

**STREAMLINING SOFTWARE RELEASE PROCESS
AND RESOURCE MANAGEMENT FOR
MICROSERVICE-BASED ARCHITECTURE ON
MULTI-CLOUD
23-193**

Project Proposal Report

Herath H.M.I.P.

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

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

April 2023

**STREAMLINING SOFTWARE RELEASE PROCESS
AND RESOURCE MANAGEMENT FOR
MICROSERVICE-BASED ARCHITECTURE ON
MULTI-CLOUD
23-193**

Project Proposal Report

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

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology
Sri Lanka


April 2023

Declaration

I declare that this is our own work and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning 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 acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature:

IT20125516	Herath H.M.I.P.	
------------	-----------------	---

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

.....
Signature of the Supervisor

.....
Date

Abstract

Microservice-based architectures are becoming increasingly common due to their scalability, flexibility, and dependability. However, managing and deploying microservices on multi-cloud platforms can be complicated and time-consuming, particularly when it comes to releasing new software updates.

The software release process consists of several phases, including development, testing, staging, and deployment. Each step necessitates a unique set of resources and tools, and each stage introduces the possibility of human error. As a result, optimizing and automating the software release process can assist companies in lowering the risk of errors, improving release quality, and speeding up the deployment process. Fully automating the software release process entails using tools and technologies to create, test, and deploy software updates automatically. CI/CD pipelines, which automate the complete release process from code changes to deployment, can help with this. By completely automating the release process, organizations can remove the need for manual intervention, lowering the risk of errors and increasing deployment speed.

Identifying and eliminating inefficiencies in the software delivery process is part of optimizing it. This can be accomplished through the identification of bottlenecks, the automation of repetitive duties, and the improvement of resource utilization. Organizations can cut costs, improve resource management, and boost productivity by optimizing the release process. Several variables must be considered in order to streamline the software release process for microservice-based architectures on multi-cloud platforms. Containerization, orchestration, automation, tracking, and security are examples. Organizations can improve the reliability and scalability of their microservices, reduce downtime, and ensure application security by adopting best practices for each of these factors. The microservice-based applications are large and must be controlled effectively with service mesh management tools such as Istio. When deploying an application on the cloud, Kubernetes is the most common method to use in microservice-based architecture. As previously stated, Docker and Kubernetes are widely used for microservice-based architecture, and this research focuses on the software release process using those tools and technologies.

Overall, this research focuses on helping organizations optimize and automate their software release process for microservice-based architectures on multi-cloud platforms. By doing so, organizations can improve their resource management, reduce costs, and increase efficiency while ensuring the reliability and security of their applications.

Keywords: *Microservice, CI/CD, Containerization, Orchestration, Automation, Tracking, Security, Downtime, Kubernetes, Docker, Istio, Service Mesh, Multi-cloud*

Acknowledgments

I would like to express my gratitude to my supervisor, Dr. Nuwan Kodagoda, and my co-supervisor, Mr. Udara Samarathunga, without whose experience and motivation this study would not have been feasible. In addition, I would like to thank the Department of Computer Science and Software Engineering at the Sri Lanka Institute of Information Technology, as well as the CDAP lecturers and employees, for providing me with the tools and platform to perform this study. Their contributions and help have been invaluable throughout this voyage.

Table Of Content

DECLARATION.....	I
ABSTRACT	II
ACKNOWLEDGMENTS	III
TABLE OF CONTENT	IV
LIST OF FIGURES	VI
LIST OF TABLES	VI
LIST OF ABBREVIATIONS	VIII
1.0 INTRODUCTION.....	9
1.1. BACKGROUND AND LITERATURE	12
1.2. RESEARCH GAP	15
1.3. RESEARCH PROBLEM	18
2.0 OBJECTIVES	19
2.1 MAIN OBJECTIVE	19
2.2 SPECIFIC OBJECTIVE.....	19
3.0 METHODOLOGY.....	20
3.1 REQUIREMENT GATHERING	20
3.1.1 PAST RESEARCH ANALYSIS	20
3.1.2 REFER OFFICIAL DOCUMENTATIONS	20
3.1.3 IDENTIFY EXISTING METHODOLOGIES	21
3.2 FEASIBILITY STUDY	21
3.2.1 TECHNICAL FEASIBILITY	21
3.2.1.1 KNOWLEDGE OF CI/CD	21
3.2.1.2 KNOWLEDGE OF MICROSERVICES	21
3.2.1.3 KNOWLEDGE OF JENKINS	21
3.2.1.4 KNOWLEDGE OF ARGOCD	21
3.2.1.5 KNOWLEDGE OF DOCKER.....	21
3.2.1.6 KNOWLEDGE OF CONTAINERIZATION	22
3.2.1.7 KNOWLEDGE OF KUBERNETES	22
3.2.1.8 KNOWLEDGE OF GO LANGUAGE	22
3.2.1.9 KNOWLEDGE OF AGILE METHODOLOGY AND PROJECT MANAGEMENT	22
3.2.2 ECONOMIC FEASIBILITY	23
3.2.3 SCHEDULE FEASIBILITY	23
3.3 SYSTEM ANALYSIS	24
3.3.1 SOFTWARE SOLUTION	24
3.4 SYSTEM DEVELOPMENT AND IMPLEMENTATION	24
3.4 TIMELINE	27
4.0 PERSONNEL AND FACILITIES	27
5.0 COMMERCIALIZATION	28

6.0	BUDGET.....	29
7.0	SUMMERY.....	30
	REFERENCES.....	31
	GLOSSARY.....	34

List Of Figures

<i>Figure 1: CI/CD/CDT Architecture</i>	<i>9</i>
<i>Figure 2: Monolithic vs Microservice-based Architecture.....</i>	<i>10</i>
<i>Figure 3: Wall of Confusion</i>	<i>12</i>
<i>Figure 4: Kubernetes Architecture</i>	<i>13</i>
<i>Figure 5: CI/CD Pipeline</i>	<i>14</i>
<i>Figure 6: Download resources from the internet and finally clean the workspace ..</i>	<i>16</i>
<i>Figure 7: System Overview Diagram.....</i>	<i>24</i>
<i>Figure 8: Automated Software Release process Architecture Diagram.....</i>	<i>25</i>
<i>Figure 9: Gantt Chart.....</i>	<i>27</i>

List Of Tables

<i>Table 1: Languages, tools, and Technologies.....</i>	<i>26</i>
<i>Table 2: Personnel and Resources.....</i>	<i>27</i>
<i>Table 3: The budget for the research.....</i>	<i>29</i>

List of Abbreviations

CI	Continuous Integration
CD	Continuous Delivery
CDT	Continuous Deployment
VM	Virtual Machines
MSA	Microservice System Architecture
TLS	Transport Layer Security
RBAC	Role Based Access Control
SRP	Software Release Process
PaaS	Platform as a Service
SaaS	Software as a Services
CPU	Central Processing Unit
IaaS	Infrastructure as a Service

1.0 INTRODUCTION

The software release process is critical in software development because it governs how software updates are planned, developed, tested, and distributed. A well-defined and optimized software release process can greatly improve software development efficiency, quality, and speed. Software release methods have changed dramatically in recent years, and Agile methodologies have emerged as a popular strategy to software development. Cloud computing, CI/CD and CDT automation [1] are becoming popular with the Agile Methodology, as most organizations began to use it and it became a fast way of creating software for both monolithic and microservice based architectures. With the rise of Agile and Microservice-based architecture, it became necessary to launch and maintain applications more quickly than previously. The primary goal of developing DevOps is to increase the automation of the software delivery process [2].

The software needed to be monitored and maintained after the software application was released during the software release process. There may be several phases of releasing a software program for creation, testing, user approval testing, and ultimately for use by end users called production or live. These stages may be different for each software company and the products depends on the need of the development process.

Error! Reference source not found.

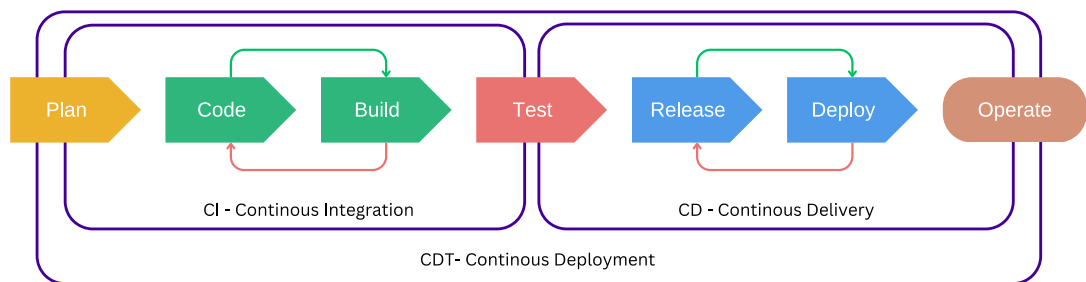
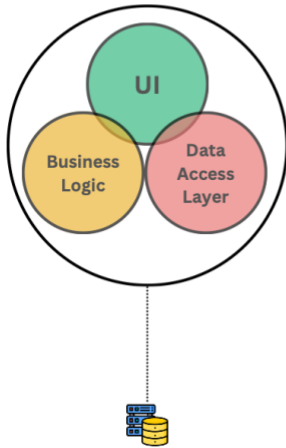


Figure 1: CI/CD/CDT Architecture

MSA [3] is based on set of micro-applications and communicate with each application as a single application to archive a business goal or goals. With the trend of using microservice based architecture, Docker is introduced and started to use that the micro-applications are deployed as docker container. Earlier, the monolithic architecture was popular in the software industry and used VMs as the deployment servers. VMs were huge barrier for the software release process with the growth of business needs and

fast software releases by using microservice based architecture. Many of the lead companies such as Netflix, Coca-Cola, Spotify, Amazon, SoundCloud, eBay, Karma and Uber moved from monolithic to microservice based architecture due to the reason of scaling the application with their developed features. Docker [4] became the most popular containerization platform that can be used to deploy microservice based applications easily and reduce the resource usage.

Monolithic Arcgitecture



Microservice-based Arcgitecture

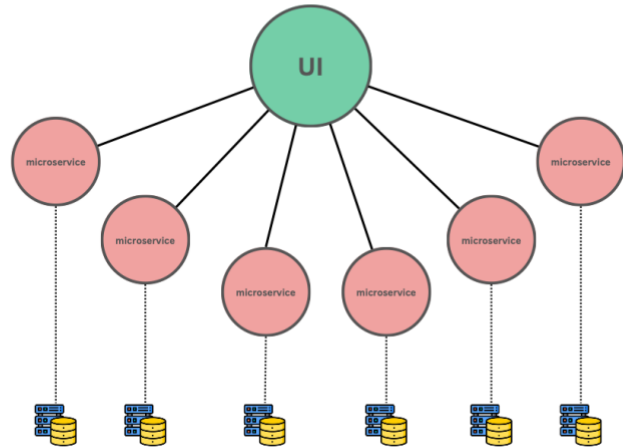


Figure 2: Monolithic vs Microservice-based Architecture

The application deployment was easy, scalable, and reliable with Docker, but with increasing number of containers, it was a hard and unmanageable to maintain the thousands of containers simultaneously. Kubernetes became the most popular and most usable platform to maintain containers as a container orchestration tool. Microservice based architecture was became a trend of software companies with the combination of Docker and Kubernetes platforms due to the easy and manageable of all the containers [5]. The micro-application communication mesh was the next challenge that needed to be resolved with giving a solution to manage and visible to manage easily. As a result, “Service Mesh Manipulation tools” were introduced such as Istio.

Kubernetes Service Mesh is a networking infrastructure component that manages and monitors interactions between Kubernetes cluster microservices. It includes several features that help with traffic control, security, and observability. Istio is one of the most common Kubernetes service networks.

Istio is an open-source service mesh with a robust collection of tools for managing microservices [6]. It is based on Kubernetes and employs the Envoy proxy as a data

plane to handle network communication between microservices. Istio includes functionality such as network routing, load sharing, circuit breaking, and fault injection.

One of Istio's main benefits is its ability to modify the service mesh setup. It enables users to set network traffic management policies for their microservices, such as how much data is routed to each service and which services are permitted to interact with one another. Istio also includes a robust collection of security features such as mutual TLS, RBAC, and permission rules to help safeguard communication between microservices. Istio also includes a robust collection of observability tools. It offers measurements, tracing, and recording for network activity between microservices. Istio makes it simple to identify and fix problems in a Kubernetes cluster.

The security, testing and performance are most important aspects of the software release process. During the software release process, they needed to be measure and fixed if they needed any modification. The SRP is partially automated and using by software companies, but they are not fully automated and optimized. There are human interactions in between software development to software release and those stages may be roadblocks and frictions during the entire SRP. Some of the stages may be error prone due to the human interaction. Not only the automation is needed in the SRP, but it is also necessary to optimize the SRP according to architectural attributes such as performance, security, extendibility, reliability, and modifiability [7].

The optimized software release process is designed as streamlined architecture and developed with open-source solutions for all the aspects of the software release process as a fully automated system. The main reason of using open-source solutions is to reduce the cost for paid solutions. And the other reason of using open-source solution is, able to customize with using the source code and developing with necessary components for the defined streamlined SRP architecture.

This research proposal aims to address the need and feasibility of software companies by providing a solution to optimize the SRP based on architectural attributes and automate the software release process. The proposed system will be tested as a real-world solution to finalize the architectural solution, which will be fully automated and optimized. The ultimate goal is to streamline the software development and release process, improve the quality of software releases, and help software companies in Sri Lanka to stay competitive in the global market with having fast and reliable software release process.

1.1. Background and Literature

Before the "Agile" and "Microservice" based architecture is being popular in the software industry there were two main teams during the software development life cycle as "Developer Team" and the "Operation Team". The development team is responsible to release new features and hotfixes, but the operational team is responsible for software stability and reliability. With the different interests in both teams, the release process had many gaps and barriers during the quick release of a software version. Software versioning has become normalized after the microservice-based architecture and agile methodology comes to the industry, because of the rapid development of the software product. After the release of the initial product, there were hotfixes, minor updates, and major update releases on the software. With the rapid growth and release process, the old barriers were also a roadblock to releasing the updated software quickly to the end users. As a result, Development and Operation teams needed a solution and the solution was having DevOps architecture and DevOps is responsible to implement and manage the software development and software release process [8]. After the CI/CD and Containerization became the main components of the software release process, the wall of confusion between "development" and "operation" teams was destroyed.

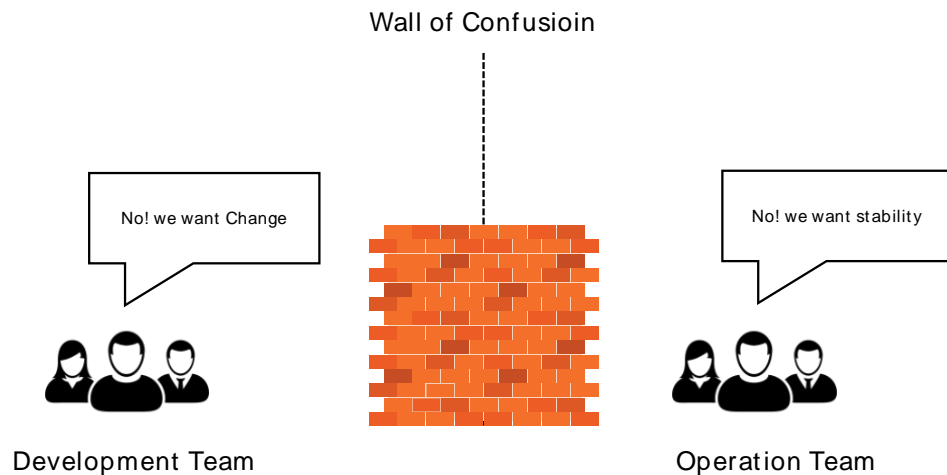


Figure 3: Wall of Confusion

The software release process is an essential part of software development, and optimization is critical for efficient, high-quality software releases. Agile methodologies, DevOps practices, containerization, container orchestration, security, testing, versioning, image manipulation, and service meshes are emerging trends in software development that are transforming the software release process. Istio is a popular open-source service mesh that provides a robust set of features for managing microservices in Kubernetes clusters. There are many numbers of solutions were

introduced for found problems and grow backs during the journey of DevOps into software release process. As the winner of the competition with different tools and technologies for containerization and container orchestrations, finally docker and Kubernetes survived their position with providing many numbers of features and solutions for raised problem during software release process and maintenance. Kubernetes architecture and Docker Architecture gives the clear and well-defined view for software release life cycle with providing the support to deploy and manage thousands of containerized applications [9]. During Docker manage the application images, Kubernetes manipulate the available containerized application.

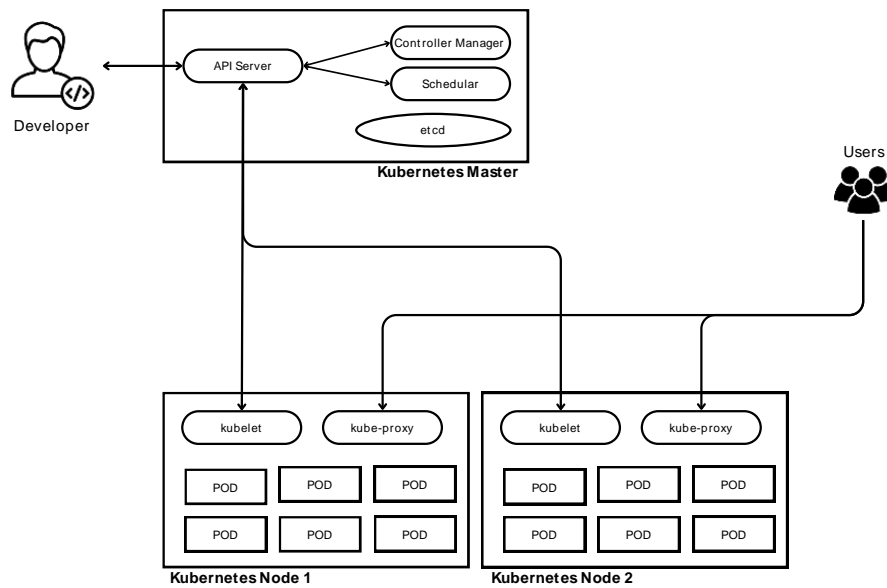


Figure 4: Kubernetes Architecture

Automated software release processes are invented to reduce the manual processes during software development life cycle. Currently, most of the software release process systems are automated but, they are not fully automated without the human interactions during approval processes and testing processes.

Because of the reason of not fully automated software release process systems, the need of fully automation become a common problem for the companies and businesses. As solutions CI/CD/CDT tools were invented. But there are only few of them are survived and became commonly used in the software industry. Not only those tools but also the tools and technologies for automated integration testing, automated security testing, unit testing and many other testing tools were invented and used in the industry.

Many reasons have influenced the need for automation and the implementation of DevOps techniques as software release processes have evolved gradually. One of the primary causes of this development has been the rising complexity of software systems, which has made manual release management increasingly challenging. As a

result, a larger focus has been placed on automation and the use of tools and methods that can streamline the software release process.

Continuous Integration/Continuous Deployment (CI/CD) was a significant step forward in this development [10]. CI/CD is a software development methodology that stresses the significance of regular and automated code testing, integration, and deployment. This strategy has reduced the likelihood of mistakes and disputes during the software release process while also allowing teams to deliver software more swiftly and effectively.

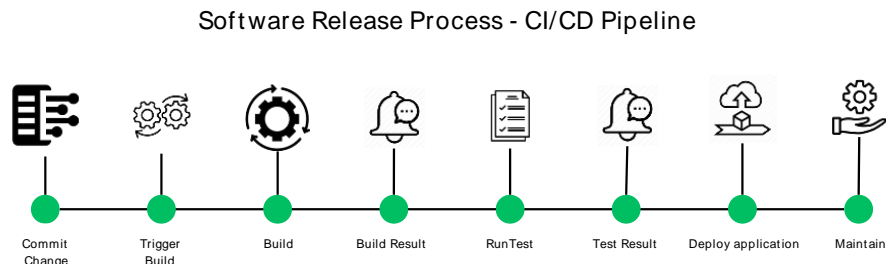


Figure 5: CI/CD Pipeline

The development of software release procedures has also been influenced by automation testing. Manual testing has become less efficient, time-consuming, and error prone as software systems have grown more complicated. Automation testing, on the other hand, allows teams to perform tests more swiftly, consistently, and frequently, lowering the risk of mistakes and conflicts during the software delivery process.

Another important element in the evolution of software release procedures has been the rise of DevOps techniques. DevOps is a concept that promotes cooperation, communication, and integration between development and operations teams. DevOps has helped to reduce the risk of errors and disputes during the software release process while also increasing the speed and efficiency of software delivery by breaking down silos between these teams and promoting a culture of cooperation.

Cloud computing has also changed the way software is released. Cloud platforms have allowed teams to scale their software systems rapidly and easily by giving on-demand access to a variety of computing resources, while also lowering the need for on-premises infrastructure [11]. Cloud platforms also offer a variety of tools and services that can assist teams in automating many parts of the software release process, from testing to distribution.

Even the software release processes are available with current automated tools, they have roadblocks due to some architectural attributes such as performance, security, extendibility, reliability, and modifiability [7]. And the current software release processes are not fully automated but only partially automated for different stages with

the business requirement of different software companies. The current release process has mentioned architectural attributes that needed to be optimized because the systems are not easy to modify and easy to integrate with new tools and technologies to improve the quality of software development. Some systems take too much time to build and run tests, and as a result the software application get too much time to deploy in the necessary environments due to performance issues. The security is a main problem that needed to be covered during the software release life cycle and needed to be integrated and improved with streamlined software release process. Overall, the current software release processes needed to be well optimized under the mentioned attributes as a fully automated system.

1.2. Research Gap

The main research gap in this research to fulfill the gap in current software release processes with optimizing according to the architectural attributes. This research is focused on invent an architecture to automate and optimize to avoid the gap between current solutions available in the software industry for software release process. The security of the available software release processes is too low level because both external and internal developers are working in a same company and must share the resources with external teams. It became a challenge and security issue of software release process because the external teams get accesses to databases and storages. The performance issues are getting higher with the scaling up the number of users of the software release process systems. The system may perform slowly or get some down times due to the reason of unable to maintain with the increasing number of users. With the new technologies and tools, the current release processes can be optimized and have more benefits with extending with novelties for performance, security, and maintaining. But the current systems are tight with the company environment and have not extendibility. Same as the extending, the modifiability is also a one main attribute that needed to have in a software release process system to improve the reliability, accessibility, and maintainability.

According to the publication [12], the security of Kubernetes and Docker Swam can be archived with cloud storages. But as discussed previously, the security is unable to archive when working with external teams. The system can have backups for a disaster recovery; but it cannot be assumed as a security aspect. PaaS and SaaS services are available to secure data inside cloud storages from external vulnerabilities. Inside the company, it needed to be easy to have the higher security for the stored data from external teams who needed to access same data during the software release process. Not only during using by the external teams, also need to improve the security of deployed application and docker images during building and pushing to a container registry. As the publication [13] is mentioned, there are challenges during applying Security with DevOps as DevSecOps. Also need to consider the given solutions but, data security and containerization security has lack of resources and available solutions as open-source solutions.

According to the publication [14], the performance on Kubernetes deployment is measured and studies under VMs. But the necessary solutions are only on Kubernetes

clusters with docker images. The performance may slow due to the lack of memory, lack of storage and lack of available CPU. The available solutions of publications are not meeting the necessary performance level to avoid the mentioned problems. The Docker image build time is also a waste of the time for the software release process with downloading necessary dependencies or packages for Maven or Node building technologies. Current available release process systems are downloading and pulling necessary data from internet all the time. Need to be optimized the performance with giving a solution for mentioned performance drawbacks.

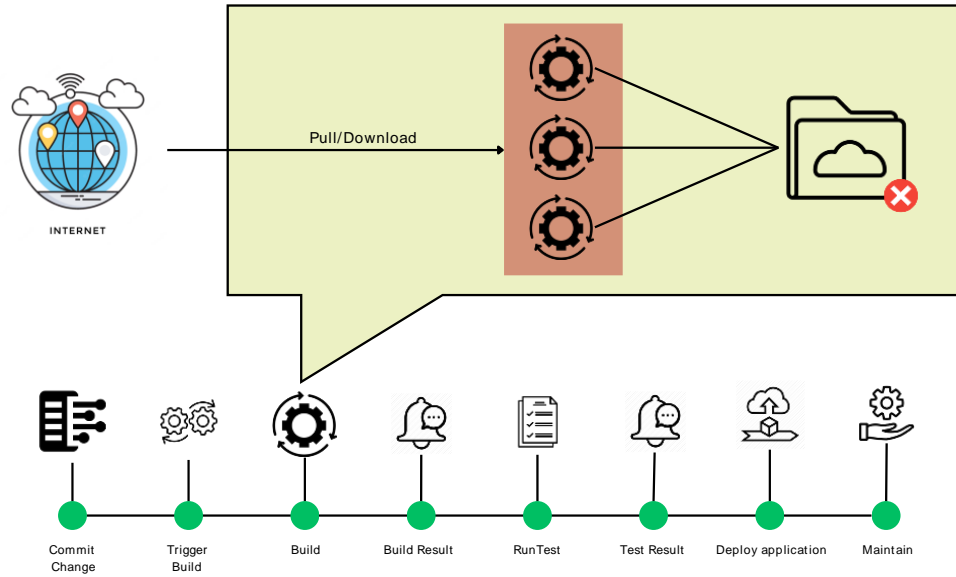


Figure 6: Download resources from the internet and finally clean the workspace

Furthermore, as the publication [15], the extendibility is applied with automated configuration, initialization, and deployment with algebraic approach. But the proposed system is not optimized with Kubernetes and Docker based automated software release process system. The solution needed to be more effective and improved to available the extendibility and modifiability. According to the publication [16], the unit testing is automated to avoid manual process of unit testing. But the solution is proposed with focusing Hydrologic Modeling. The current research is ongoing for microservice based architecture. Therefore, the solution needed to be improved to apply on microservice-based architecture. Moreover, the publication [17] I focused to automate the testing NodeJS and React projects. This needed to improve to available for any of the language or technology used in the software development. Once the testing is automated and could be applied to any programming technology, the extendibility can be optimized.

Software development reliability is needed to optimize from software building, testing, delivery, and deployment phases. As the publication [18], is focused to allocate resources during software testing phase. But the phases other than the testing is also needed to be optimized with the attribute of reliability. As the publication [19], the

proposed solution is expensive and directly it effects to the cost of the software release process. The cost reducing is also a gap between available paid versions. Therefore, the proposed algorithm on cloud [19] is not the best solution when continue with open-source solutions.

There are some research and case studies are based on different areas of the architectural attributes that covered in this research projects. But they are not covered the architectural attributes cover in this research project during the optimization phase of the software release process. As mentioned in the publication [20], the attributes such as performance, security and reliability are considered during optimization, but they have not focused on extendibility, modifiability, and automation of the software release process. Also, in the publication [21], the research is focused to optimize performance and reliability, but they have not well optimized security, extendibility, modifiability, and automation.

As mentioned in the publication [22], the proposed automation is based on industrial architecture, and investigated with different microservice based architectures. But the solution is based on Eclipse and depends on the specific Eclipse based framework. If this solution is applied, the overall optimization cannot be archived due to the performance, extendible and modifiability architectural attributes cannot be archived.

1.3. Research Problem

One of the most important problems in microservice-based designs is performance optimization, particularly in multi-cloud settings where service delay and network traffic can have a major effect on performance. Several factors contribute to performance optimization, including improving containerization and orchestration methods, making effective use of cloud resources, and finding performance obstacles through real-time tracking and analysis of system data.

When optimizing the software release process for microservice-based designs on multi-cloud platforms, security is another important factor to consider. Microservices are implemented in distributed settings, with each application posing a risk. To ensure security, possible security risks and weaknesses must be identified, safe communication routes between services must be established, and effective access control methods must be implemented. Security worries in microservice deployments are one of the most pressing study issues in this field. As more companies migrate to multi-cloud settings, guaranteeing the security of microservices becomes more difficult. This problem can be solved by conducting study on finding possible security risks and weaknesses in the software release process for microservice-based architectures on multi-cloud platforms. Solutions to minimize these risks and improve the security of microservices can then be created.

Another aspect that can affect the software delivery process for microservice-based designs on cloud platforms is extensibility. Extending microservices to satisfy shifting business needs frequently necessitates architectural changes, which can influence the release process. Adopting a modular architecture and applying design patterns such as the microkernel and plug-in patterns can aid in ensuring extendibility while reducing the effect on the release process. Microservice-based architectures must be reliable because they are frequently used to create mission-critical apps. Ensure reliability by finding possible sources of failure, adopting fault-tolerant mechanisms such as replication and redundancy, and having automatic recovery mechanisms.

Modifiability is another factor that must be considered when optimizing the software distribution process for microservice-based designs on Kubernetes platforms. Understanding the dependencies between services, finding the effect of changes on the system, and ensuring backward compatibility are all required when modifying microservices.

Collaboration and communication are essential for effective software delivery processes among coders, testers, and operations employees. Because of the distributed structure of the system, effective cooperation and communication can be especially difficult in multi-cloud microservices deployment.

With all the architectural characteristics we concentrated on here, the software release process must be completely automated, with no manual contact from software creation to software release.

2.0 OBJECTIVES

2.1 Main Objective

This research is aimed to invent an automated architecture and optimize the software release process system according to the architectural attributes such as performance, security, extendibility, reliability, modifiability.

2.2 Specific Objective

The sub-objectives of the research are as follows.

- Identify the existing pain spots and inefficiencies in the software delivery process for cloud-based microservice designs. This could include performing questionnaires or conversations with release partners, evaluating existing release paperwork and data, or watching the current release process in action.
- Understand the architectural characteristics that are pertinent to the software release process, such as speed, security, extendibility, reliability, and modifiability. Conducting a literature study of best practices and research in these areas, speaking with subject matter specialists, or evaluating case studies of effective microservice-based designs are all examples of this.
- Using suitable metrics and tools, assess the present software release process against the specified architectural characteristics. This will aid in identifying areas for development and may entail running trials or models to try various release strategies or setups.
- Create and execute an automated release process system that takes into consideration the architectural characteristics found. Selecting suitable tools and technologies, creating a pipeline or procedure that processes the various phases of the release process, and ensuring that the system is scalable, dependable, and secure are all examples of what this entails.
- Test and verify the automated release process system using suitable measurements and benchmarks and compare its efficiency to the prior manual release process. This will help to show the efficacy of the new system and may include user testing or other forms of proof.
- Over time, monitor and optimize the automated release process system, using input from stakeholders and continuing data to find areas for growth. This may entail updating the system to account for new technologies or shifting requirements, as well as continuous maintenance and assistance to ensure that the system continues to operate as intended.

3.0 METHODOLOGY

3.1 Requirement Gathering

Requirement gathering is an important part of the research process because it entails gathering and evaluating pertinent information to fulfill the research goals. Various methods are used to ensure a thorough grasp of the subject matter, including evaluating previous research performed in recent years and finding existing systems that offer alternative answers to the issues stated in previous parts. Furthermore, the research team collects information from several web sites, including official documentation from Jenkins, Kubernetes, OCI, Azure, Docker, and ArgoCD. These sites contain a wealth of information on the tools and technologies under consideration, such as their features, functionalities, constraints, and best practices for application.

A systematic strategy is used during the requirement gathering process to guarantee that all required information is gathered and analyzed. Identifying important stakeholders, interviewing subject matter specialists, running surveys, and reading pertinent literature are all part of this process. Using these methodologies, the research team can find gaps in current study and create research questions that can be answered through additional analysis.

3.1.1 Past Research Analysis

Many research papers have been published with the goal of automating the software release procedure. However, some study has been conducted to optimize software release procedures. Furthermore, no publishing has been made to provide the ability to create completely automated and optimize based on the architectural attributes concentrated on in this study.

The primary goal of past study analysis is to determine the tools and technologies required to construct the proposed system. Furthermore, it aids in identifying issues with previous publications as well as current difficulties.

3.1.2 Refer Official Documentations

Access to official documents is required to guarantee that we have current knowledge about the technologies that will be used in the development of the suggested system. While previous study papers can be useful tools, they may contain out-of-date material due to continuous technological updates. We have access to the most precise and up-to-date knowledge about the technologies we use because we use formal documentation from trustworthy sources. As a result, we will be able to create a more effective and efficient system that serves the requirements of our clients.

3.1.3 Identify Existing Methodologies

There are several current methods for automating the software release procedure, as described in publications [20, 21]. However, the suggested methods are not optimized, completely automated, or usable in real-world general software businesses. However, the systems are not completely optimized because they have many disadvantages due to fully automating and refining the system to provide the best performance, dependability, and availability.

3.2 Feasibility Study

3.2.1 Technical Feasibility

3.2.1.1 Knowledge of CI/CD

The knowledge of CI/CD is very helpful during architect the fully automated software release process. Because the main concept behind automation is grab with CI/CD pipelines. Understanding the usability and features are more important during inventing the fully automated software release process system.

3.2.1.2 Knowledge of Microservices

The system is focused to developed for microservice based architecture. Therefore, the core concept behind the inventing system is needed to be well-known before give the solution. The knowledge of microservices and architecture with related technologies will give the help to optimize and automate the software release process.

3.2.1.3 Knowledge of Jenkins

Once the CI/CD knowledge is achieved, the Jenkins is needed to know as the solution for CI. The Jenkins can be used to do the automaton building, delivery during the software release process.

3.2.1.4 Knowledge of ArgoCD

Knowledge of ArgoCD is the other main objective to have before moving forward with architecting the fully optimized and automated software release process. CD phase can be archived with ArgoCD in the CI/CD automations.

3.2.1.5 Knowledge of Docker

Before moving forward with microservice based architecture, it is necessary to have the knowledge of Docker. Docker images can be used to deploy the application as a

set of micro-services. Docker enables us to maintain the necessary versions of images easily with container registries.

3.2.1.6 Knowledge of Containerization

As mentioned in the Docker section; the containerization is the core concept behind the Docker architecture. Having the knowledge of difference between VMs and Containerization is mandatory before designing the architecture for the release process.

3.2.1.7 Knowledge of Kubernetes

To manipulate and maintain the Docker containers, it is mandatory to have a container orchestration tool. As a result, the Kubernetes became the best solution and the common platform for container orchestration. Therefore, the knowledge of Kubernetes is necessary to optimize and automate the software development life cycle.

3.2.1.8 Knowledge of Go Language

For the configurations and developments of the project components, it is mandatory to have the knowledge of Go programming language. Because most of the development tools are developed using Go programming language that we are going to apply in the research project.

3.2.1.9 Knowledge of Agile Methodology and Project Management

Having knowledge of Agile methodology and project management for completing the research project is high. The Agile methodology is well-suited for iterative development, which allows for continuous improvement and adaptation to changing requirements. Project management skills are also essential for organizing and coordinating the various aspects of the project, including team members, timelines, and resources. Moreover, expertise in these areas can help identify and mitigate potential technical challenges and risks associated with the project, ensuring a successful outcome.

3.2.2 Economic Feasibility

The suggested system cost should be as reasonable as feasible to be preferred over current automatic systems and tools. Most companies make their systems and solutions for Kubernetes and Docker available as open source. All automation and efficiency solutions must also be linked with open-source tools and technologies.

3.2.3 Schedule Feasibility

The suggested solution must be implemented within the time frame of the research. At the end of the study, the suggested automated and optimized software release process system can be combined with other members' solutions such as resource allocation, tracking, and multi-cloud deployment.

3.3 System Analysis

3.3.1 Software Solution

The following is an overview of the suggested software solution. The following are the major components of the approach.

- Continuous Integration Server
- Continuous Delivery Server
- Optimization of Automated CI/CD

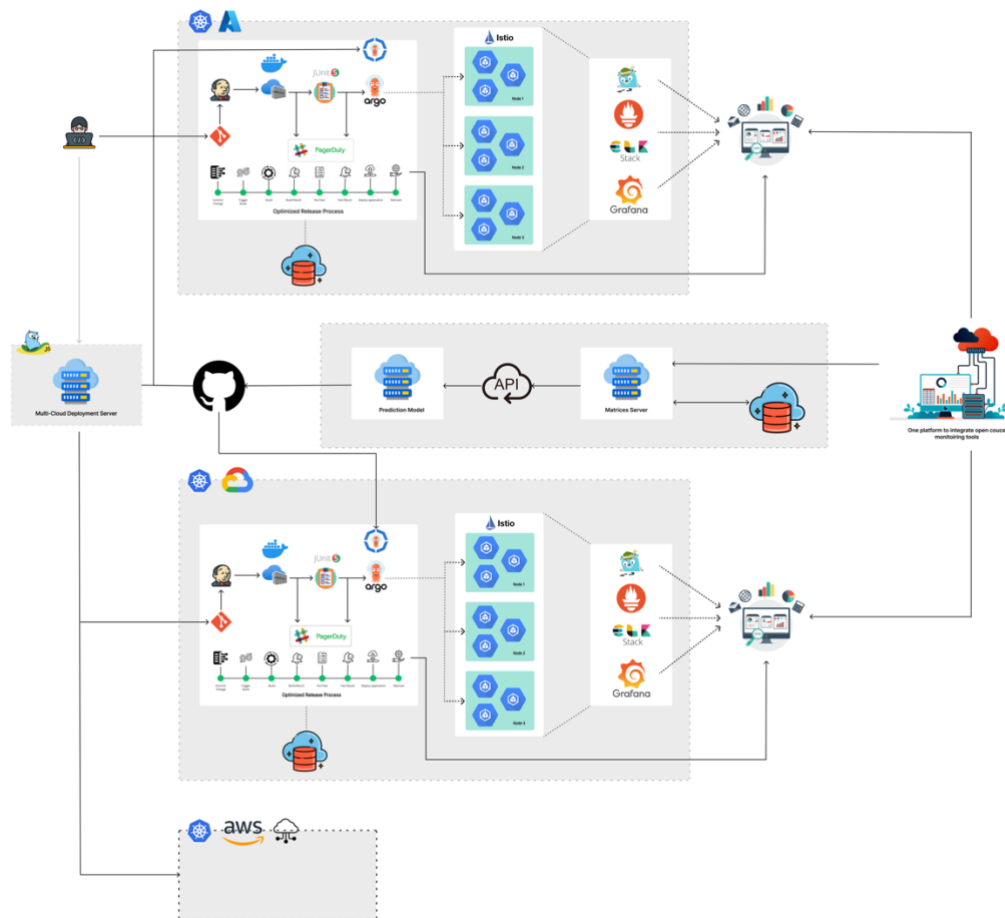


Figure 7: System Overview Diagram

3.4 System Development and Implementation

The high-level diagram of the automated and optimized release process architecture is as follows.

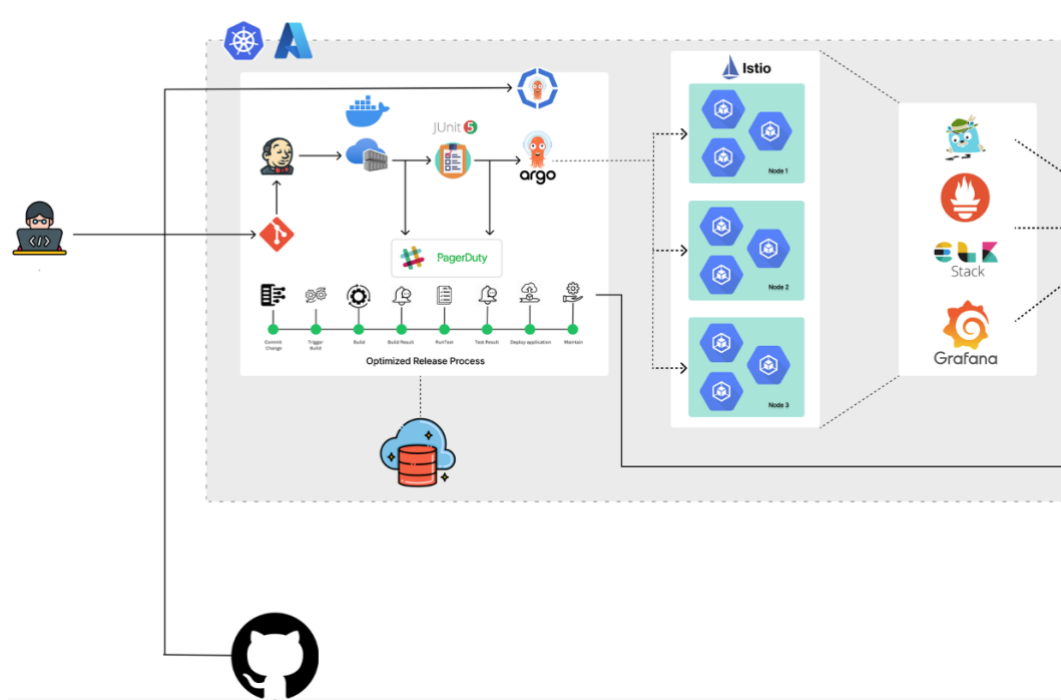


Figure 8: Automated Software Release process Architecture Diagram

Followings are the main steps that needs to be covered during the development and implementation of the fully automated and optimized software release process.

1. Choose open-source tools and technologies that can help optimize the release process for the defined attributes, such as Jenkins for continuous integration, SonarQube for code quality analysis, and Docker for containerization.
2. Re-architect the software release process with the open-source solutions for fully automated software release process.
3. Optimize the architected software release process according to the architectural attributes focused on this research.
4. Define the stages and steps of the release pipeline, such as code integration, testing, deployment, and monitoring.
5. Implement a continuous integration process to automatically build, test, and integrate code changes, using tools such as Jenkins, Git, and Maven.
6. Implement a continuous delivery process to automate the deployment of code changes to staging and production environments, using tools such as Ansible, Chef, or Puppet.
7. Implement automated security testing to detect vulnerabilities in the code, using tools such as OWASP ZAP or SonarQube.

8. Implement automated performance testing to measure the performance of the system under different loads, using tools such as JMeter or Gatling.
9. Implement monitoring and alerting tools to detect and respond to issues in real-time, using tools such as Nagios or Prometheus.
10. Define and measure success metrics for the release process, such as the frequency and quality of releases, reduction in time to market, and improvement in customer satisfaction.
11. Train and educate stakeholders on the new release process and tools, to ensure smooth adoption and usage.
12. Continuously iterate and improve the release process and tools, based on feedback, results, and emerging best practices.

The tools and technologies are as mentioned as follows to development and implement the proposed architecture.

Programming Languages	<ul style="list-style-type: none"> • Go • Java
Configuration Language	<ul style="list-style-type: none"> • YAML • Bash
Tools & Technologies	<ul style="list-style-type: none"> • Docker • Kubernetes • Git • GitHub • Azure • Jenkins • Argo CD • Kustomize

Table 1: Languages, tools, and Technologies

3.5 Timeline

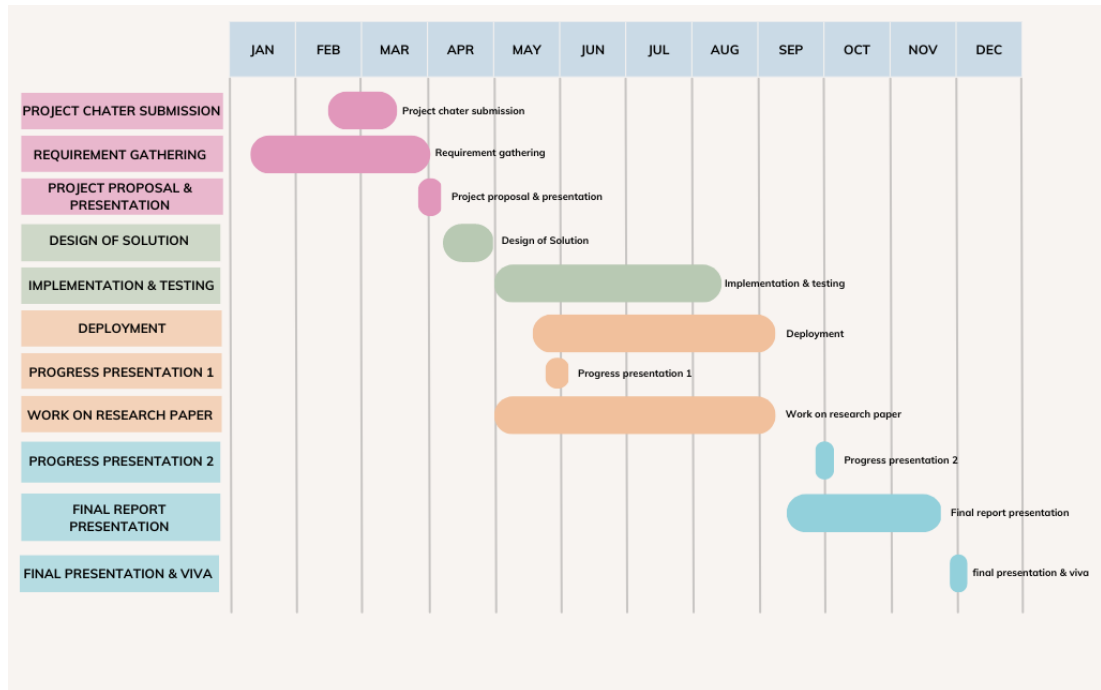


Figure 9: Gantt Chart

4.0 PERSONNEL AND FACILITIES

Name	Tasks
Herath H.M.I.P.	<ul style="list-style-type: none"> Information gathering related to the software release process systems currently available in the software companies. Architect initial architecture automated software release process system. Configure Kubernetes cluster and start to deploy necessary open-source tools as containers. Optimize the architected software release process system according to the proposed architectural attributes. Configure and integrate IaaS storage.

Table 2: Personnel and Resources

5.0 COMMERCIALIZATION

The proposed fully automated and optimized software release process, which utilizes open-source solutions, can be a cost-effective solution for software companies and the industry. By utilizing open-source solutions, organizations can reduce the cost of licensing proprietary software and access a wide range of tools and resources for development and deployment. The use of open-source solutions also promotes collaboration and knowledge-sharing within the industry.

The automated and optimized release process can help organizations save costs by reducing the manual effort required for releases, minimizing the risk of errors, and reducing the need for additional resources. The optimized architectural attributes of the software release process can help ensure that the releases are of high quality and meet the desired performance, security, extendibility, reliability, and modifiability requirements.

The benefits of this solution can be commercialized in the following ways:

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

By adopting an automated and optimized release process that utilizes open-source solutions, organizations can gain a competitive advantage in the industry by offering faster and higher quality software releases at a lower cost. This can lead to increased efficiency, improved quality, and overall cost savings for software companies and the industry.

6.0 BUDGET

The proposed system for the fully automated and optimized software release process is entirely software-based, and no hardware components are involved in the implementation process. Therefore, there will be no additional hardware costs to be incurred. The system will be implemented in a cloud environment, and Azure has been chosen as the cloud provider to implement the proposed system. As a result, the primary cost sources will be the subscription-based fees charged by Azure for the computing power of the Kubernetes cluster and virtual machines.

Expense Category	Cost per month
Azure Subscription	Free
Virtual Machines	3 500 LKR
Storage	Free
Load Balancer	7 000 LKR
Network Security Group	Free
Azure Kubernetes Service	45 000 LKR
Total	55 500 LKR

Table 3: The budget for the research

The budget for implementing, architecting, and using Azure for deployment and testing for the research project will depend on the selected services and usage requirements, as well as the development and deployment resources required. However, by using Azure services, the project can benefit from a scalable and flexible infrastructure, pay-as-you-go pricing, and access to a wide range of tools and resources for development and deployment, which can help reduce costs and improve efficiency. And the Azure Free Tier and the free \$100 will be used for the research project.

Note: It is important to mention that the proposed system will be developed using the Azure free tier, which provides \$100 worth of free credit for a period of 12 months. This credit will be utilized to create and manage the necessary virtual machines and Kubernetes clusters for testing and implementing the proposed system. Therefore, the total cost of the project will be minimized, and no additional expenses will be incurred for these purposes.

7.0 SUMMERY

The proposed research project aims to optimize and automate the software release process for microservice-based architectures on multi-cloud platforms. Microservice architectures are becoming increasingly popular due to their scalability, flexibility, and dependability. However, deploying and managing microservices on multi-cloud platforms can be complex and time-consuming, particularly when it comes to releasing new software updates.

The software release process consists of several phases, including development, testing, staging, and deployment. Each step requires a unique set of resources and tools, and each stage introduces the potential for human error. By optimizing and automating the software release process, organizations can reduce the risk of errors, enhance release quality, and speed up the deployment process.

To fully automate the release process, organizations need to use tools and technologies to automatically create, test, and deploy software updates. CI/CD pipelines, which automate the entire release process from code changes to deployment, can help with this. By eliminating the need for manual intervention, organizations can reduce the risk of errors and increase deployment speed. Inefficiencies in the software delivery process must be identified and eliminated to optimize it. This can be achieved through identifying bottlenecks, automating repetitive tasks, and improving resource utilization. By optimizing the release process, organizations can reduce costs, improve resource management, and increase productivity.

The research is focused to optimize the software release process according to five architectural attributes such as performance, security, extendibility, reliability, and modifiability. The solution is proposed using open-source solutions available for DevOps and cloud platforms.

Several factors must be considered to streamline the software release process for microservice-based architectures on cloud platforms, including containerization, orchestration, automation, monitoring, and security. By adopting best practices for each of these factors, organizations can improve the reliability and scalability of their microservices, reduce downtime, and ensure application security.

The proposed research project will help organizations optimize and automate their software release process for microservice-based architectures on multi-cloud platforms using tools and technologies such as Docker and Kubernetes. By doing so, organizations can improve their resource management, reduce costs, and increase efficiency while ensuring the reliability and security of their applications.

REFERENCES

- [1] “Exploring the Benefits of Combining DevOps and Agile,” Fernando Almeida, Jorge Simoes, Sergio Lopes, 19 February 2022. [Accessed 04 March 2023].
- [2] “Method for Continuous Integration and Deployment Using a Pipeline Generator for Agile Software Projects,” Ionut-Catalin Donca, Ovidiu Petru Stan, Marius Misaros, Dan Gota, Liviu Miclea, 20 June 2022. [Accessed 04 March 2023].
- [3] “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation,” Grzegorz Blinowski, Anna Ojdowska, Adam Przybyłek, 18 February 2022. [Accessed 04 March 2023].
- [4] “Microservice Architecture Practices and Experience: a Focused Look on Docker Configuration Files,” Luciano Baresi, Giovanni Quattrocchi, Damian Andrew Tamburri, 6 Dec 2022. [Accessed 04 March 2023].
- [5] “The key differences between Kubernetes and Docker and how they fit into containerization”, Josh Campbell, [Online]. Available: <https://www.atlassian.com/microservices/microservices-architecture/kubernetes-vs-docker>, [Accessed 04 March 2023].
- [6] “Analyzing and Monitoring Kubernetes Microservices based on Distributed Tracing and Service Mesh,” Yu-Te Wang, Shang-Pin Ma, Yue-Jun Lai, Yan-Cih Liang, December 2022. [Accessed 04 March 2023].
- [7] “The Challenges and Mitigation Strategies of Using DevOps during Software Development,” Dhaya Sindhu Battina, 1 January 2021. [Accessed 25 February 2023].
- [8] “Performance Assessment of Traditional Software Development Methodologies and DevOps Automation Culture,” P. Narang, P. Mittal, December 2022. [Accessed 10 March 2023].
- [9] “Kubernetes - evolution of virtualization,” Marek Moravcik, Martin Kontsek, Pavel Segec, David Cymbalak, 15 December 2022. [Accessed 10 March 2023].
- [10] “Effect of Using Continuous Integration (CI) and Continuous Delivery (CD) Deployment in DevOps to reduce the Gap between Developer and Operation,” Abrar Mohammad Mowad, Hamed Fawareh, Mohammad A. Hassan, 04 January 2023. [Accessed 10 March 2023].

- [11] "Containerized Microservices Orchestration and Provisioning in Cloud Computing: A Conceptual Framework and Future Perspectives," Abdul Saboor, Mohd Fadzil Hassan, Rehan Akbar, Syed Nasir Mehmood Shah, Farrukh Hassan, Saeed Ahmed Magsi, Muhammad Aadil Siddiqui, 07 June 2022. [Accessed 10 March 2023].
- [12] "CLOUD DATA SECURITY METHODS: KUBERNETES VS DOCKER SWARM," Avinash Ganne, December-2022. [Accessed 19 March 2023].
- [13] "Challenges and solutions when adopting DevSecOps," Roshan N. Rajapakse, Mansoor Zahedi, M. Ali Babar, Haifeng Shen, January 2022.
- [14] "Performance Study of Kubernetes Cluster Deployed on Openstack, VMs and BareMetal," Yeddula Sai Dhanush Reddy, Padumati Saikiran Reddy, Nithya Ganesan, B. Thangaraju, 30 August 2022. [Accessed 19 March 2023].
- [15] "Automation of Configuration, Initialization and Deployment of Applications Based on an Algebraic Approach," Pavel Shapkin, 26 November 2022. [Accessed 19 March 2023].
- [16] "Automated Unit Testing of Hydrologic Modeling Software with CI/CD and Jenkins," Levi T. Connelly, Melody L. Hammel, Benjamin T. Eger, Lan Lin, 2022. [Accessed 19 March 2023].
- [17] "AUTOMATED TESTING IN A CI/CD PIPELINE," Ville Santala, 2022. [Accessed 19 March 2023].
- [18] "Testing resource allocation for software with multiple versions," Adarsh Anand, Subhrata Das, Ompal Singh, Vijay Kumar, February 19, 2022. [Accessed 19 March 2023].
- [19] "Reliability modelling and optimization for microservice-based cloud application using multi-agent system", Zheng Liu, Huiqun Yu, Guisheng Fan, Liqiong Chen, 13 March 2022.
- [20] Z. Zhu, M. A. Naeem, and L. Li, "Automated Continuous Deployment at Facebook," in Proceedings of the 17th International Conference on Mining Software Repositories, IEEE Press, 2020, pp. 460-470.
- [21] H. Guo, J. Chen, T. Li, Z. Li, L. Zhang, and J. Li, "An optimization model for software release management based on continuous delivery," Information and Software Technology, vol. 107, pp. 101-115, 2019. H. Guo, J. Chen, T. Li, Z. Li, L. Zhang, and J. Li, "An optimization model for software release management based on continuous delivery," Information and Software Technology, vol. 107, pp. 101-115, 2019.

- [22] “AUTOMATING DEPLOYMENTS OF THE LATEST APPLICATION VERSION USING CI-CD WORKFLOW”, 2022, Spoorthi Jayaprakash Malgund, Dr Sowmyarani C N. [Accessed 19 March 2023].
- [23] “Commit, Release, Package: Automation in the development process for the ReFEx GNC System,” Sommer, Jan, 9 February 2022. [Accessed 19 March 2023].

GLOSSARY

- Microservices - A software architecture style that structures an application as a collection of small, autonomous services.
- Multi-cloud - The use of multiple cloud computing and storage services in a single network architecture.
- Software release process- A set of steps and procedures to build, test, and deploy software updates.
- Optimization - The process of making something as effective and efficient as possible.
- Automation - The use of tools and technologies to perform tasks automatically without human intervention.
- Containerization - A method of packaging and deploying software code and its dependencies into a container to run reliably across different computing environments.
- Orchestration - The automated management of containerized applications, including deployment, scaling, and connectivity.
- Tracking- The process of monitoring and recording the software release process to identify inefficiencies and improve it.
- Security - The protection of software and its data from unauthorized access, use, disclosure, disruption, modification, or destruction.
- Open-source - Software that is freely available and can be modified and distributed by anyone.
- Docker - An open-source containerization platform used to package and deploy applications and their dependencies.
- Kubernetes - An open-source container orchestration platform used to automate the deployment, scaling, and management of containerized applications.
- CI/CD pipelines - A set of tools and technologies used to automate the software release process from code changes to deployment.