

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

2020-021

Project Proposal Report

M.V Lakshitha.

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

Department of Software Engineering

Sri Lanka Institute of Information Technology  
Sri Lanka

February 2020

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

2020-021

Project Proposal Report

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

Department of Software Engineering

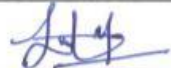
Sri Lanka Institute of Information Technology  
Sri Lanka

February 2020

## Declaration

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

Signature:

IT17A10250	M.V. Lakshitha	
------------	----------------	---

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

Signature of the Supervisor: .....

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

## **Abstract**

Deploying microservices, Kubernetes acts as a management tool which assists to generate a better microservice governance. Advantages of using Kubernetes for microservice deployments are the ability to scale cluster resources based on traffic, improve productivity, security and more stables the application. This abilities of Kubernetes enables developers and system administrators to reduce costs and make use of their deployed microservices more effectively and efficiently.

However, even though the there are many researches and studies done on creating an optimal deployment plan for microservices, the factors they have consdierd are mainly the dependency network distribution, CPU usage, memoy and network bandwith. Futuer pedictions by analyzing these metics are not used to plan an optimal deployment plan and the resiliency of microservices have not been taken into consideration

This research thereby aims to evaluate the resiliency of the microservices using chaos engineering so as to provide an input to come up with an optimal deployment plan for microservice governance

**Keywords:** Kubernetes, , Microservices, Resiliency, Chaos Engineering

## TABLE OF CONTENTS

<b>Declaration.....</b>	<b>Error! Bookmark not defined.</b>
<b>Abstract.....</b>	<b>ii</b>
<b>List of Figures.....</b>	<b>iv</b>
<b>List of Tables .....</b>	<b>v</b>
<b>List of Abbreviations .....</b>	<b>vi</b>
<b>1.0 INTRODUCTION.....</b>	<b>1</b>
<b>1.1 Background and Literature .....</b>	<b>3</b>
<b>1.2 Research Gap .....</b>	<b>5</b>
<b>1.3 Research Problem .....</b>	<b>6</b>
<b>2.0 OBJECTIVES .....</b>	<b>7</b>
<b>2.1 Main Objective .....</b>	<b>7</b>
<b>2.2 Specific Objectives .....</b>	<b>7</b>
<b>3.0 METHODOLOGY .....</b>	<b>8</b>
<b>3.1 Requirement Gathering.....</b>	<b>8</b>
<b>3.1.1 Past Research Analysis .....</b>	<b>8</b>
<b>3.1.2 Identifying Existing Systems.....</b>	<b>8</b>
<b>3.2 Feasibility Study.....</b>	<b>9</b>
<b>3.2.1 Technical Feasibility .....</b>	<b>9</b>
<b>3.2.2 Schedule Feasibility.....</b>	<b>9</b>
<b>3.2.3 Economic Feasibility .....</b>	<b>9</b>
<b>3.3 Requirement Analysis.....</b>	<b>10</b>
<b>3.4 System Analysis.....</b>	<b>11</b>
<b>3.5 System Development and Implementation .....</b>	<b>12</b>
<b>3.6 Project Requirements .....</b>	<b>13</b>
<b>4.0 WORK BREAKDOWN .....</b>	<b>15</b>
<b>5.0 COMMERCIALIZATION .....</b>	<b>16</b>
<b>6.0 BUDGET.....</b>	<b>17</b>
<b>7.0 SUMMARY .....</b>	<b>18</b>
<b>8.0 REFERENCES.....</b>	<b>19</b>
<b>9.0 Appendices.....</b>	<b>21</b>

**List of Figures**

Figure 1.0 outages experienced by popular internet services.....1

Figure 1.1 five phases of chaos engineering.....4

Figure 3.1 System Overview ..... 14

## List of Tables

Table 3.1 Tools and Technology .....	16
Table 5.1 Budget .....	20

## List of Abbreviations

Abbreviation	Description
AWS	Amazon Web Service
FIT	Faliure Injection Testing
HTTP	Hypertext transfer protocol
REST	Representational state Transfer



## 1.0 Introduction

Modern online applications are moving rapidly towards microservice architecture [1], as it supports quick changes and feedbacks to the user. In this architecture, the application is a collection of web services, each serving a single purpose, i.e., a microservice. Each microservice is developed, deployed and managed independently; new features and updates are delivered continuously [2], hundreds of times a day [3]–[5], making the applications extremely dynamic. Microservice applications are typically polyglot: developers write individual microservices in the programming language of their choice, and the communication between services happens using remote API calls. As cloud-native applications, microservices are designed to withstand infrastructure failures and outages, yet struggle to remain available when deployed[6].

Company	Downtime	Postmortem findings
<i>Parse.ly</i> , 2015 [25]	13 hours	Cascading failure due to message bus overload
<i>CircleCI</i> , 2015 [19]	17 hours	Cascading failure due to database overload
<i>BBC</i> , 2014 [18]	48 hours	Cascading failure due to database overload
<i>Spotify</i> , 2013 [26]	Several hours	Cascading failure due to degradation of a core internal service
<i>Twilio</i> , 2013 [28]	10 hours	Database failure caused billing service to repeatedly bill customers

Figure 1.0 A subset of recent outages experienced by popular internet services[9].

In the deployment of microservices, Kubernetes has become the most popular platforms. It has the ability to manage and orchestrate containerized workloads and services. The use of Kubernetes has therefore enabled developers the ability to move on from the traditional deployment methodologies such as physical servers and Virtual Machines and make use of the newer containerized deployment methodologies through the use of open-source tools such as Docker. Furthermore, since Kubernetes provides a vast array of features and services such as service

discovery and load balancing, storage orchestration, self-healing [7], it has managed to become one of the leading container orchestration tools currently available.

Failures are inevitable, even a strongest platform with concrete operations infrastructure can face outage in production, when system's threshold to withstand turbulent conditions go out of control. There is no single reason why a system fail and it is not possible to immediately address a failure without prior knowledge on why and when that specific failure might occur, the same implies to widely used Kubernetes platform. Even when all of the individual services in a Kubernetes environment are functioning properly, the interactions between those services can cause unpredictable outcomes[8].

Resiliency in the sense, it is the ability recover from failures. To evaluate the resiliency of microservices, a concept called Chaos Engineering has been introduced. Chaos Engineering is the discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production[8].Chaos Engineering concept was brought up by Netflix to identify its system weaknesses. FIT tool laid the foundation to build up the Chaos monkey tool which allows to inject failures carefully to the system and examine the behavior[9].

This research aims to evaluate the resiliency of microservices of a system, to generate an input to develop an optimal deployment strategy of microservice governance.

## 1.1 Background and Literature

Popular large scale internet applications such as Netflix, Facebook, Amazon have shown that in order to achieve scalability, robustness and agility, it is important to break a monolithic web application into a collection of web services, called microservices [1]. Each microservice is a simple REST [13] based web service that interacts with other services using HTTP. Modern applications leverage both managed services offered by the hosting cloud platform. To reach loose coupling, microservices use application protocols such as HTTP to provide better integration with other microservices. Each microservice is owned and operated by an independent team of developers. The ability to immediately integrate updates into the production deployment [14] has led to a continuous software delivery model [15], where development teams incrementally deliver features while incorporating user feedback.

To remain available in the face of infrastructure outages in the cloud, a microservice must guard itself from failures of its dependencies. Unpredictable events are bound to happen in a distributed system. Distributed systems contain so many interacting components that the number of things that can go wrong is enormous. Hard disks can fail, the network can go down, a sudden surge in customer traffic can overload a functional component. These kind of events lead to poor performance, and other undesirable behaviors. We'll never be able to prevent all possible failure modes, but we can identify many of the weaknesses in our system before they are triggered by these events. Chaos Engineering is a method of experimentation on infrastructure that brings systemic weaknesses to light. This concept was introduced by Netflix to identify their system weaknesses[9].

In late 2010 Chaos monkey was introduced by Netflix as a tool to evaluate the resiliency of microservices[9]. It randomly selects a running instance of a service and kills it. Through automation, redundancy, fallbacks, and other best practices of resilient design, engineers quickly make the failure scenario irrelevant to the operation of their service.

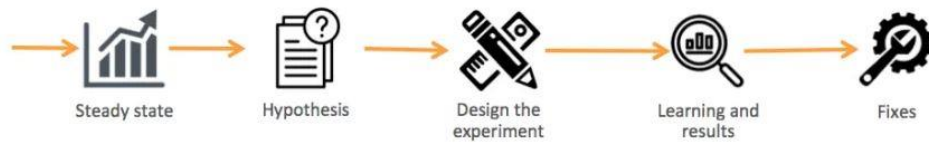


Figure 1.1 five phases of chaos engineering[16]

When analyzing the optimal deployments of microservices, the fact resiliency has not been considered as an important factor[11]. Dependency, CPU usage, disk, memory like factors are considered mainly. Resiliency evaluation can be an important in having an optimal microservice governance.

## **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 detailed 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 enable in the identification of the research gap, as mentioned in the previous statement.

The initial inspiration for the identification of this research problem and also the objectives of this research can be found in [10]. 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.

Microservices deployment strategies have not focused on considering the resiliency of microservices[11]. Eventhough fault tolerance and resiliency evaluations have been done on microservices, the results of them are only used to identify the weaknesses of the system.

This research aims to analyse the resiliency of the microservices and use the results to come up with an optimal deployment strategy to a better microservice governance.

### **1.3 Research Problem**

With the use of Kubernetes for microservice deployment, it provided a good environment to chaos engineering as it has native features for resiliency[16]. As the fact resiliency was not taken into consideration lot of popular service providers suffered with unexpected failures[9].

As a result Netflix came up with the concept chaos engineering introducing tools like chaos monkey, Latency Monkey and FIT[8]. Still the resiliency factor do not involve in the deployment strategies of microservices. Eventhough by performing resiliency tests and being prepared to the failures, still the possibilities of occurring failures are significant.

Therefore based on the issue mentioned above, it is clear that the resiliency factor has to be taken into consideration when planning the deployment strategy of microservices.

## **2.0 OBJECTIVES**

### **2.1 Main Objective**

The main objective of this research is to evaluate the resiliency of microservices via chaos engineering.

### **2.2 Specific Objectives**

The specific objectives for the research project are as follows.

- To develop a suitable environment to perform resiliency evaluation
- Perform health checks on the system to determine whether it is possible to perform targeted attacks on services without interrupting the client.
- Setting up FIT to perform targeted attacks.
- Setting up chaos monkey to perform random attacks on services
- Evaluate the results of the resiliency testing.

## **3.0 METHODOLOGY**

### **3.1 Requirement Gathering**

Requirement gathering for the research was done by analyzing related research publications through online resources. The focal point on this study is to indentify existing or similar developed systems and to study about the methodology used.

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

Past research analysis was primarily performed through reading research publications mainly focused on key areas such as microservices, Kubernetes, resiliency, chaos engineering, chaos monkey and targeted attacks

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 resiliency of microservices and chos engineering.

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

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



## **3.2 Feasibility Study**

### **3.2.1 Technical Feasibility**

Technical feasibility was a key factor considered in the requirements analysis phase of this research project since this project mainly focused on the configuring the required tools to perform resiliency evaluation. A key focus was given in the identification of potential system requirements as well as the required tools and technologies that may be used in performing targeted attacks and defining maetics required.

### **3.2.2 Schedule Feasibility**

The schedule feasibility was also a key factor considered throughout this research. A key focus was given in the identification time periods to set up the required tools to perform resiliency evaluation.

### **3.2.3 Economic Feasibility**

A key focus was given in the identification of the possible costs that might be incurred in the development process, as well as the costs that may be incurred in the use of the planned development tools and technologies.

### **3.3 Requirement Analysis**

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

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

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

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

### 3.4 System Analysis

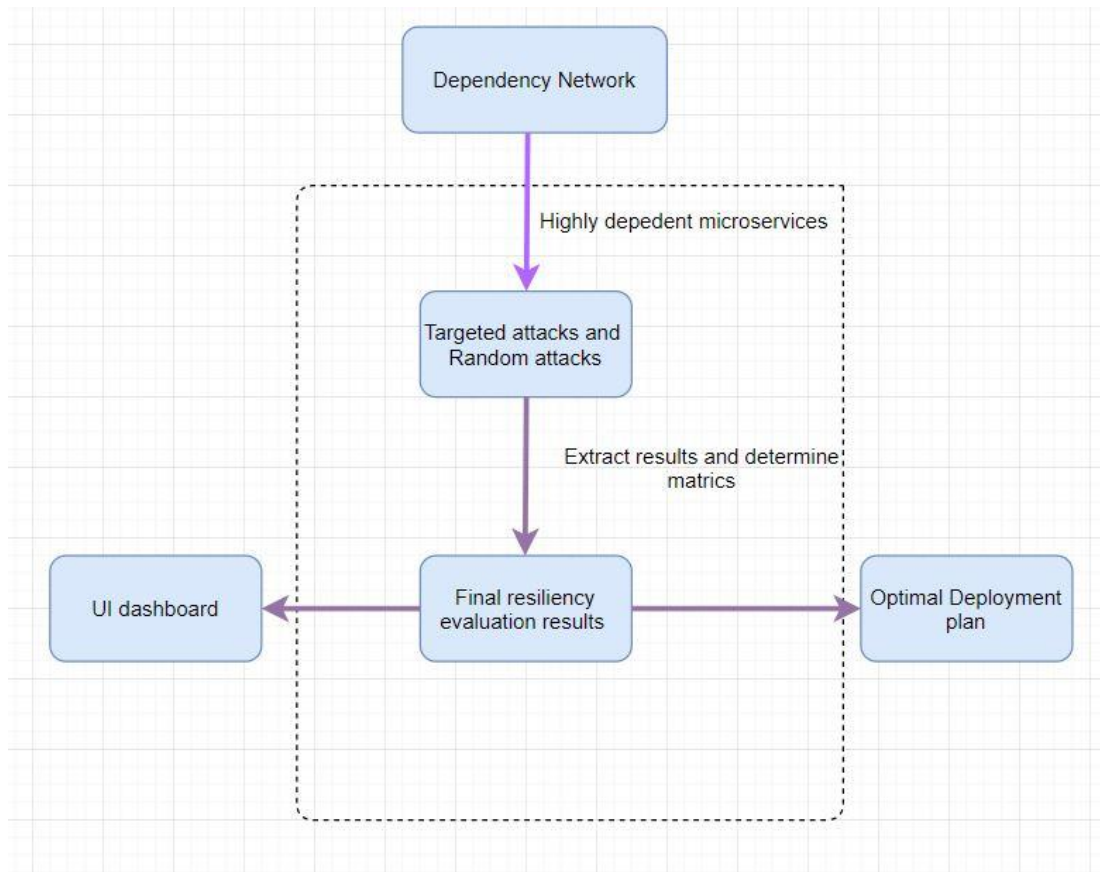


Figure 3.1 System Overview

Using the tools like chaos monkey and IFT, the targeted attacks will be performed to relevant microservices. It is important to perform health checks and find out a suitable time and suitable way to do these attack so as no to interrupt with the availability of the other services.

### 3.5 System Development and Implementation

The steps in the development of the proposed resiliency evaluation are as follows.

1. Obtain the list microservices that show a high level of dependency based on the proposed co-dependency network.
2. Perform health checks in the system before stepping into targeted attacks.
3. Determine how to perform targeted attacks on nodes, without interrupting other functions.
4. Selecting suitable metrics to be analyzed.
5. Perform random attacks using chaos monkey.
6. Perform targeted attacks on highly dependent microservices to observe the behavior using Failure Injection Testing tool.
7. Process the evaluation results to depict a clear idea about the behavior or the importance of certain nodes.
8. Implement UI dashboard to display the results to the user.

Tools	<ul style="list-style-type: none"><li>• Chaos Monkey</li><li>• Failure Injection Testing</li><li>• Doctor Monkey</li></ul>
Programming Languages	<ul style="list-style-type: none"><li>• Python</li></ul>
Libraries	<ul style="list-style-type: none"><li>• Chaos Toolkit</li><li>• Spinnaker</li></ul>

Table 3.1: Tools and Technology

## **3.6 Project Requirements**

### **3.6.1 Functional requirements**

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

- Critical nodes should be identified from the co-dependency network developed.
- Suitable metrics should be identified as needed to planning the deployment strategy.
- Targeted attacks should be performed by not interrupting the customers.
- Evaluation results should be processed to give a meaningful input to create the deployment plan.
- Dashboard UI should show all the information the user needs.

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

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

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

- Interoperability – The proposed system should be able to interact and communicate with the other components proposed in this research and receive inputs as well as forward outputs to the desired components.

## **4.0 WORK BREAKDOWN**

The following are the key tasks that are expected to be performed in the implementation process of this research.

- Examine the dependency network and identify critical nodes
- Preparing Chaos Monkey tool to evaluate resiliency of microservices.
- Finding out the best metrics to evaluate the resiliency.
- Performing targeted attacks on nodes, bridges and in other selected ways.
- Extract the results and process them to a meaningful output.
- Provide the evaluation results to create an optimal deployment plan.
- Development of the UI dashboard, which displays the final findings of the optimization process to the user.

## 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 open source, the initial plan is to develop this dashboard into an open source 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	6000 LKR
Publication costs	2000 LKR
Stationary	2000 LKR
TOTAL	10000 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.

This research, as described above, thereby aims to provide a input to create a optimal deployment strategy, by evaluating the resiliency of the microservices and how it will affect on other services. This helps to deploy microservices in a way which leads to less number of faliures and better resiliency.

## 8.0 References

- [1] S. Neuman, Building Microservices: “Designing Fine-Grained Systems”. O’Reilly Media, February 2015
- [2] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, 1st ed. Addison-Wesley Professional, July 2010.
- [3] “GRUBHUB. Enabling Continuous (Food) Delivery at GrubHub,” DockerCon (2015).
- [4] “HUBSPOT. How We Deploy 300 Times a Day,” <http://product.hubspot.com/blog/how-we-deploy-300-times-a-day>, November 2013, [ONLINE].
- [5] “ORBITZ. Enabling Microservices at Orbitz,” DockerCon (2015).
- [6] Gremlin: Systematic Resilience Testing of Microservices, Victor Heorhiadi UNC Chapel Hill [victor@cs.unc.edu](mailto:victor@cs.unc.edu), Shriram Rajagopalan IBM T. J. Watson Research
- [7] “What is Kubernetes,” Kubernetes. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>. [Accessed: 13-Feb-2020].
- [8] “Improving Kubernetes Resiliency with Chaos Engineering” [Online]. Available: <https://medium.com/faun/failures-are-inevitable-even-a-strongest-platform-with-concrete-operations-infrastructure-can-7d0c016430c6>. [Accessed: 10-Feb-2020]

[9] Chaos Engineering, Building Confidence in System Behavior through Experiments. Casey Rosenthal, Lorin Hochstein, Aaron Blohowiak, Nora Jones, and Ali Basiri

[10] 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".

[11] "Optimal and Automated Deployment for Microservices". Mario Bravetti, Saverio Giallorenzo, Jacopo Mauro, Iacopo Talevi

[12] "Automated Fault-Tolerance Testing". Adithya Nagarajan, Ajay Vaddadi.

[13] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, 2000.

[14] P. M. Duvall, S. Matyas, and A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk, 1st ed. AddisonWesley Professional, June 2007.

[15] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, 1st ed. Addison-Wesley Professional, July 2010.

[16] "How chaos engineering will guarantee the resilience of your services"[Online]. Available:<https://eldermoraes.com/how-chaos-engineering-will-guarantee-the-resilience-of-your-services/>[Accessed:18-February-2020]

## **9.0 Appendices**