabilities, including service discovery, load balancing, encryption, observability, traceability, authentication and authorization, and support for the circuit breaker pattern.

The service mesh is usually executed by providing a proxy instance, called a *sidecar*, for each service instance. Sidecars handle interservice communications, monitoring, and security-related concerns – indeed, anything that can be abstracted away from individual services. This way, developers can handle development, support, and maintenance for the application code in the services; operations teams can maintain the service mesh and run the app [14]. Istio, backed by Google, IBM, and Lyft, is currently the best-known service mesh architecture. Kubernetes, which was initially designed by Google, is currently the only container orchestration framework supported by Istio.

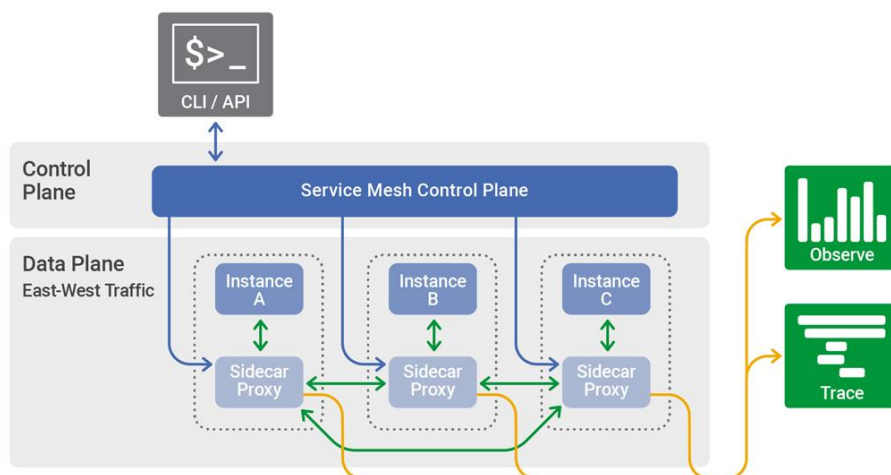


Figure 3.4.2: Service mesh architecture

Istio [15] , make it easy to convert a microservice application to a service mesh enabling load balancing, service authentication, routing, monitoring, and many more.

There are several core features bundled with Istio,

- Traffic Management
- Security
- Policies
- Observability

In the proposed research, the feature observability will use heavily.

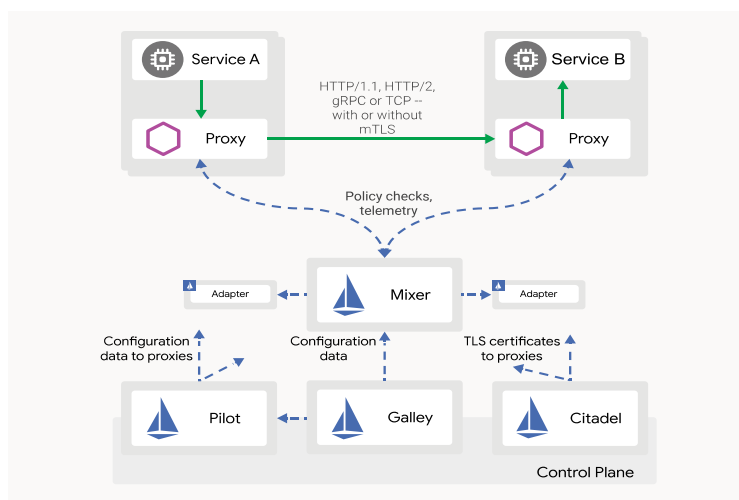


Figure 3.4.3: Istio Architecture for a single cluster

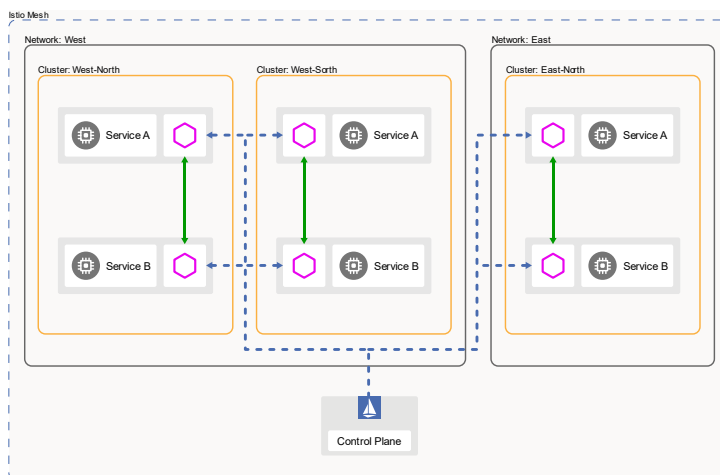


Figure 3.4.4: Istio Architecture for a multi cluster

Configuration of Istio with the Kubernetes cluster exposes a set of dashboards to the user allowing the user to visualize a set of metrics of the nodes and pods inside the Kubernetes cluster.

### **Prometheus** [25]

Prometheus is an open-source system monitoring and alerting toolkit originally built at SoundCloud. Many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. It is now a standalone open source project and maintained independently of any company.

When it comes to recording any purely numeric time series, Prometheus works like a charm. Prometheus is well known for its' reliability. It fits with monitoring dynamic service-oriented architectures, as shown in figure 3.4.4.

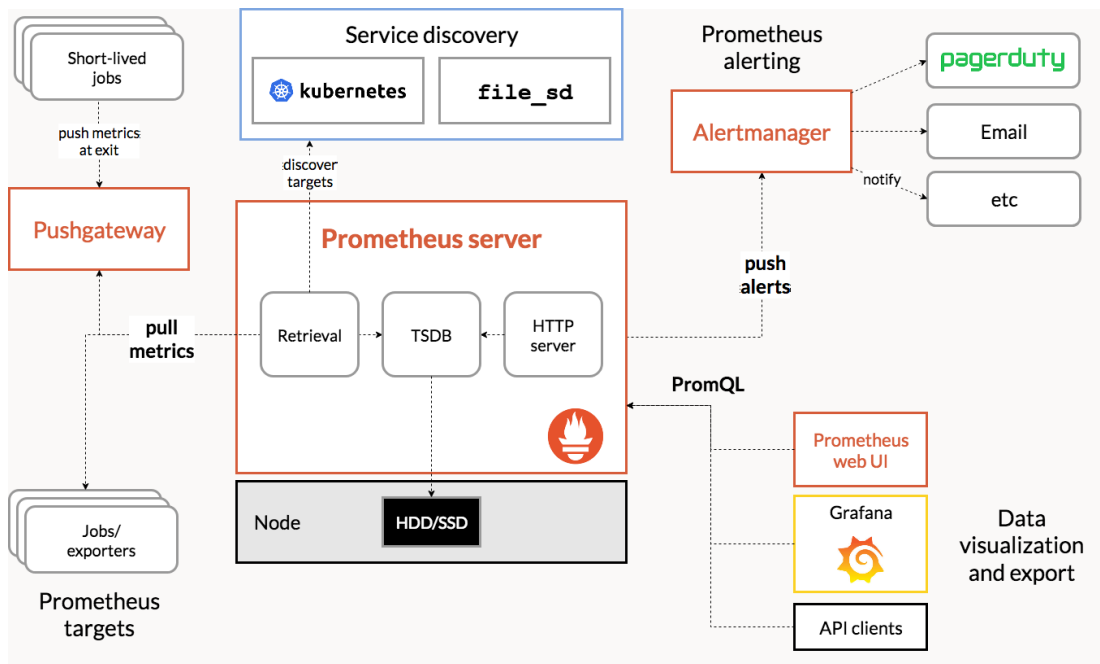


Figure 3.4.5: Prometheus Architecture

### **Grafana** [26]

Grafana is a dashboard that enables developers to query, visualize, and understand the metrics of a configured Kubernetes cluster in the format of multiple configurable charts, as depicts in figure 3.4.5. Grafana allows creating customized dashboards to capture and monitor relevant values to a particular subject.

Grafana is compatible with an array of data sources. In the case of this proposed research, Prometheus mentioned above can act as a data source for the Grafana model.



Figure 3.4.6: Sample Grafana Dashboard [31]

## Kiali

Kiali is an observability console for Istio with service mesh configuration capabilities [27]. It helps developers to understand the structure of their service mesh by inferring the topology, and also provides the health of the mesh. Kiali provides detailed metrics, and a fundamental Grafana [26] integration is available for advanced queries. Distributed tracing is provided by integrating Jaeger.

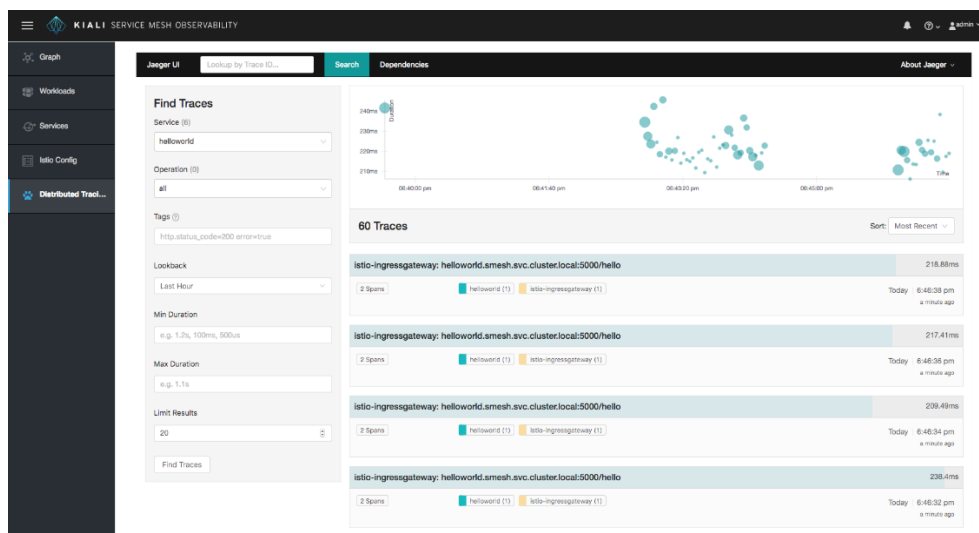


Figure 3.4.7: Sample Kiali Dashboard [27]

## Jaeger

Jaeger [28] is used heavily in production level microservice architectures to find the network traces. It is an open-source system that can use for monitoring and troubleshooting a microservice architecture, including distributed context propagation, root cause analysis, service dependency analysis, and many more.

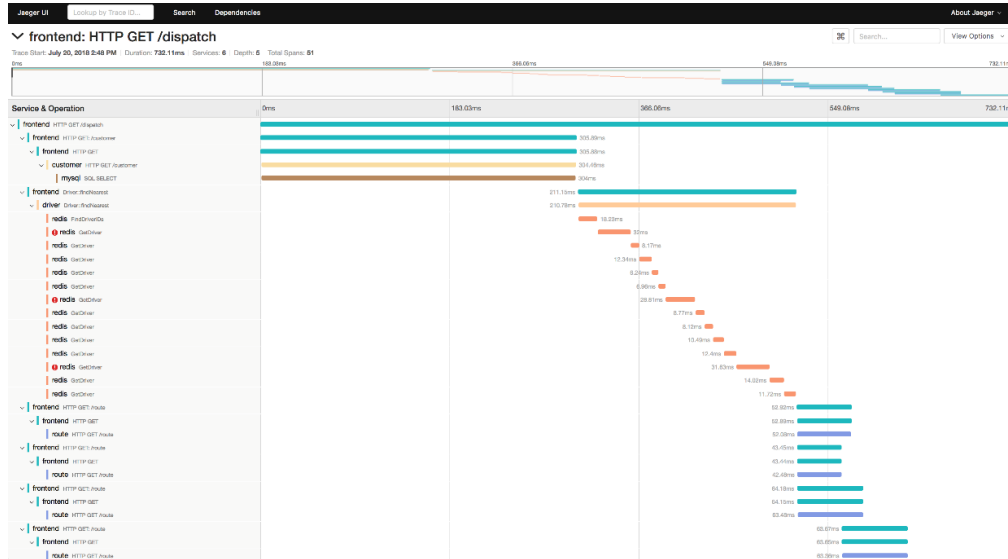


Figure 3.4.8: Sample Jaeger Dashboard

The plan is to get necessary details from the above mentioned dashboards and tools using the APIs exposed from the Istio. In summary, the following tools and languages and frameworks will use for the implementation of the proposed model.

Programming Language s	Python Java JavaScript
Configuration Languages	Yaml
Frameworks	React JS Node JS SpringBoot
Tools	Istio Grafana Kiali Prometheus Jaeger
Libraries	NetworkX (Python)

Table 3.4.1: Tools and Languages to be used in implementation

### **3.5 Project Requirements**

#### **3.5.1 Functional Requirements**

The functional requirements for the proposed model are as follows:

- Users should be able to view the whole microservice architecture in a visualized format.
- Users should be able to view the dependency network with quantified dependency levels.
- Users should be able to view the optimal deployment suggestions given by the proposed model.
- Users should be able to generate Yaml files for the selected deployments
- Users should be able to apply the generated yaml configurations directly to the cluster.

#### **3.5.2 Non-Functional Requirements**

The following are the non-functional requirements focused during the development of the proposed model.

- Usability
- Reliability
- Availability
- Interoperability
- Performance

### **3.6 Testing**

When it comes to testing the research outcome, it plays a crucial role in the success of the research. In order to test the functionality of the proposed solution, the initial plan is to create a sample set of microservices ) and deploy those to the cluster, monitor the dependency network manually.

However, in the final part of the research, the proposed plan is to deploy the output solution to a production level cluster. Moreover, check if there is an actual performance improvement, performing a comparison between the cluster before, after with the proposed deployment strategy.

Testing of the automated-deployment part will be done by applying the generated configuration yaml files directly to the cluster and making sure the necessary deployment changes have happened.

### 3.7 Time Line

The proposed timeline for the project is as follows.

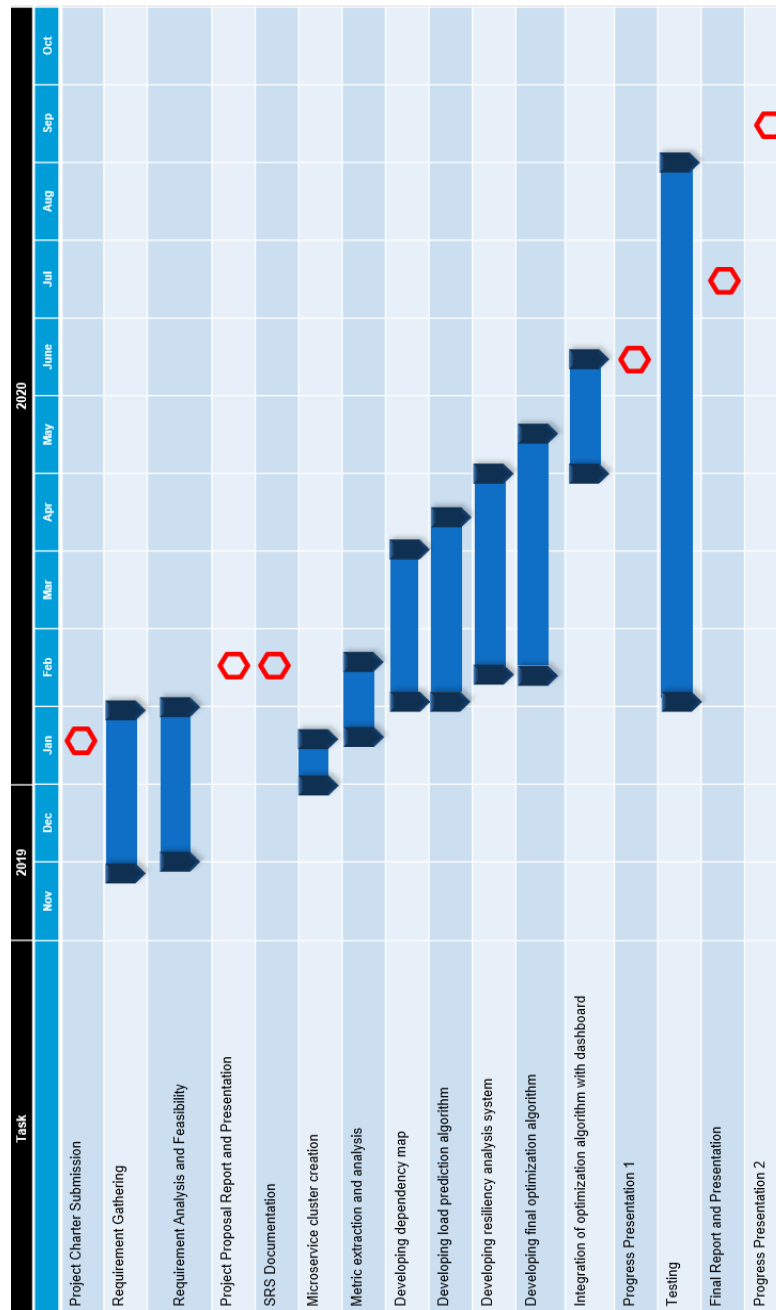


Figure 3.7.1: Gantt Chart

## 4.0 PERSONNEL AND FACILITIES

Name	Key Tasks
Saranga S.A. G	<ul style="list-style-type: none"><li>• Setting up and configuring the initial Kubernetes microservice cluster for metric analysis.</li><li>• Setting up and configuring the relevant tools such as Prometheus, Grafana, Kiali, Jager, and Istio. in order to retrieve the required metrics.</li><li>• Identify and retrieve the required metrics from the various metrics analyzing tools</li><li>• Analyze the retrieve metrics such that it will be useful to determine the interdependencies between the deployed microservices.</li><li>• Development of a solution to display the interdependencies gathered using metrics in order to display as a dependency network.</li><li>• Performing configuration of the auto-scaling of the cluster based on the load-prediction algorithm</li></ul>

Table 4.0.1: Personnel and Resources

## 5.0 COMMERCIALIZATION

The commercialization of this research project will be through the development of a tool using the proposed model. The developed tool will make use of the proposed model to provide developers and system administrators an easy and efficient way in which to optimize their Kubernetes deployment by aiming to provide the following benefits.

- Visualize the level of inter-dependency among deployed microservices
- Receive suggestions in potential ways to optimize the performance and configure current deployments and automatically perform deployments based on the suggestions.
- Provide an overview of the resiliency of the deployed microservices
- Automatically configure and auto-scale Kubernetes autoscaling tools based on predicted load and centrality measures.
- A Business Intelligence Dashboard allows users to access all the features mentioned above and provide a holistic view of the deployment.

This tool targeted to market as an APM tool for Kubernetes deployments, and due to the wide variety of APM tools currently available which are mostly free and opensource currently available in the market, the initial plan is to develop the tool into an opensource tool in order to enter the current market space effectively.

However, throughout the years, a freemium based strategy will be adopted with the inclusion of additional features.

## 6.0 BUDGET

Since the outcome of the proposed model is a software-based solution, there are no hardware components connected to the implementation. The primary source of the cost will be the subscription fees to the cloud provider for the computing power of the virtual machines.

However, there will be some other costs expected to as given in the table below.

Internet use and web hosting	2000 LKR
Publication costs	2000 LKR
Stationary	1000 LKR
TOTAL	5000 LKR

Table 6.0.1: Budget

**Note:** Azure Student subscription with \$100 of free credit for 12 months will use for this project. Therefore, resource creation for the VMs needed for testing and creation of the Kubernetes cluster, which will use for implementation purposes, could obtain without any additional cost.

## **7.0 SUMMARY**

The primary objective of this proposed research is to develop a model that aims to improve microservice governance in Kubernetes deployments through a network science-based approach.

This proposed research is focusing on generating a dependency network for the whole microservice architecture. The outcome of this proposed solution enables the user to have a clear and comprehensive idea about the deployed microservices, how they communicate with each other, and what are the dependency levels between each microservice communication.

In the second part, the proposed model focus on support automated deployment by generating the necessary configuration in the Yaml format and apply them directly to the Kubernetes cluster based on the user's requirements.

The proposed model will ultimately be able to provide developers and system administrators an overview of their current deployment configurations for performance and aid in governing their microservice deployments.

## References

- [1] S. Mandal, "Microservices — Why Is It Rapidly Gaining Popularity Now?," DZone, [Online]. Available: <https://dzone.com/articles/microservices-why-is-it-rapidly-gaining-popularity>. [Accessed 20 2 2020].
- [2] S. Ray, "Hackernoon," 31 01 2019. [Online]. Available: <https://hackernoon.com/what-is-containerization-83ae53a709a6>. [Accessed 22 2 2020].
- [3] "ComputerHope," [Online]. Available: <https://www.computerhope.com/jargon/c/chroot.htm>. [Accessed 18 02 2020].
- [4] "Docker Alternatives," Educuba, [Online]. Available: <https://www.educba.com/docker-alternatives/>. [Accessed 05 02 2020].
- [5] B. B. Rad, H. J. Bhatti and M. Ahmadi, "Advantages of Docker Container," *An Introduction to Docker and Analysis of its Performance*, vol. 16, pp. 230-231, 2017.
- [6] J. Turnbull, *The Docker Book: Containerization Is the New Virtualization*, 2019.
- [7] D. Vohra, "Why Kubernetes ?," in *Kubernetes Microservices with Docker*, pp. 41-42.
- [8] "Kuberners," Kubernetes.io, [Online]. Available: <https://kubernetes.io/>. [Accessed 14 01 2020].
- [9] The Kubernetes Authors, "What is Kuberners," Kuberners.io, [Online]. Available: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>. [Accessed 14 01 2020].
- [10] L. Mauersberger, "Microservices: What They Are and Why Use Them," *leanix.com*, [Online]. Available: <https://www.leanix.net/en/blog/a-brief-history-of-microservices>. [Accessed 20 02 2020].
- [11] I. Nadareishvili, R. Mitra, M. McLarty and M. Amundsen, in *Microservice Architecture: Aligning Principles, Practices, and Culture*, O'Reilly Media Inc., 2016, pp. 91-92.
- [12] N. Dabare, "Predictive Based Multi factor Horizontal POD Auto Scalar for Kubernetes," p. 01, 2020.
- [13] A. Modak, S. D. Chaudhary, P. S. Paygude and S. R. Ldate, "Techniques to Secure Data on Cloud: Docker Swarm or Kubernetes?," *IEEE*, 2018.
- [14] C. Fred Moyer, "Comprehensive Container-Based Service Monitoring with Kubernetes and Istio".

- [15] 2019 Istio Authors, "What is Istio?," Istio, [Online]. Available: <https://istio.io/docs/concepts/what-is-istio/>. [Accessed 21 02 2020].
- [16] W. Li, Y. Lemieux, J. Gao, Z. Zhao and Y. Han, "Service Mesh: Challenges, State of the Art, and Future Research Opportunities," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, San Francisco East Bay, CA, USA, USA, 2019.
- [17] P. Bakker, "One year using Kubernetes in production: Lessons learned," TechBeacon, [Online]. Available: <https://techbeacon.com/devops/one-year-using-kubernetes-production-lessons-learned>.
- [18] R. Gibb, "What is YAML?," StackPath, 17 7 2019. [Online]. Available: <https://blog.stackpath.com/yaml/>. [Accessed 22 2 2020].
- [19] "How to automate Kubernetes workflows," [Online]. Available: <https://buddy.works/guides/how-optimize-kubernetes-workflow>. [Accessed 24 02 2020].
- [20] J. Hirschauser, "Problems solved and problems created by Kubernetes," Instana, 10 12 2018. [Online]. Available: <https://www.instana.com/blog/problems-solved-and-problems-created-by-kubernetes/>. [Accessed 23 02 2020].
- [21] A. Murty, "Kubernetes: How New Relic Sets You Up for Success," [Online]. Available: <https://blog.newrelic.com/engineering/monitoring-application-performance-in-kubernetes/>. [Accessed 18 02 2020].
- [22] D. Berman, "Top 10 Open Source Monitoring Tools for Kubernetes," logz.io, 4 10 2019. [Online]. Available: <https://logz.io/blog/open-source-monitoring-tools-for-kubernetes/>. [Accessed 12 02 2020].
- [23] "Azure Kubernetes Service (AKS)," Microsoft Azure, [Online]. Available: <https://azure.microsoft.com/en-us/services/kubernetes-service/>. [Accessed 24 02 2020].
- [24] F. Smith and O. Garrett, "What Is a Service Mesh?," NGINX, 03 04 2018. [Online]. Available: <https://www.nginx.com/blog/what-is-a-service-mesh/>. [Accessed 12 02 2020].
- [25] Prometheus Authors, "What is Prometheus," [Online]. Available: <https://prometheus.io/>. [Accessed 21 02 2020].
- [26] Grafana Labs, "What is Grafana ?," [Online]. Available: <https://grafana.com/>. [Accessed 22 02 2020].

- [27] G. Chandra, "Istio Service Mesh Observability with Kiali," 29 7 2019. [Online]. Available: <https://itnext.io/istio-service-mesh-observability-with-kiali-c2ded8a413c4>. [Accessed 22 02 2020].
- [28] "Introduction to Jaeger," Jaeger, [Online]. Available: <https://www.jaegertracing.io/docs/1.6/>. [Accessed 22 02 2020].
- [29] "Microservices," [Online]. Available: <https://en.wikipedia.org/wiki/Microservices>. [Accessed 21 02 2020].
- [30] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis and S. Tilkov, "The Technological Perspective," *Microservices: The Journey So Far and Challenges Ahead*, vol. 35, pp. 24 - 35, 04 05 2018.
- [31] H. Omans, "Adding Consistency and Automation to Grafana," 25 07 2019. [Online]. Available: <https://techblog.commercetools.com/adding-consistency-and-automation-to-grafana-e99eb374fe40>. [Accessed 22 02 2020].

