

A NETWORK SCIENCE BASED APPROACH FOR OPTIMAL MICROSERVICE GOVERNANCE

2020-021

Project Proposal Report

Saranga S.A.G

De Silva N

LS Jayasinghe

M.V. Lakshitha

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

We declare that this is our 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.

Signatures:

| | | |
|-----------------|------------|---|
| Saranga S.A. G | IT17016230 |  |
| De Silva N. | IT17006880 |  |
| L.S. Jayasinghe | IT17012966 |  |
| M.V. Lakshitha | IT17410250 |  |

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


.....
Signature of the Supervisor

23/02/2020
.....
Date

Abstract

Microservices have become one of the most popular development architectures for many software applications developed today, mainly 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 which supports scalability and performance. As a result of this, 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.

Nevertheless, even with the use of microservice deployment tools and platforms such as Kubernetes, a common issue that many developers come across, is an effective approach to evaluate and monitor their effectiveness and viability of their microservice deployment and identify performance bottlenecks. This issue is especially an important factor to be considered in the case when there are a vast number of interconnected microservices in an application.

This research carried out aims to find an effective approach to the above-identified problem through developing a model which will help and provide aid to evaluate, monitor, maintain and oversee microservice deployments through the use of several techniques which include improvement of efficiency through metric analysis gathered based on identified dependency measurements, effective autoscaling through load prediction, a microservice monitoring solution by providing a Business Intelligence dashboard as well as providing solutions to come up with an optimal deployment strategy for the microservice cluster.

Keywords: Auto-scaling, Chaos Engineering, Container, Docker, Kubernetes, Machine Learning, Microservices, Time Series

Table of Contents

| | |
|--|------------|
| Declaration..... | i |
| Abstract..... | ii |
| Table of Contents | iii |
| List of Figures..... | v |
| List of Tables | vi |
| List of Abbreviations | vii |
| 1.0 INTRODUCTION..... | 1 |
| 1.1 Background and Literature | 4 |
| 1.2 Research Gap | 7 |
| 1.3 Research Problem | 9 |
| 2.0 OBJECTIVES | 11 |
| 2.1 Main Objective | 11 |
| 2.2 Specific Objectives | 11 |
| 3.0 METHODOLOGY | 12 |
| 3.1 Requirement Gathering..... | 12 |
| 3.1.1 Past Research Analysis | 12 |
| 3.1.2 Identifying Existing Systems..... | 12 |
| 3.2 Feasibility Study..... | 13 |
| 3.2.1 Technical Feasibility | 13 |
| 3.2.2 Schedule Feasibility | 14 |
| 3.2.3 Economic Feasibility | 14 |
| 3.3 Requirement Analysis..... | 15 |
| 3.4 System Analysis..... | 16 |
| 3.4.1 Software Solution | 16 |
| 3.5 System Development and Implementation | 18 |
| 3.5.1 Building the Dependency Network..... | 20 |
| 3.5.2 Building the Load Prediction Component | 21 |
| 3.5.3 Building the Resiliency Evaluation System..... | 22 |
| 3.5.4 Building the Final Optimization Algorithm | 23 |
| 3.6 Project Requirements | 24 |
| 3.6.1 Functional Requirements | 24 |

| | |
|--|----|
| 3.6.2 Non-Functional Requirements | 24 |
| 3.7 Testing | 25 |
| 3.8 Time Line | 26 |
| 4.0 PERSONNEL AND FACILITIES | 27 |
| 5.0 COMMERCIALIZATION | 31 |
| 6.0 BUDGET | 32 |
| 7.0 SUMMARY | 33 |
| 8.0 REFERENCES | 34 |

List of Figures

| | |
|--|----|
| Figure 1.1 Kubernetes Logo..... | 3 |
| Figure 1.2 Microservice Technology Timeline..... | 4 |
| Figure 3.1 System Overview Diagram..... | 16 |
| Figure 3.2 Proposed Methodology | 18 |
| Figure 3.3 Gantt Chart | 26 |

List of Tables

Table 4.1 Personnel and Resources..... 27

Table 6.1 Budget 32

List of Abbreviations

| Abbreviation | Description |
|--------------|---|
| AKS | Azure Kubernetes Services |
| APM | Application Performance Monitoring |
| AWS | Amazon Web Service |
| IT | Information Technology |
| SOA | Service-Oriented Architecture |
| TOSCA | Topology and Orchestration Specification for Cloud Applications |

1.0 INTRODUCTION

Microservices have become one of the most popular development architectures for many software applications developed today. This is mainly because microservices provide an effective way to develop a software application as a set of interconnected modular services that can be independently deployed and scaled [1]. Furthermore, due to benefits such as easy integration and deployment, support for continuous delivery as well as improved fault isolation, many organizations, including several well-known organizations such as Amazon, Netflix, and eBay have migrated from their traditional monolithic architecture and instead embraced microservice architecture [1].

With the widespread use of microservices, a variety of tools and platforms have been developed in order to deploy microservices efficiently. Among them, containerization of microservices is the most widely used and preferred approach. Containerization enables efficient and easy deployment of microservices through enabling developers and system administrators the ability to package and deploy microservices as container images through opensource tools such as Docker [2-3].

Furthermore, since all required dependencies, libraries, and all other components needed for the microservice are 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 [4]. However, in the case when there are multiple containers present in a particular application, it becomes difficult to coordinate, schedule as well as monitor the deployed containers and ensure service availability [5]. As a result, Kubernetes was introduced in the year 2014 [6] in order to provide a framework to run such distributed systems more resiliently by providing effective solutions for load balancing, storage orchestration, automated rollouts, rollbacks, self-healing mechanisms, etc. [5]. Integrating Kubernetes into an application's deployment strategy, therefore, enables an organization to easily deploy, manage, and handle an application, keeping the inherent benefits of container-based deployments while ensuring service availability. The existence of this unique

characteristic in Kubernetes has therefore resulted in its widespread use in microservice deployments and the increased popularity which can be seen today.

However, even though the use of deployment tools like Kubernetes, there are still some issues that need to be addressed, particularly concerning the deployment policies followed. 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 are deployed far away, then issues such as latency could be a problem.

In addition, there also some issues that exist in the current way in which the effectiveness of Kubernetes deployments is evaluated with respect to performance. Currently, the process being used by developers and system administrators uses a wide array of monitoring tools known as Application Performance Monitoring (APM) tools in order to gather performance metrics regarding the various resources in the Kubernetes cluster and get an idea of the performance of their deployed clusters. This task is usually performed through constant observation of the metrics gathered through these APM tools. This technique is also used in key tasks such as in the identification, setting, and configuring of resource limits and thresholds for resource utilization as well as in the identification of potential performance bottlenecks. Nevertheless, this approach of constant monitoring and observation in order to set up parameters and thresholds related to performance optimization is quite inefficient due to the constant amount of observation required.

Furthermore, even though these APM tools help in the identification of potential problems in a deployment, they are not able to provide an in-depth insight as to why the problem occurred. Therefore, it is quite clear that developers and system administrators are not able to easily obtain a holistic view of the performance of their deployed cluster and get a clear understanding regarding the current and expected performance of their deployments.

This research aims to provide a solution to this problem by taking the initial step by aiming to develop a model that takes into consideration a metric analysis-based approach to optimize and govern microservice deployments in Kubernetes through evaluation of identified dependency measurements. The proposed model will ultimately aid developers and system administrators to effectively govern their microservice deployments by providing a holistic view, enabling them to configure their deployments with minimal time and effort.



Figure 1.1: Kubernetes Logo

1.1 Background and Literature

The term “microservices” was first introduced in 2011 [7,8]. It was considered as a specialized implementation of Service Oriented Architecture (SOA) and introduced to denote the common architectural approach of decomposing applications into smaller self-contained services and, in turn, develop loosely coupled services. Later on, it was widely adopted by many companies such as Amazon, Netflix, LinkedIn, and SoundCloud as a result of the traditional monolithic applications being hard to develop, maintain and scale [9].

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 1.2. According to the authors of [8], the development of microservice-based technologies came in the form of ten “waves” starting from lightweight container technologies like Docker and LXC and leading up to the last “wave” of service mesh technologies such as Linkerd and Istio.

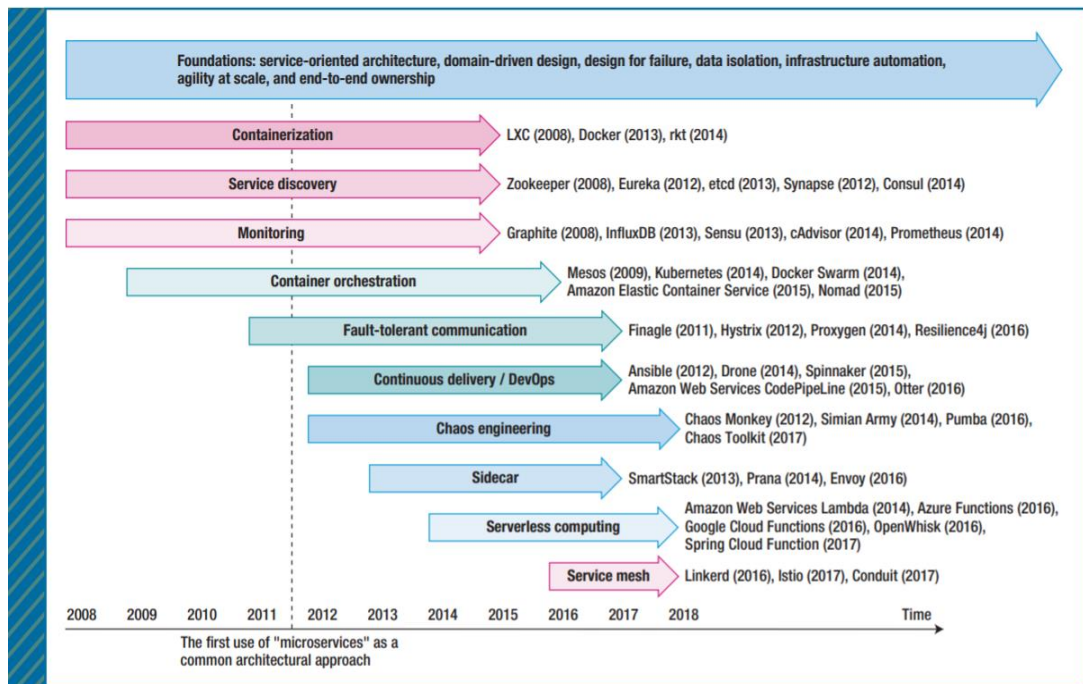


Figure 1.2: Microservice technologies timeline [8]

As evident from Figure 1.2 given above, with the introduction of various technologies supporting microservices throughout the years, microservices became an increasingly popular development architecture mainly due to several advantages it possesses when compared to the traditional monolithic applications developed. Key advantages include improved scalability, maintainability, delivery, and greater performance. Furthermore, since microservices are developed in a loosely coupled manner, it has enabled developers to develop, deploy, version and scale applications independently, bringing in benefits such as faster delivery, greater performance, and greater autonomy [8].

Another key difference when between microservice and monolith architecture is when it comes to how they are governed. This term is known as “Microservices Governance” concerning a microservice architecture, and it can be simply defined 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 are made “top-down” [10], whereas governance in microservices embraces a decentralized governance approach. This, in turn, enables microservices to make use of a polyglot model technology stack in the development of applications.

However, due to the decentralized approach used in the governance of microservices more steps should be taken in order to ensure effective governance is maintained since typical applications require interconnections between a vast number of microservices where business process workflows are continuously introduced, Therefore, in order to ensure an effective microservice governance is in place, organizations currently make use of a variety of tools which facilitate tasks such as monitoring, autoscaling, configuration management, fault tolerance, etc. as seen in some of the examples provided in Figure 1.2 above.

Throughout the years, several strategies have been proposed by research a variety of research publications in order to minimize this issue and optimize performance. These researches primarily focus on performance modeling strategies as well as improving

processes on orchestration platforms such as Kubernetes. In this regard, some of the key researches that propose relevant models and methodologies are highlighted below.

Research publications such as [11], propose an architectural approach along with its implementation that federates Kubernetes clusters using a TOSCA-based cloud orchestration tool, whereas research publications such as [12] proposes a tool named Terminus to solve the problem of finding the best-suited resources for the microservice to be deployed, so that the whole application achieves the best performance while minimizing the resource consumption.

Other key researches include the reference net-based model for pod & container lifecycle in Kubernetes proposed by the authors of [13] and the generative platform for benchmarking performance and resilience engineering approaches in microservice architectures as proposed in [14]. This platform comprised of elements such as an underlying metamodel, generation platform, supporting services for workload generation, problem injection and monitoring.

However, through analyzing the above research publications described above, it is quite clear that there is no current solution proposed that takes into consideration an integrated modeling strategy, and takes into consideration elements such as co-dependencies present as well resilience and centrality measures among microservices when developing a holistic orchestration policy for Kubernetes based microservice deployments, as proposed in this research.

1.2 Research Gap

The primary research gap that this research aims at fulfilling is the gap that exists in current methodologies used in the governance and optimization of microservice deployments, particularly concerning Kubernetes. A thorough analysis of published research papers throughout the years has managed to highlight some of the issues that are present in current microservice governance methodologies and thereby enabled in the identification of the research gap.

The initial inspiration for the identification of this research problem and also the objectives of this research can be found in [15]. This publication clearly describes some of the key challenges faced in the deployment of microservices and the need for APM tools, especially those deployed in containers to include additional measures to monitor microservices such that they could be used as input for resilience mechanisms and creation of auto-scaling policies. This publication also goes on to highlight the fact that the shift in use cases for microservice performance modeling, particularly with respect to design-time performance modeling, has moved on from the traditional use cases such as capacity planning to newer emerging areas such as reliability and resilience engineering. Furthermore, the need for new modeling strategies that capture the recent advances in deployment technology such as Kubernetes, and also some of the key challenges that have to be faced, and the fact that making use of techniques from machine learning could solve some of these challenges, is also clearly highlighted in this publication.

Research publications such as [16] state the need for the development of techniques for accurately modeling, representing, and querying configurations of microservices and data center resources in a container and hypervisor-based technology. This publication states that the inability of monitoring frameworks such as Amazon EC2 and Heapster in Kubernetes to measure microservice-level performance metrics, will lead to the creation of several new research topics which include the development of holistic techniques for collecting and integrating monitoring data from microservices and datacenter resources, and, that users such as administrators or a computer program

such as a scheduler could track and understand the impact of runtime uncertainties (failure, load-balancing, overloading, etc.). on performance without understanding the whole platform's complexity.

The publication [8] discusses the evolution of microservice-based technologies throughout the years and some of the challenges that are yet to be addressed in the future. Among the challenges stated that need to be addressed is the challenge faced in the resource monitoring and management process currently in use. Issues such as the overloading of monitoring events, which hinders effective decision making and management decisions, as well as making use of past actions and events to better inform resource management decisions, are clearly highlighted in this regard.

Based on the information derived from research publications mentioned above, it is clear that new modeling strategies for microservice governance should be explored which should incorporate a more effective and efficient use of monitoring solutions as well integration with other technologies such as resiliency evaluation and autoscaling in order to improve upon and develop a more optimized governance model for microservice deployments.

1.3 Research Problem

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

Currently, the only way in which developers and system administrators can effectively evaluate the effectiveness of their Kubernetes deployments with respect to performance is through the use of a wide variety of monitoring solutions provided via Application Performance Management (APM) tools. This process is quite challenging due to the fact that developers and system administrators have to focus on a variety of factors such as resiliency, security, auto-scaling, etc. which are obtained from these vast arrays of monitoring sources before deciding the optimal deployment configuration for the cluster in order to ensure the optimal performance. Examples of this could also be seen in processes such as resource utilization as well [17,18].

Furthermore, due to the use of vast arrays of APM tools that are used in these instances, it is also difficult to identify potential performance bottlenecks and identify the root cause for these problems, since developers and system administrators are unable to easily get a holistic view the status of their deployments regarding the expected performance of the deployed cluster unless they constantly monitor and analyze the vast amount of metrics obtained by these APM tools [19-21].

Also, with the current trend moving towards cloud-based platforms provided by Azure, Google, and AWS to perform Kubernetes deployments, which primarily incur a cost based on resource utilization [22-24], the need for APM tools has become even more evident. This, in turn, has led developers and system administrators to be more mindful in maintaining a proper balance between performance and resource utilization. However, maintaining this balance between performance and resource utilization is quite a challenging and time-consuming process for developers and system

administrators since this process requires constant monitoring of metrics and finetuning multiple variables and parameters in their deployment configurations such that the optimal performance criteria are met. [25]

Adding to this set of problems is the difficulty posed in successfully configuring and integrating these APM tools with the existing tools used by organizations [26]. According to a survey conducted by the Cloud Native Computing Foundation (CNCf) [27] regarding the challenges users faced in Kubernetes deployments as well as a survey of 1,300 attendees at the KubeCon CloudNativeCon 2019 [28], monitoring was stated as one of the top-ranked challenges faced.

Therefore, based on the issues mentioned above, as well as the related research gaps highlighted in the previous section, it is quite clear that a more suitable solution should be implemented to enhance the governance of microservice-based deployments. This solution should make use of a unified model, taking into account the variety of factors considered in configuring Kubernetes deployments, to help system administrators and developers to get a holistic view of the status of the current deployment and enable them to easily govern and configure their deployments with minimal time and effort.

2.0 OBJECTIVES

2.1 Main Objective

The main objective is to model a network science-based approach to govern microservice deployments through evaluation and analysis of metrics gathered and ultimately come up with a proposed model that aids in optimizing microservice deployments.

2.2 Specific Objectives

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

- To increase the efficiency of microservices deployments by applying the metrics used in network analysis, such as centrality and resilience measures, and link predictions on identified dependency measurements.
- To develop an improved auto-scaling policy for a deployment, based on load prediction.
- To develop a business intelligence dashboard to evaluate performance and monitor microservice deployments.
- To identify key factors that lead to performance reduction in microservice deployments and come up with an optimal deployment strategy.

3.0 METHODOLOGY

3.1 Requirement Gathering

Requirement gathering was mainly performed 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.

3.1.1 Past Research Analysis

The past research analysis process was mainly performed through reading and analyzing a wide array of research publications published through recent years. Key topics of interest included microservice deployment optimization, microservice performance engineering, microservice governance, centrality evaluation, load prediction, and forecasting, resource prediction and optimization, resiliency analysis, and microservice monitoring. During the research analysis process, 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 performance optimization in microservices.

3.1.2 Identifying Existing Systems

A thorough analysis was conducted on a variety of existing APM tools as well as other similar systems, that were available to use with the Kubernetes platform. This process was mainly done by visiting the various online sources and analyzing the available documentation and videos published. During this process, the primary focus was given in identifying the key features and drawbacks that were present in the tools analyzed.

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 are required to 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 are required to 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 Knowledge on Resiliency Evaluation and Chaos Engineering

For the resiliency evaluation system to be developed, a thorough knowledge of “Chaos Engineering” and knowledge related Chaos Engineering tools are required. Furthermore, members should have sufficient knowledge of configuring and integrating the selected Chaos Engineering tool with the optimization model.

3.2.1.4 Machine Learning Knowledge

In order to develop the proposed optimization model, all members are required to 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.

3.2.2 Schedule Feasibility

The proposed project should be able to be implemented within the scheduled time period of about five months, with about two months allocated for research, requirement gathering, and analysis. Finally, the proposed project should be completed within the end of 7 months, including sufficient testing.

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. This is mainly due to the fact that most APM tools and solutions offered currently with respect to Kubernetes are often opensource.

3.3 Requirement Analysis

During the requirement analysis phase, key information obtained during the requirement gathering phase is analyzed. Analyzing the gathered information will prove to be of utmost importance to the research process, since key information regarding the potential challenges that may be faced, the potential complexity of tasks involved, as well as other key information regarding the tools used by other research teams will be easily identifiable.

Also, since the research carried out a software-based approach, by performing requirement analysis, key information regarding the schedule, technical and economic feasibility was realized and helped in aligning research goals such that the research carried out does not exceed the technical skills of the research members while maintaining the expected deadlines.

In the research paper analysis, the primary focus will be given to the analysis of the methodology and tools used, as well as the outcomes of the research conducted. This helps in improving the decision-making process in the current research by providing credible evidence that will help in deciding upon the direction in which the current research should progress by highlighting the research gaps.

Furthermore, analyzing the online resources regarding the available tools will help in the identification of the existing tools that possess similar features to what the current research aims at implementing and help in identifying the research gap by comparing the existing feature with those that are proposed. Also, by analyzing tools that could prove to be of use in the implementation of the current research, a clear idea regarding the features they possess, and how they could be integrated into the current research could be identified.

3.4 System Analysis

3.4.1 Software Solution

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

- Dependency network
- Load prediction
- Resiliency evaluation
- Optimization algorithm

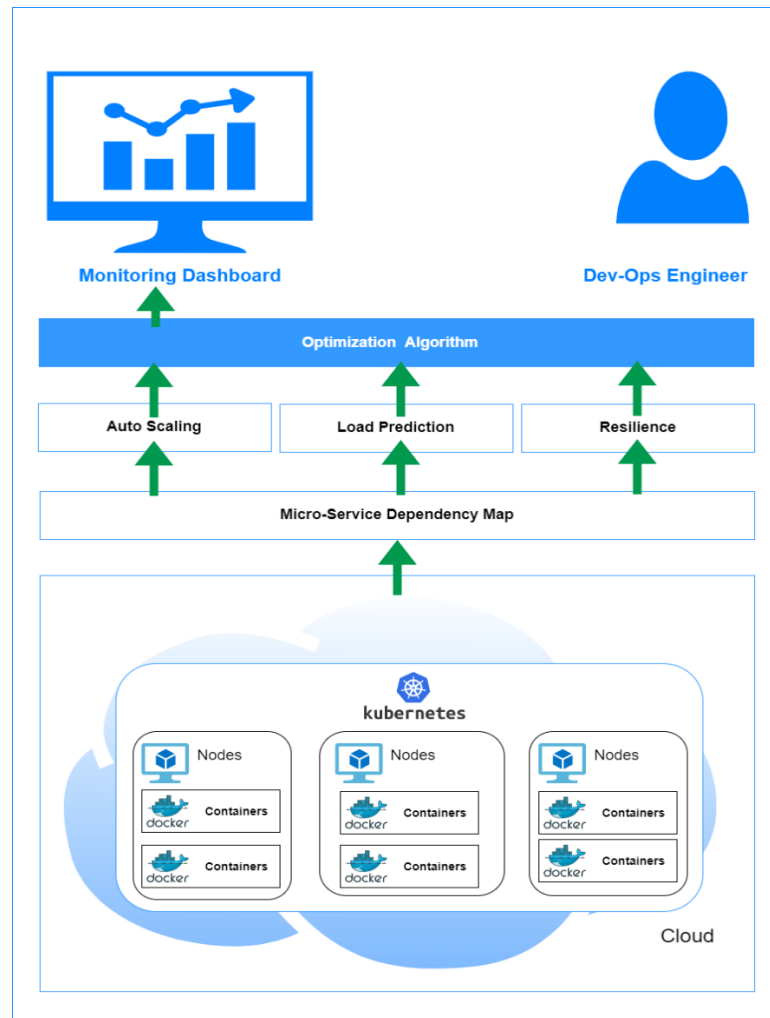


Figure 3.1: System Overview Diagram

3.4.1.1 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. The level of dependency among microservices is calculated by using a selection of APM tools to retrieve metrics from microservices to determine the level of dependency among the microservices in order to develop a co-dependency network between the microservices.

3.4.1.2 Load Prediction Component

This component is responsible for the prediction of the load through analysis of load-based resource metrics and the evaluation of centrality measures derived from the predicted loads. This component will primarily make use of statistical and machine learning techniques utilizing a time series prediction model for the process mentioned above. 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.

3.4.1.3 Resiliency Evaluation Component

This component is responsible for the evaluation of resiliency in the microservices identified using dependency measures created using the dependency network. The resiliency will be evaluated by performing targeted attacks on the identified microservices using the Chaos Monkey tool. This component will also enable identification of key microservices within a cluster based on the evaluated resiliency measures.

3.4.1.4 Optimization Algorithm

This component is responsible for determining the optimal deployment strategy for the cluster based on inputs from the load prediction and resiliency analysis components described above. The optimal strategy determined by this optimization algorithm will then be displayed in the dashboard to the user.

3.5 System Development and Implementation

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

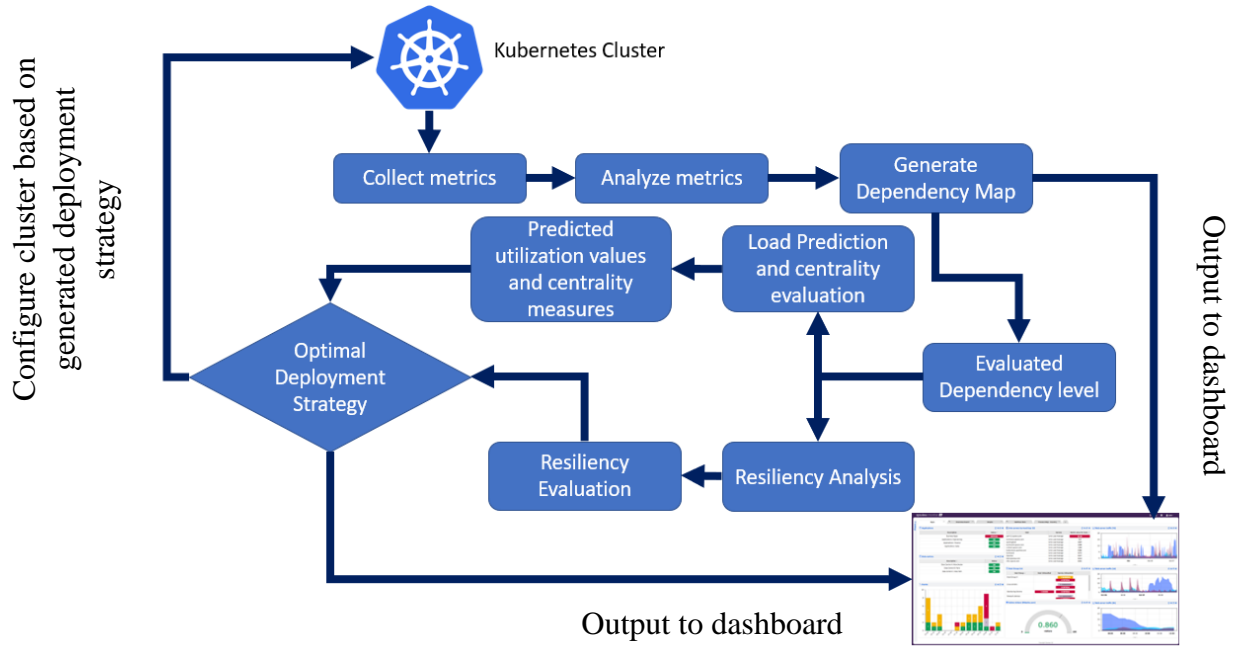


Figure 3.2: Proposed Methodology

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

1. Creation of a Kubernetes cluster in which the proposed model could be implemented and tested.
2. Retrieval and gathering of metrics from the deployed cluster.
3. Generation of a dependency network that depicts the level of dependency among the deployed microservices.
4. Making use of the dependency level among the deployed microservices obtained from the previous step, two solutions that would ultimately aid in optimizing the current deployment will be implemented.
 1. A load prediction algorithm will be developed through the analysis of the metrics in order to determine the centrality of a particular microservice based on the predicted loads.
 2. A system which evaluates and analyses the resiliency of microservices in the cluster by performing targeted attacks to evaluate the resiliency.
5. Then, all findings from the above two developed components will be fed into an optimization algorithm which will be used to come up with the optimal deployment strategy for the cluster.
6. Lastly, the resulting output will be integrated into a dashboard such that the user will be able to receive potential suggestions which will help to optimize their current deployment.

3.5.1 Building the Dependency Network

The dependency network and the service mesh will be implemented by reading the logs and metrics for all the nodes and the pods in a Kubernetes cluster. An open-source independent service mesh called "Istio" will be configured and deployed to the Kubernetes cluster. Istio will create a proxy on top of every node, and it will be able to capture the network requests and responses of a particular node.

The whole research will be based on quantifying the dependency between two microservices. To quantify the dependency, all the network responses and requests between two microservices will be recorded with the timestamp. Then the quantity of network traffic can be calculated for a unit of time.

Metrics such as CPU utilization, memory allocation, network usage, and disk utilization will be needed for the load prediction, and the auto-scaling part of the research and the metrics mentioned above will be gathered by configuring Prometheus and Kiali. Then the gathered metrics will be added to the dependency network.

Tools

- Azure CLI
- Kubernetes CLI (Kubectrl)
- Docker
- Istio

Programming Languages

- Python
- Node JS
- Spring Boot
- React JS

3.5.2 Building the Load Prediction Component

The load prediction component will primarily make use of data obtained from the co-dependency network in order to perform the prediction process. This data will include load-based metrics such as CPU utilization, network utilization, as well as other metrics used in the development of the co-dependency network. The obtained metric data will then be recorded and stored in a temporary database within the cluster for analysis. After proper data cleaning processes are performed, the data will then be converted into a time series model, which will be sent to the prediction model for the time series forecasting process. This process will be performed using the Python programming language as well as using supplementary tools such as Jupyter Notebook and Anaconda. Open-source libraries and tools such as TensorFlow are also expected to be used to facilitate the prediction process. Using the predicted load derived from the forecasted time series model, centrality measures will be evaluated from the microservices based on the predicted loads. Here too, libraries such as NetworkX will be used for the centrality evaluation process.

Tools

- Anaconda
- Jupyter Notebook

Programming Languages

- Python

Key Python libraries expected to be used

- Numpy
- Scikit-learn
- TensorFlow
- Pandas
- Matplotlib
- NetworkX

3.5.3 Building the Resiliency Evaluation System

The resiliency evaluation component will be done using the concept of Chaos Engineering with the help of tools such as Chaos Monkey, Failure Injection Testing, Latency Monkey, and Doctor Monkey. With the use of the dependency network, main nodes are identified, and targeted attacks are performed on these nodes to evaluate how it will affect other nodes and functions of the system. Data collected will contain performance metrics such as CPU utilization, IO events, Disk, Memory, and business metrics such as orders per second or lower-level metrics like response latency and response error rate.

From the metrics obtained by the evaluation, the most important metrics will be selected and prioritized. The final evaluation details will be forwarded as an input to the final optimization algorithm to create an optimal deployment strategy.

Tools

- Chaos Monkey
- Doctor Monkey

Programming Languages

- Python
- NodeJS

Libraries expected to be used

- Chaos Toolkit
- Spinnaker

3.5.4 Building the Final Optimization Algorithm

The optimization algorithm will be used for generating a better microservices deployment strategy based on load prediction-based centrality and resiliency measures. This algorithm ensures that microservice deployment possesses optimal performance and availability. To implement this algorithm, a genetic algorithm will be used instead of the typical brute force optimization methods used. In this scenario, the fitness function is considered as the most important part of the genetic algorithm since it defines how to score a given deployment plan. The scored method is mainly based on the load prediction measures and network latency. Also, it is concerned about node resource power, microservices resource consumption, and resiliency of the microservice architecture. For example, if most communicated pairs of microservices deployed in nearby nodes, that is a point that gains a higher score. Among the variety of random deployment plans, the algorithm will select the highest scored plan.

However, the genetic algorithm does not guarantee the generation of the best plan, but it always generates a better plan. After selecting the better strategy plan, the system can perform the automated deployment. To obtain node resource power and microservice resource consumption, Istio will be used. Python will be used for implementing the algorithm whereas libraries such as NetworkX be used for dealing with the node latency map. Python GA library will be used to perform the genetic algorithm part.

Tools

- Istio

Programming Languages

- Python

Key Python libraries expected to be used

- NetworkX
- GA

3.6 Project Requirements

3.6.1 Functional Requirements

The functional requirements for the proposed model are as follows:

- Users should be able to view the optimal deployment strategy for a given cluster.
- Users should be able to view possible deployment suggestions in order to optimize the deployment.
- The system should analyze the resiliency of the identified microservices in the cluster.
- Load prediction should be performed on the identified microservices.
- The auto scaler should be configured to proactively auto-scale based on predicted load.
- The optimal deployment strategy for a particular deployment should be determined using the predicted load, evaluated centrality measures and resiliency evaluation.

3.6.2 Non-Functional Requirements

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

- Usability
- Reliability
- Availability
- Interoperability
- Performance

3.7 Testing

The testing phase is a key phase in the development process of this research, and it is expected to be conducted throughout the implementation process of this research project. Initial testing will be performed on a component level basis in which each component will be tested individually, and once a specific component has been developed, the testing process will begin. During component-level testing, key features and functions developed will be evaluated to ensure they are functioning as required. Once all components have been completely developed, integration testing will be performed, and the final developed model will be evaluated to ensure the overall performance is improved in the microservice cluster through the use of the proposed governance model. Here, key factors such as latency and its related measures will be used to evaluate the performance of the proposed governance model.

For this research, a test-bed of consisting of about 50 microservices is expected to be deployed in a Kubernetes cluster. Furthermore, a load-generation tool is expected to be used to generate sufficient metrics in order to develop the co-dependency network. These captured metrics will then be used in the load prediction and resiliency evaluation components for further analysis.

3.8 Time Line

The proposed timeline for the project is as follows.

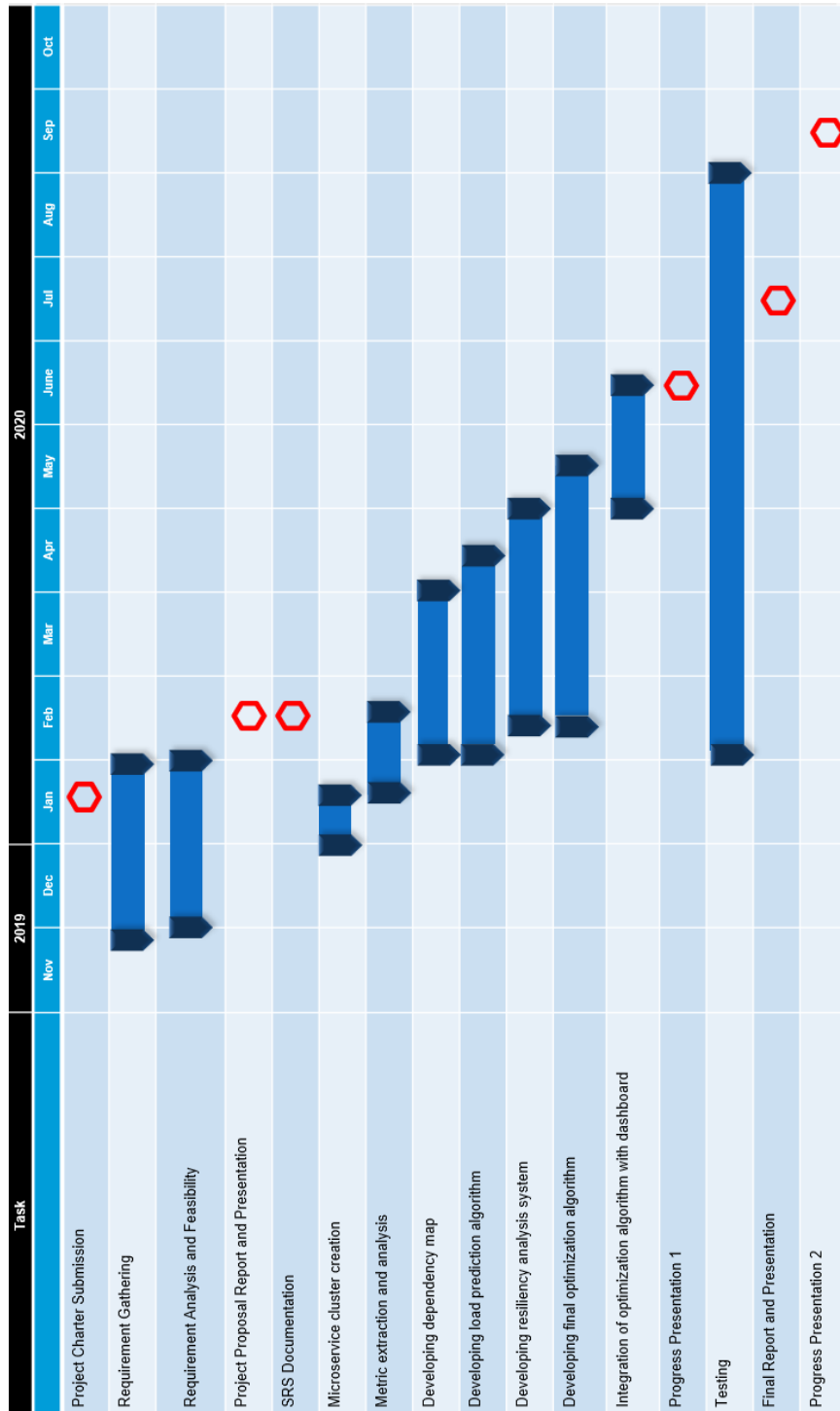


Figure 3.3: Gantt Chart

4.0 PERSONNEL AND FACILITIES

| Name | Key Tasks |
|----------------|--|
| Saranga S.A.G. | <ul style="list-style-type: none">• Setting up and configuring the initial Kubernetes microservice cluster for metric analysis.• Setting up and configuring the relevant tools such as Prometheus and Istio etc. in order to retrieve the required metrics.• Identify and retrieve the required metrics from the various metrics analyzing tools.• Analyze the retrieve metrics such that it could be used to determine the interdependencies between the deployed microservices.• Development of custom metrics using the retrieved metrics which can be used to represent the interdependency between microservices.• Development of a solution in order to display the interdependencies gathered using metrics in order to display as a dependency network. |

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

| | |
|---------------|---|
| | <ul style="list-style-type: none"> • Finetune the developed model in order to get the most accurate prediction. |
| LS Jayasinghe | <ul style="list-style-type: none"> • Retrieval of the analyzed outputs from the load prediction, dependency analysis map, as well as resiliency evaluation process. • Formatting and data manipulation of the data received from the above-mentioned output processes, in order to be fed as inputs to the optimization algorithm. • Development of final optimization algorithm making use of Machine Learning, which takes in inputs from the load prediction, dependency analysis map, and resiliency evaluation process, and proposes the suggested optimal deployment strategy. • Integrate with the UI dashboard in order to display the optimal deployment strategy to the user. • Evaluate the effect of the developed optimization algorithm with respect to the performance of the microservice cluster. |

| | |
|----------------|--|
| Lakshitha M.V. | <ul style="list-style-type: none"> • Using the dependency network, perform the development of a system that analyses the resiliency of the cluster using “Chaos Engineering” tools such as “Chaos Monkey.” • Identification of the microservices which show low resilience from the above-mentioned process. • Evaluating the deployed microservice cluster and coming up with proposed suitable measures in order to improve the resiliency of the deployed microservice cluster. • Evaluating the effect of the proposed resiliency measures with respect to the performance of the microservice cluster. • Development of the UI dashboard, which displays the final findings of the optimization process to the user and the displaying of the optimal deployment strategy to the user. |
|----------------|--|

Table 4.1: Personnel and Resources

5.0 COMMERCIALIZATION

The commercialization of this research project is mainly considered through the development of a tool through the use of the proposed model. The developed tool will be developed as a Business Intelligence Dashboard which makes 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.

The developed Business Intelligence Dashboard will allow users to access all the above-mentioned features and provide a holistic view of their deployments. Hence, this tool will be mainly targeted to be marketed as an APM tool for Kubernetes deployments for system administrators and developers. Due to the wide variety of APM tools currently available in the market which are mostly free and opensource, the initial plan is to develop this dashboard into an opensource tool in order to enter the current market space effectively. However, throughout the years, a freemium based marketing strategy will be adopted with the inclusion of additional features.

6.0 BUDGET

The main aspect of this research is primarily focused on the development of an optimization model that aids in the deployment of microservices through Kubernetes, and hence it is primarily a software-based solution with no inclusion of external hardware.

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

| | |
|------------------------------|-----------|
| Internet use and web hosting | 15500 LKR |
| Publication costs | 6500 LKR |
| Stationary | 5500 LKR |
| TOTAL | 27500 LKR |

Table 6.1: 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 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.

The model is developed primarily through a microservice dependency network based on metric analysis, and performing load prediction, and resiliency evaluation on the microservices identified using the dependency network. Finally, the outputs of these components will be fed into an optimization algorithm and display an optimal deployment strategy to the user.

The proposed model will ultimately be able to provide developers and system administrators an overview of their current deployment configurations with respect to performance and aid in governing their microservice deployments such that the optimal performance is achieved.

References

- [1]"Benefits of Microservices Architecture Implementation - DZone Microservices", dzone.com, 2020. [Online]. Available: <https://dzone.com/articles/benefits-amp-examples-of-microservices-architectur>. [Accessed: 15-Feb-2020].
- [2]"Microservices in Practice: From Architecture to Deployment - DZone Microservices", dzone.com, 2020. [Online]. Available: <https://dzone.com/articles/microservices-in-practice-1>. [Accessed: 15-Feb-2020].
- [3]"What is a Container? | Docker", Docker, 2020. [Online]. Available: <https://www.docker.com/resources/what-container>. [Accessed: 15-Feb-2020].
- [4]"What is Docker?", Opensource.com, 2020. [Online]. Available: <https://opensource.com/resources/what-docker>. [Accessed: 15-Feb-2020].
- [5]"What is Kubernetes", Kubernetes.io, 2020. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>. [Accessed: 15-Feb-2020].
- [6]"Kubernetes", En.wikipedia.org, 2020. [Online]. Available: <https://en.wikipedia.org/wiki/Kubernetes>. [Accessed: 15-Feb-2020].
- [7] "Microservices," *Wikipedia*, 13-Feb-2020. [Online]. Available: <https://en.wikipedia.org/wiki/Microservices> [Accessed: 15-Feb-2020].
- [8] Jamshidi, P., Pahl, C., Mendonca, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The Journey So Far and Challenges Ahead. IEEE Software.
- [9] M. Kalske, N. Mäkitalo, and T. Mikkonen, "Challenges When Moving from Monolith to Microservice Architecture," *Current Trends in Web Engineering Lecture Notes in Computer Science*, pp. 32–47, 2018.
- [10] G. K. Behara, "Microservices Governance: A Detailed Guide," *Enterprise Architecture Management*, 31-Jan-2019. [Online]. Available: <https://www.leanix.net/en/blog/microservices-governance>. [Accessed: 15-Feb-2020].

- [11] D. Kim, H. Muhammad, E. Kim, S. Helal, and C. Lee, "TOSCA-Based and Federation-Aware Cloud Orchestration for Kubernetes Container Platform," *Applied Sciences*, vol. 9, no. 1, p. 191, Jul. 2019.
- [12] A. Jindal, V. Podolskiy, and M. Gerndt, "Performance Modeling for Cloud Microservice Applications," *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering - ICPE 19*, 2019.
- [13] V. Medel, O. Rana, J. Á. Bañares, and U. Arronategui, "Modelling performance & resource management in Kubernetes," *Proceedings of the 9th International Conference on Utility and Cloud Computing - UCC 16*, 2016.
- [14] T. F. Düllmann and A. V. Hoorn, "Model-driven Generation of Microservice Architectures for Benchmarking Performance and Resilience Engineering Approaches," *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion - ICPE 17 Companion*, 2017.
- [15] Heinrich, Robert & van Hoorn, André & Knoche, Holger & Li, Fei & Lwakatare, Lucy Ellen & Pahl, Claus & Schulte, Stefan & Wettinger, Johannes. (2017). "Performance Engineering for Microservices: Research Challenges and Directions".
- [16] Fazio, Maria & Celesti, Antonio & Ranjan, R. & Liu, Chang & Chen, Lydia & Villari, Massimo. (2016). "Open Issues in Scheduling Microservices in the Cloud. IEEE Cloud Computing".
- [17]"DataDog/the-monitor", GitHub, 2020. [Online]. Available: <https://github.com/DataDog/themonitor/blob/master/kubernetes/monitoring-kubernetes-performance-metrics.md>. [Accessed: 15-Feb-2020].
- [18]"Resource Usage Monitoring in Kubernetes", Kubernetes.io, 2020. [Online]. Available: <https://kubernetes.io/blog/2015/05/resource-usage-monitoring-kubernetes/>. [Accessed: 15-Feb-2020].
- [19] D. Snyder, "How to Overcome Kubernetes Monitoring Challenges", OverOps Blog, 2020. [Online]. Available: <https://blog.overops.com/how-to-overcome-monitoring-challenges-with-kubernetes/>. [Accessed: 15-Feb-2020].

- [20]"Monitoring Kubernetes (Part 1): The Challenges and Data Sources - DZone Performance", dzone.com, 2020. [Online]. Available: <https://dzone.com/articles/monitoring-kubernetes-part-1-the-challenges-and-da>. [Accessed: 15-Feb-2020].
- [21]"Challenges of Monitoring and Troubleshooting in Kubernetes Environments | Sumo Logic", Sumo Logic, 2020. [Online]. Available: <https://www.sumologic.com/blog/troubleshooting-kubernetes>. [Accessed: 15-Feb-2020].
- [22]"Pricing - Container Service | Microsoft Azure", Azure.microsoft.com, 2020. [Online]. Available: <https://azure.microsoft.com/en-us/pricing/details/kubernetes-service/>. [Accessed: 15-Feb-2020].
- [23]"Amazon EKS Pricing - Managed Kubernetes Service", Amazon Web Services, Inc., 2020. [Online]. Available: <https://aws.amazon.com/eks/pricing/>. [Accessed: 15-Feb-2020].
- [24]"Pricing | Kubernetes Engine Documentation | Google Cloud", Google Cloud, 2020. [Online]. Available: <https://cloud.google.com/kubernetes-engine/pricing>. [Accessed: 15-Feb-2020].
- [25]"Which One Should You Prioritize? Kubernetes Performance, Cluster Utilization, or Cost Optimization?", Medium, 2020. [Online]. Available: <https://medium.com/@Mohamed.ahmed/which-one-should-you-prioritize-kubernetes-performance-cluster-utilization-or-cost-optimization-21469263b6a7>. [Accessed: 15-Feb-2020].
- [26]"Kubernetes: The Challenge of Deploying & Maintaining", Techolution, 2020. [Online]. Available: <https://techolution.com/kubernetes-challenges/>. [Accessed: 15-Feb-2020].
- [27] L. Hecht, L. Hecht, and L. Hecht, "The Top Challenges Kubernetes Users Face with Deployment - The New Stack", The New Stack, 2020. [Online]. Available: <https://thenewstack.io/top-challenges-kubernetes-users-face-deployment/>. [Accessed: 15-Feb-2020].

[28] M. Vizard, "Running Kubernetes at Scale Top 2020 Challenge - Container Journal", Container Journal, 2020. [Online]. Available: <https://containerjournal.com/topics/container-management/running-kubernetes-at-scale-top-2020-challenge>. [Accessed: 15-Feb-2020].

