National College of Ireland

# Enhanced Automation Solution for Multi-Cloud platform: Leveraging Advanced CI/CD Tools for Deployment, Security and Testing

MSc Research Project
MSc cloud computing

Azhar Akhtar
Student ID: X23195215@student.ncirl.ie

School of Computing
National College of Ireland

Supervisor: Shivani Jaswal

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Azhar AKhtar ……. …… |
| **Student ID:** | X23195215@Student.ncirl.ie …… |
| **Programme:** | Msc cloud Computing …… **Year:** 2024 …….. |
| **Module:** | Research Project …… |
| **Supervisor:** | 12/12/2024 …… |
| **Submission Due Date:** | …… |
| **Project Title:** | Enhanced Automation Solution for Multi-Cloud platform: Leveraging Advanced CI/CD Tools for Deployment, Security and Testing …… |
| **Word Count:** | 10028 …… **Page Count** 32 ……. |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

| | |
|---|---|
| **Signature:** | Azhar Akhtar …… |
| **Date:** | 12/12/2024 …… |

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Enhanced Automation Solution for Multi-Cloud platform: Leveraging Advanced CI/CD Tools for Deployment, Security and Testing

Azhar Akhtar
X23195215@student.ncirl.ie
MS Cloud Computing

## Abstract

This research establishes an innovative technique to solve critical challenges of DevOps Practices which includes vendor lock-in, deployment complexity, Integration of automated testing in multi-cloud and security issues in multi-cloud. Most of the previous research has explored the use of different tools like Terraform and Docker automating infrastructure management, deployment optimization and testing problems. However, many of these studies are restricted to single-cloud environments, fail to address the problems of vendor lock-in and in many cases overlook the critical phases of testing, security and deployment phases of CI/Cd pipelines in multi-cloud. The research often focuses on isolated tool comparisons such as Terraform Vs Pulumi AWS-specific solutions or other cloud provider solution without thinking about complex multi-cloud deployment, interoperability issues, testing issues and security challenges posed by using different cloud infrastructures. This research automated DevOps practices in multi-cloud by enhancing the integration of Terraform, Jenkins, GitHub and Docker using different techniques that solve these gaps and enable dynamic workload migration and cross-cloud orchestration. It leverages the terraform tools to use infrastructure as code to handle the infrastructure in multi-cloud and Docker to containerise the applications and Jenkins plays an important role in this which is used to automate the process of CI/CD. It automates dynamic deployments and allows continuous integration and delivery across multiple Cloud Providers. It also enhances the testing and security enforcement with the CI/CD pipeline. Automated testing and deployment, containerized applications and security policies is seamlessly integrated into the Jenkins pipeline with other tools which ensure compliance and operational standards across multi-cloud environments. This research provides a comprehensive solution which integrates different tools terraform, Docker, and Jenkins to address the issues in multi-cloud Environments providing enterprise solutions for the application with scalable, secure and cloud agnostic for CI/CD and infrastructure management. The system achieved 0% error rates across all test cases, with AWS handling a throughput of up to 118.41 hits/**second** and an average response time of 318.3 ms under heavy traffic, demonstrating its robustness in managing high-traffic and write-intensive workloads. Google Cloud, on the other hand, managed a higher throughput of 152.6 hits/second with an average response time of 246.68 ms, showcasing its efficiency and cost-effectiveness for dynamic scaling and rapid deployment. While AWS is optimal for enterprise-level applications requiring high reliability and performance under complex workloads, Google Cloud is better suited for agile projects and smaller workloads, emphasizing cost efficiency and quick

deployments. Both platforms displayed excellent scalability and operational reliability across varying traffic conditions

**Keywords – Docker, Terraform, Jenkin, GitHub, CI/CD, Multi-Cloud, DevOps, Deployment, Testing, Security, AWS Cloud, Google Cloud**

# 1 Introduction

The transformation toward a multi-cloud ecosystem in IT organizations has profoundly changed the software development lifecycle, offering opportunities and challenges. In the current fast-changing IT field, organizations are increasingly adopting multi-cloud strategies to leverage the distinct benefits offered by different cloud providers, such as improved scalability, availability, cost efficiency, and risk mitigation (Kim and Wang, 2023). However, this shift introduces new complexities in managing, orchestrating, and automating the infrastructure of different cloud providers and application deployments across these diverse environments. DevOps plays a crucial role in the software development lifecycle. It combines development and operations to enable efficient software delivery and automate processes. DevOps emphasizes Continuous Integration, Deployment, and testing, allowing organizations to achieve continuous delivery while maintaining reliability and stability (Tanzil et al., 2023; Tanzil et al., 2024)

One of the key practices in DevOps is Infrastructure as Code (IaC), which allows cloud services to be programmatically controlled and managed. It enhances scalability, reproducibility, and reliability. Tools such as Terraform and Pulumi have become widely used for implementing IaC at the enterprise level, enabling automated and consistent resource provisioning across cloud platforms (Karlsson, 2023; Ghosh et al., 2024). However, managing these tools in a multi-cloud ecosystem presents significant challenges. These challenges include maintaining consistent resource configurations, securing cross-cloud communication, and handling complex dependency management (Manca, 2023; Obi et al., 2024). Additionally, developers often face struggles in maintaining and integrating suitable DevOps tools that cater to the unique requirements of multi-cloud infrastructure for applications (Farayola et al., 2023).

The shift from on-premise servers to cloud-native architectures has also necessitated the use of containerization mechanisms such as Docker, which provides a standardized ecosystem for running applications independent of the platform. Containers simplify deployment and configuration across platforms by packaging applications and their dependencies, ensuring consistent performance and configuration (Farah and Patel, 2024). However, orchestrating containers and automating processes for deployment in a multi-cloud environment requires robust CI/CD tools like Jenkins. Jenkins automates the integration and deployment of various tools, reducing human error and enhancing overall efficiency (Chavan and Khadkikar, 2023; Sokolowski and Salvaneschi, 2023)

Despite these advancements, several research gaps persist at every step of the software development lifecycle. Many previous studies have focused on single-cloud deployments or

only compared individual tools used for IaC and pipelines that automate single-cloud processes. For example, studies comparing Terraform and AWS Cloud Development Kit (AWS CDK) often fail to address the challenges of the multi-cloud ecosystem (Pessa, 2023; Bafana and Abdulaziz, 2024). There is a critical need for comprehensive frameworks that integrate IaC tools and create pipelines capable of handling automated testing and deployment in multi-cloud setups to manage complex use cases effectively (Kalliomaai, 2024; Tanzil et al., 2024). Security is also a vital concern in both single-cloud and multi-cloud environments. Misconfigurations and vulnerabilities in containerized applications and automated deployments can have severe consequences for organizations and users. Therefore, implementing best practices for container security and compliance at every stage of application development is essential (Farah and Patel, 2024; Olaoye and Luz, 2024

This research addresses these challenges by developing a practical application with an end-to-end CI/CD framework using different multi-support tools for a multi-cloud ecosystem. The study utilized Node.js as the backend language to develop servers that handle user data and manage client requests. The application was containerized using Docker for consistent deployment and testing. For the frontend, the project created a user interface with React.js, deployed to Amazon S3 using Terraform. The backend application was deployed using Jenkins to orchestrate the CI/CD pipeline, automating code integration, testing with Mocha and Chai, and deployment to Amazon ECS with EC2 instances in AWS Cloud. The entire infrastructure was written using Terraform, ensuring efficient and reliable resource management (Ghosh et al., 2024; Pessa, 2023). The project was extended to other cloud providers, deploying the application on Google Cloud to demonstrate true multi-cloud flexibility and address vendor lock-in concerns. Additionally, this research explored the integration of advanced security libraries to protect Docker containers from threats and secure applications against attacks. The framework enhances security posture by ensuring compliance and operational standards (Farah and Patel, 2024; Obi et al., 2024).This research builds on previous studies by proposing a scalable, secure, and automated CI/CD framework for more efficient multi-cloud software delivery (Kim and Wang, 2023; Tanzil et al., 2023).

# 1.1 Research Question

To what scale can a comprehensive framework integrating advanced DevOps tools enhance automation in multi cloud ecosystem, particularly concerning in operational efficiency, resource management and system reliability?

Organizations are day by day change from the single cloud to multi cloud environments to get benefits from it for application scalability, flexibility, and cost efficiency. The different cloud providers offer these services for different types of applications (Obi et al., 2024) (Kim and Wang, 2023). still, this shift comes with great complexities. One big challenge is managing the infrastructure of different clouds and automating the process and communication between different modules of the application across multiple. which requires consistent configuration and secure, efficient deployment processes as well as checking the different threats at every step. (Farayola et al., 2023) (Obi et al., 2024) Existing research often focuses on single-cloud setups, it did not perform testing and security checks at different stages of application development which is independent from the cloud which solves a lot of 2 issues of multi-cloud. They failed to address these complexities in the multi-cloud ecosystem. such as security integration and vendor lock-in mitigation (Bafana and

Abdulaziz, 2024) (Pessa, 2023). Moreover, the management of applications on multi-cloud arises issues such as secure cross-cloud communication, preventing misconfigurations, and handling unauthorized changes, all of which can affect the reliability and performance of systems (Sokolowski and Salvaneschi, 2023; Olaoye and Luz, 2024). This thesis addresses these gaps by proposing an integrated framework that uses Infrastructure as Code, independent testing and security best practices to ensure seamless and secure automation across different cloud platforms (Farah and Patel, 2024) (Ghosh et al., 2024). By focusing on a practical, end-to-end solution, this research aims to advance the present capability of multi cloud DevOps practices make sure the importance of efficiency, security, and flexibility in complex, distributed environments and make generic environments for testing and security parts which run independent run from the cloud providers which reduces the communication problem between the different modules on different cloud provider.

# 2 Related Work

The change from premise server to cloud toward multi-cloud environments has transformed the application development life cycle in the organization. It is very important and necessary part to use modern tools for configuration, testing, and deployment another part of the development lifecycle stage. These tools are key for efficiently developing and managing software systems and automating the process. The IT landscape is evolving day by day. This research explores different studies of comparing tools and techniques for configuration management, Infrastructure as Code (IaC), security libraries and deployment automation, highlighting the pros and cons of these tools and identifying research gaps that guide future advancements

## 2.1 Significance of Configuration Management (CM) and Automation Tools

The research ”Configuration Management in the Modern Era: Best Practices, Innovations, and Challenges” focuses on the important role of configuration management in maintaining the integrity, reliability, and efficiency of IT organizations in the development of any system. As organizations evolve day by day to improve the process of their products as they transition from cloud to multi-cloud environments, CM becomes necessary to manage fast technological changes and large digital infrastructure for the systems. The study shows modern practices, including integrating CM with DevOps workflows for developing and managing the system, making use of containerization technologies, and implementing Infrastructure as Code (IaC) for automation to reduce the manual handling in the life cycle of software development. The research also addresses challenges such as scalability, security, and collaboration difficulties, it gives strategic solutions to improve CM practices. (Farayola et al., 2023)

## 2.2 Infrastructure as Code for resources

The IaC for resources is used to control and automate the cloud infrastructure resources with code. It automates cloud infrastructure management resources and there are different researches has explored its impact on the cloud infrastructure. In Daniel Karlsson’s research,

”Comparison of Infrastructure as Code (IaC) Frameworks from a Developer Perspective,” shows the comparison of AWS CDK services and Pulumi services. There are other tools but These are the two prominent IaC frameworks tools used for infrastructure as code. The research investigates these tools based on factors such as code readability, ease of use of these tools, and complexity of configuration of tools on the cloud provider. The result of the result shows that AWS CDK’s high-level abstractions that’s why it is easier to use and less cognitively demanding, On the other side, the Pulumi tool is very proficient in flexible, stack-specific configuration management but still both tools have constraints particularly in Command line interface functionality, Integration problems, security practices and support for complex use case scenarios for multi-cloud. The research gives the idea that future studies should inspect multi cloud helps also need advanced state management, security automation in every stage, CI/CD integration, and fault management to develop a more comprehensive IaC framework. (Karlsson, 2023).

Another research which is establish on the relative study of different infrastructures as code tools by Pessa, ”Comparative Study of Infrastructure as Code Tools for Amazon Web Services,” contrasts of AWS CDK and Terraform tools. This research mainly focuses on features, functionality, and performance in both tools In the result the researcher’s findings show that Terraform tools for infrastructure as code are better suited for multi-cloud ecosystems due to their efficient deployment and update operations. On the other hand, the AWS CDK is more user-friendly because it has a high level of abstraction for AWS-specific cloud providers. It is very friendly for seamless integration and familiar programming languages. The study recognizes gaps in CI/CD pipeline integration and consistent testing environments for multi-cloud as well, calling for more research on complex use cases and dependency management (9) There is another paper that addresses the growing challenges of complexity for IaC tools is ”Towards Reliable Infrastructure as Code” y Sokolowski and Salvaneschi

The paper ”Towards Reliable Infrastructure as Code” by Sokolowski and Salvaneschi addresses the growing complexity of IaC tools, likening them to traditional software methods rather than simple configuration scripts. The author’s attention to the need for robust testing and verification methods in each step of the creation of infrastructure as code services is because as failures in IaC scripts can cause significant security and deployment issues. The study proposes a solution that integrates modern fuzzing testing methods and property-based testing to enhance IaC reliability. The research shows that current IaC tools lack efficient testing mechanisms, and unit testing practices remain unmanageable. Future research should study and explore more streamlined testing techniques and the development of IaC tools optimized for performance and error prevention. (Sokolowski and Salvaneschi, 2023)

## 2.3 Automation and Multi-Cloud Tools

The study ”Comparative Analysis of CI/CD Tools in Multi-cloud Environments” by Nguyen and Lee’s explore different CI/CD tools and their productiveness in multi-cloud environments. The research compares Spinnaker, Jenkins, and other CI/CD solutions and

highlights the importance of tool compatibility with the cloud provider, ease of integration, and performance optimization in the lifecycle of development of applications. The research highlights the need for frameworks or use of libraries that facilitate seamless CI/CD pipeline integration, manage complex deployments with multi-cloud environments and ensure high system reliability. It also underscores the importance of handling cross-cloud communication and dependency management efficiently, an area still underexplored (Nguyen and Lee, 2024).

The study ”Choosing the Right IaC Tool for Building Reusable Cloud Infrastructure” further explores the strengths and limitations of different infrastructures such as code tools Terraform, Pulumi, and Ansible. It shows Terraform is ideal for multi-cloud environments due to its declarative approach and strong community support. On the other side, Pulumi with features like multi-language support is favoured for complex application configurations, while Ansible is best at automating repetitive tasks across multiple systems. The study emphasizes evaluating these tools based on their learning curve, scalability, and security features. Gaps identified include the need for improved security integration and better support for multi-cloud deployments (Kalliomaai, 2024).

## 2.4 Deployment and Automation Challenges

The study ”Automatic Deployment Solution for Multi-Cloud Environments” by Gon¸calves investigates challenges faced by developers using DevOps tools. Through conducting different surveys and analyzing post data of stack overflow the research categorizes common issues such as CI/CD tools, infrastructure as code for cloud, container orchestration, and quality assurance these are the categories which are study findings and give the ratio according to the severity of the issue. There are tools such as Jenkins, Ansible, Puppet, Terraform, and Kubernetes are highlighted as the most important tools with findings in the study emphasizing the complexity of version compatibility and cloud infrastructure automation. This research underscores the essential of real-world experience and prioritizes the development of supportive resources for efficient DevOps implementations in the software development life cycle (Gonçalves, 2023).

Further study by Ghosh, Srivastava, and Supraja ”Streamlining Multi-Cloud Infrastructure Orchestration,” is based on Terraform’s capabilities, and features to be enhanced with custom wrappers to simplify deployment across AWS and Azure. This research also discloses that a declarative approach is used by terrafrom to manage infrastructure resources to ensure alignment between the desired and actual states. However, the authors focus on future research as well in the areas of including open standards for cloud orchestration, unified management interfaces, and enhanced dynamic configuration management (Ghosh et al., 2024)

## 2.5 Importance of Containerization and Orchestration

Containerization technologies one of the most important and widely used technologies nowadays such as Docker, have become necessary in modern DevOps practices. According to research by Farah and Patel, analyses of containerized applications require strict security

measures, especially when deploying an application in a multi-cloud environment. This research focuses on using vulnerability scanning libraries like Trivy to ensure that images of docker are secure before deployment. In my research, the use of Docker and security scanning tools reflects the growing industry trend of integrating security libraries directly into the CI/CD pipeline using Jenkins as a pipeline for multi-cloud, often referred to as ”DevSecOps” Additionally, further in research, This research explore Docker is used for creating consistent development and production environments showing the importance of container orchestration tools, such as Docker for managing distributed workloads. Integrating practical experience into the literature review can provide a real-world perspective on existing research. Studies like ”Multi-cloud Infrastructure Management: Challenges and Best Practices” by Kim and Wang (2023) focus attention on the complexity of orchestrating CI/CD processes across diverse cloud providers. The main focus is that to implementing automation frameworks, such as Jenkins integrated with Infrastructure as Code (IaC) tools like Terraform, is crucial for maintaining deployment consistency, especially when dealing with Docker zed applications

## 2.6 Security and Compliance in Automated Frameworks

The growing factor in the multi-cloud context is reliance on automated frameworks for managing infrastructure which introduces significant security concerns. The literature, including studies by Nguyen and Lee (2024), shows that automated deployment processes must include security best practices to prevent vulnerabilities and unauthorized access to the application. The need for a robust security strategy is further researched and supported by Olaoye and Luz (Olaoye and Luz, 2024), who support comprehensive access control policies and regular security audits in multi-cloud ecosystems.

## 2.7 Research Niche

The increasing complexity of the multi-cloud ecosystem makes necessary a comprehensive framework to automate key phases of the CI/CD pipeline using advanced tools in the software development lifecycle. Organizations frequently experience challenges such as vendor lock-in, misconfiguration of automation scripts, and the management of resources across various cloud platforms. The centre of attention of most existing research points of convergence on single-cloud solutions with simple use cases has a view of critical aspects such as cross-cloud interoperability, robust testing mechanisms, and integrated security measures. This research addresses these gaps by proposing an advanced integration of different DevOps tools such as Terraform, Jenkins, Docker, and GitHub Advanced Security to streamline multi-cloud deployments. By integrating automated testing, security scanning, and dynamic infrastructure provisioning, the framework ensures scalability, reliability, and flexibility, making it a significant advancement in multi-cloud DevOps practices.

## 2.8 References, Challenges, and Key Insights Supporting Multi-Cloud DevOps Research

| ti | Problem | Key Insights and Relevance |
|---|---|---|
| Kim and Wang, 2023 | Problems in multi-cloud infrastructure management, including vendor lock-in and misconfigurations. | Bring into being the importance of addressing vendor lock-in and configuration consistency in multi-cloud environments. |
| Tanzil et al., 2023 | DevOps problems in testing and deployment. | Focuses on the need of integrating automated testing to improve pipeline reliability and scalability. |
| Tanzil et al., 2024 | Deficient focus on testing and security integration in DevOps pipelines on different stages. | Illustrate the fundamental of combining testing and security problems into CI/CD pipelines for robustness. |
| Karlsson, 2023 | Comparison of IaC tools (AWS CDK vs Pulumi). | Give a comparative basis for selecting Terraform in research for its multi-cloud capabilities and consistent configurations. |
| Ghosh et al., 2024 | Complexity in Terraform-based multi-cloud orchestration. | Validates Terraform's suitability for manage complex, scalable deployments across multiple cloud platform. |
| Farayola et al., 2023 | Configuration management problem and scalability in multi-cloud. | Focuses on the importance of scalability and configuration management to automate and simplify resource provisioning. |
| Gonçalves, 2023 | Issues in CI/CD tool compatibility and automation. | Strength the role of Jenkins and Docker integration for authorize seamless automation in multi-cloud CI/CD pipelines. |
| Pessa, 2023 | Comparative evaluation of Terraform and AWS CDK. | Set up Terraform as a preferred IaC tool for |

| | | |
|---|---|---|
| | | secure and consistent infrastructure as code in multi-cloud platforms. |
| Sokolowski and Salvaneschi, 2023 | Lack of robust testing in IaC tools. | Need to Shows the value of enhancing IaC pipelines with testing stage with mechanisms to ensure reliability. |
| Farah and Patel, 2024 | Security vulnerabilities in containerized applications. | Supports incorporating security tools like Trivy or other libraries to make sure secure and compliant deployments. |
| Nguyen and Lee, 2024 | Tool compatibility and cross-cloud communication in different clout platforms in CI/CD. | It show the importance of make sure tool compatibility to streamline CI/CD processes in multi-cloud automation setups. |
| Olaoye and Luz, 2024 | Security concerns in multi-cloud platform. | Need to focuses on the importance of access controls and regular security audits to secure multi-cloud deployments. |
| Nawagamuwa, 2023 | Testing gaps in application infrastructure using IaC. | Advocates for robust testing strategies to ensure consistency multi-cloud ecosystem. |
| Obi et al., 2024 | Security and efficiency problems in evolving cloud computing standard. | Validates the integration of advanced CI/CD procedures to manage evolving multi-cloud challenges. |
| Chavan et al., 2023 | Deployment complexities in container-based environments. | Focus on the role of automation in managing Kubernetes, Docker and multi-cloud deployments. |
| Bafana and Abdulaziz, 2024 | Difficulties in managing immutable infrastructure in AWS deployments. | Show the benefits of used of immutable infrastructure for consistent multi-cloud setups. |
| Kalliomaai, 2024 | Pick the right IaC tools for reusable cloud infrastructure. | Support for reusable Terraform modules to enhance multi-cloud |

| | | |
|---|---|---|
| | | infrastructure management. |

# 3 Research Methodology

The proposed DevOps solution is plan to implement an efficient, automated, and secure CI/CD pipeline across a multi-cloud environment on every stage of software development lifecycle, integrating code-level security controls, containerization, and deployment strategies. This section expresses the tools, components, and processes that enable this solution focusing on deployment speed, scalability, response times, security posture, and containerization efficiency. Key technologies that are used which include Jenkins, Docker, Node JS, Terraform, and AWS and Google Cloud services, each selected to streamline specific aspects of the pipeline

## 3.1 Jenkins (CI/CD Automation)

For automation the process of application this project used Jenkins as used primarily for continuous integration and continuous deployment. It is an open-source server that is used to manage the CI/CD pipeline in this project. It's have great plugins support allows flawless integration with multiple services that why it is most important tools for build, test, and deployment processes. Jenkins automates tasks from code commit to deployment which reducing manual intervention and improving deployment speed of the application as well help other stages as well.

- Code Checkout: Get the latest code version of the code from GitHub, make sure the pipeline always works with up-to-date code and run the pipeline.
- Build and Test Stages: Jenkins executes automated tests using Mocha and Chai using Docker file, validating functionality at each stage. This make sure that API is test on every stage.
- Security Scanning & dependence Scanning: Integrated with tools like CodeQL and Dependabot for security checks on GIT level which is benefit for different cloud platform. These tools identify and mitigate vulnerabilities in the codebase, enhancing the securities alerts and dependencies issues this process is independent from the cloud provider.
- Deployment Triggers: In the deployment stage application are Configured to automatically deploy to different cloud platforms (AWS and Google Cloud) upon successful test completion, optimizing deployment speed and minimizing human error.

## 3.2 Docker (Containerization)

Docker is utilizing to containerize both the backend Node.js application and the frontend React.js application. Containerization make sure that applications run consistently across different cloud platform which enabling efficient management and deployment across AWS Services and Google services.

- Container Build: In this stage Docker images are created for both backend and frontend services, encapsulating the applications and its dependencies into images. This promotes portability, ensuring consistent behavior across different platforms.
- Docker Testing: Containers are tested within Jenkins to validate functionality and any security vulnerabilities in a controlled environment before deployment, enhancing containerization efficiency by reducing issues related to platform-specific configurations.
- Image Storage and Management: Images are stored in two cloud providers such as AWS ECR and Google Artifact Registry make sure efficient retrieval for deployment on ECS (AWS) and Cloud Run (GCP). Docker's versioning capabilities allow easy rollback if needed, providing a robust fail-safe.
- A vulnerability scanner used for securing Docker images by identifying issues in containers, dependencies, and configurations which provides insight about this issues. This ensures secure and compliant deployment of containerized applications.

## 3.3 Terraform (Infrastructure as Code)

Terraform is integrated to manage infrastructure as code on both AWS and Google Cloud which provide automated, versioned, and consistent deployment of resources on both cloud providers. It provisions and manages all necessary cloud components, supporting scalability and resource utilization.

- Terraform Init: Initializes the Terraform environment and downloads necessary plugins based on the both cloud provider (AWS or GCP).
- Terraform Plan: Prepares a "dry-run" of the infrastructure setup it like plan for the resources, identifying any configuration issues before deployment.
- Terraform Apply: Run the script to create infrastructure configurations to the cloud, creating resources like ECS clusters, ECR repositories, Cloud Run services, and IAM roles. This ensures a consistent, repeatable infrastructure setup that can easily scale.
- Terraform Destroy: Give the ability to delete all resources when necessary, helping to control costs by removing unused infrastructure.

## 3.4 AWS and Google Cloud (Deployment Platforms)

The application is deployed on both AWS and Google Cloud to achieve a multi-cloud environment using Jenkins, Docker and terraform, Utilizing the different capabilities of each platform for efficient resource utilization, scalability, and monitoring.

### 3.4.1 AWS Services

- **ECS (Elastic Container Service):** Used ECS for Hosting the backend Node.js application, make sure that auto-scaling option are enable for handle increased traffic, ensuring scalability, and optimizing resource utilization.

- **ECR (Elastic Container Registry):** It is used to Stores Docker images, enabling quick and consistent deployment of containerized applications.

- **S3 and Cloud Front:** Hosting the frontend make sure fast content delivery, scalability for static resources, and improved performance for end-users.

- **RDS (Relational Database Service):** For database MySQL used for persistent data storage, providing a managed database solution to efficiently handle backend data requirements.

- **Cloud Watch:** Monitors Aws resource utilization, response times, and error rates. It managed essential metrics for analyzing scalability and system response under varying load conditions.

### 3.4.2 Google Cloud Services

- **Cloud Run:** It is used to Provides a environment for the backend Node.js application, allowing rapid deployment with automatic scaling to handle request.

- **Artifact Registry:** It is same to Stores Docker images, ensuring efficient and secure retrieval and deployment, similar to ECR on AWS.

- **Cloud SQL (MySQL):** Same database are used here for controlling persistent data storage for the backend application with Google's managed MySQL service, offering high availability and scalability.

- **Cloud Monitoring:** To tracks application performance, providing insights into CPU and memory usage, response times, and error rates across deployments.

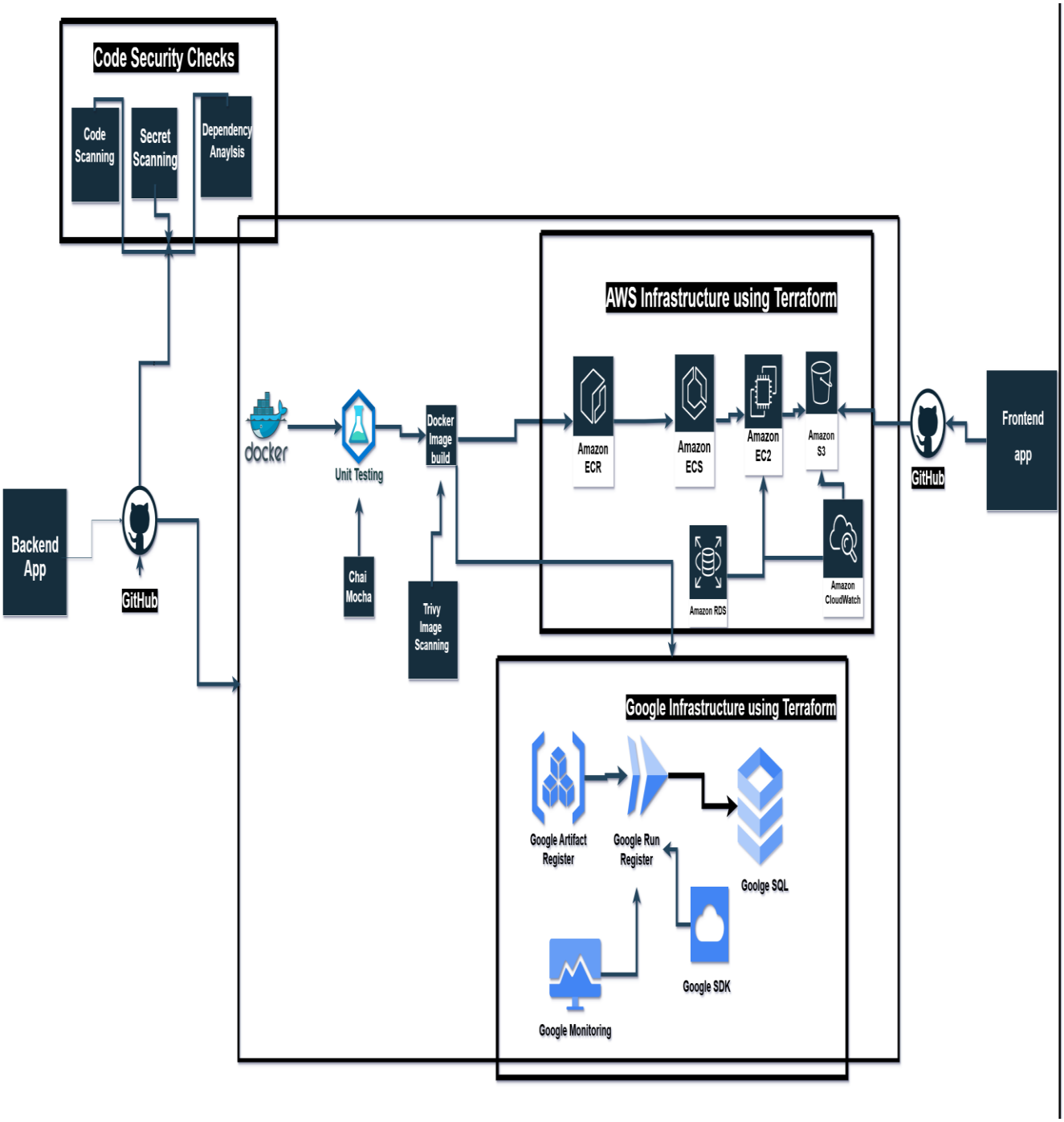
## 3.5 GitHub

- It is Used for version control with repositories secured using GitHub Advanced Security
- Scans for vulnerabilities, dependencies issues, and other security concerns to ensure high code quality

# 4 Design Specification

The proposed solution with tools is designed to implement an efficient and secure CI/CD pipeline across a multi cloud platform using both AWS and Google Cloud providers. The framework integrates key tools and techniques to make sure seamless automation, scalability, and performance monitoring. Below is a detailed explanation of the design:

## 4.1 Architecture Diagram:




Architecture Diagram of the Multi-Cloud CI/CD Framework figure (1)

The architecture diagram shows the comprehensive flow of the CI/CD pipeline and deployment processes in both environments such as AWS and Google cloud. The process start with code security checks it is independent from the cloud infrastructure that is used as generic it used GitHub Advanced Security tools such as CodeQL for vulnerability detection, secret scanning to identify hardcoded secrets in the code, and dependency analysis of the libraries that are used for the project implemented for security vulnerabilities. This make sure that only high-quality code, secure and standard code is proceeds to the next steps. The code is stored and managed in GitHub repository.it is the serves as the central repository for version control. When the code commit is made, Jenkins, the CI/CD orchestrator is run the pipeline to deployment the application that automates the build, test, and deployment stages. This includes to create the build of the application using Docker then running unit tests with Mocha and Chai to make sure backend application functionality. Same for the frontend another pipeline is created and run when the code is committed. After the testing and Before deployment of the application, Trivy scans Docker images for vulnerabilities at this stage, make sure application is secure and compliant builds. The backend Node.js application is deployed on different cloud platform such as AWS and Google Cloud to achieve a multi-cloud strategy. Terraform provision the infrastructure as code all the services on both providers which including Amazon ECS for hosting containerized applications with auto-scaling, Amazon ECR for storing application Docker images, and Amazon S3 with CloudFront for serving the React.js frontend with fast and scalable delivery. For storing data of backend is managed through Amazon RDS (MySQL), while Amazon CloudWatch monitors resource utilization, response times, and errors to provide insights into system performance. On the other hand, terraform is used for the google cloud resources. Google Cloud, the backend application is deployed using Cloud Run. Google Artifact Registry control Docker images, while Cloud SQL (MySQL) make sure a highly available and scalable database solution. Google Monitoring provides detailed metrics, including CPU and memory utilization and error rates of the application. The frontend React.js application is hosted on AWS S3, integrated with Cloud Front for fast delivery it just for the users. The combination of AWS and Google Cloud services with the tools like Jenkins, Docker, and Terraform make a robust, scalable, and secure infrastructure capable of handling different workloads and make sure seamless CI/CD operations. This solution not only make sure high performance and scalability but also integrates strong security measures at every stage.

# 5 Implementation

The implementation of the proposed solution of multi-cloud DevOps application focuses on automating the CI/CD pipeline, make sure that it takes robust security measures on every stage, and achieving fast and efficient deployment across both cloud providers. The solution process start with GitHub, which act as the repository for managing the backend app and frontend app code with features of Advanced security such as CodeQL, secret scanning, and dependency analysis of the code and with vulnerabilities and ensure the integrity of the codebase before progressing to other steps. Jenkins tool are used for the CI/CD pipeline, automating tasks such as code checkout, unit testing, containerization, and deployment. Mocha and Chai are used for unit testing for the backend to validate functionality and detect

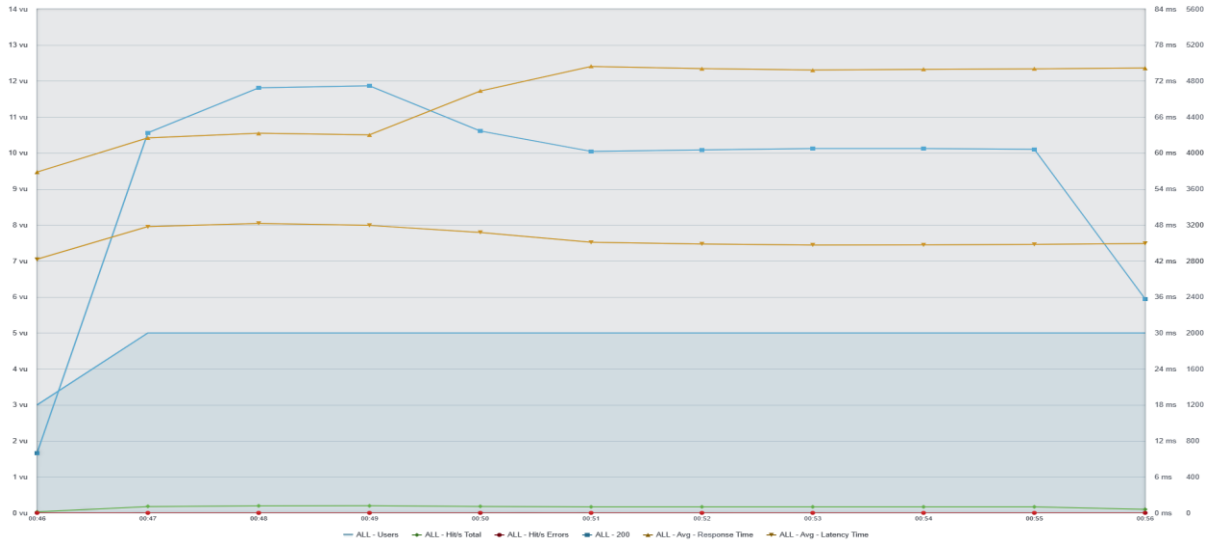
errors early in the code. Jenkins tool is integrating with Docker to build containerized versions of the backend application and frontend as well to make sure consistent runtime environments. Before deployment of the applications Trivy scans the Docker images for vulnerabilities, adding one more layer of security to the deployment process. The infrastructure is provisioned using Terraform that make enabling automated, consistent, and scalable resource management across both cloud providers. On AWS side the backend is deployed to Amazon ECS which is supported by Amazon ECR for container storage with EC2, Amazon RDS (MySQL**)** for database management, and Amazon Cloud Watch for performance monitoring and errors. The frontend application is deploying and hosted on Amazon S3, with CloudFront enabling fast and scalable content delivery of the content. On Google Cloud, the backend application is deployed using Cloud Run**,** with Google Artifact Registry controlling Docker images and Cloud SQL providing a scalable database management. Google Monitoring service is used to track application performance and monitor key metrics such as CPU and memory utilization. By combining these with secure and automated CI/CD pipelines the solution achieves faster deployment cycles, robust application security, and consistent performance monitoring, making it better for handling complex workloads in a multi-cloud ecosystem.

# 6 Evaluation

The evaluation of project focuses of different factors such as comparison between the aws and google cloud services with respect to CI/CD pipeline tools such as deployment speed, developer experience, integration with tools that emphasizing their efficiencies in different use cases. Performance metrics such as response time, throughput and error rates show the application scalability and reliability in addition with use cases and deployment speed robust security measures were integrated at different stages which show the vulnerabilities and simulating attacks at different level. This research provides comprehensive approach that validate the practical solution for optimization deployment, enhancing security, project objectives and make sure the operational efficiency in multi cloud environment.

## 6.1 AWS USE CASES

### 6.1.1 Performance Testing Results for Use Case 1




To evaluate the efficiency and scalability of the Node.js application that is deployed using Docker and AWS infrastructure with the help of terraform, performance testing was conducted using BlazeMeter tool.
The above metrics were for Use Case 1

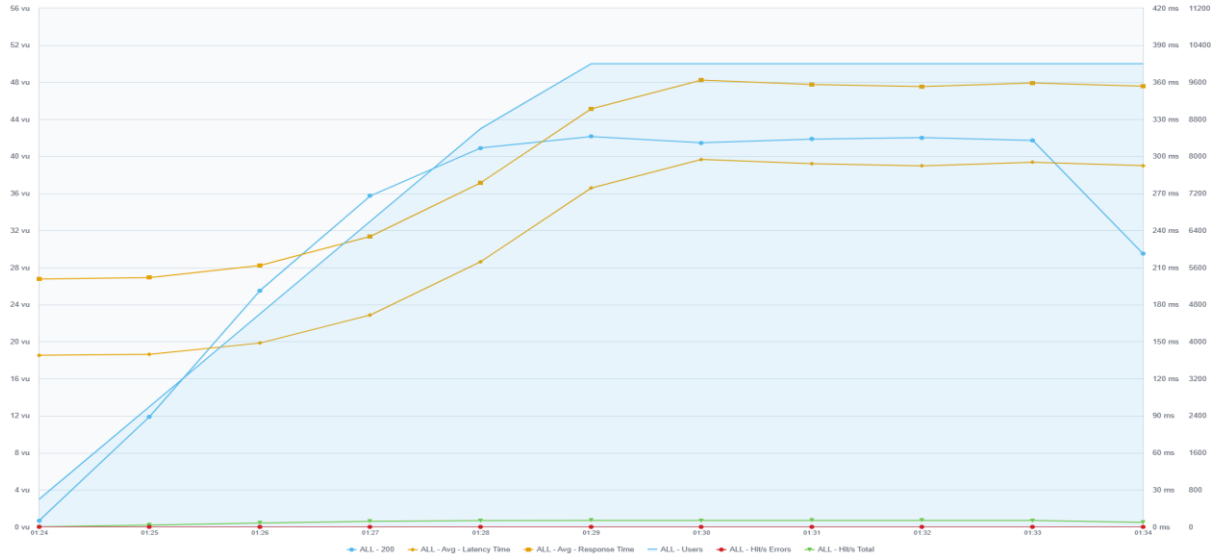
- **Maximum Users:** 5
- **Hits Per Second (Throughput):** 68.62 Hits/s
- **Error Rate:** 0% (indicating no errors during the test)
- **Average Response Time:** 69.71ms
- **90th Percentile Response Time:** 83ms
- **Average Bandwidth:** 1.99 MiB/s

**Graph Explanation:**

- **User Load (Blue Line):** The virtual user count gradually increased from 1 to 5, representing a light load scenario. The application handled this smoothly without delays to response to the use.
- **Hits Per Second (Yellow Line):** Throughput increased to 68.62 Hits/s, show that good efficiency in handling requests.
- **Average Response Time (Orange Line):** Response time stabilized at 69.71ms, indicating low latency and quick processing of requests of every user.
- **Error Rate (Green Line):** The error rate remained at 0%, confirming the application's reliability.
- **Bandwidth:** The bandwidth usage of 1.99 MiB/s indicates efficient data transfer, appropriate for the tested user load.

The application performed extraordinarily well under the load of 5 virtual users that maintaining a 0% error rate and also show low latency with an average response time of 69.71ms, and consistent throughput of 68.62 Hits/s. These results validate the reliability and robustness of the deployment pipeline and infrastructure configuration, ensuring efficient data transfer and quick processing for light traffic scenarios.

### 6.1.2 Performance Testing Results for Use Case 2




**Use Case 2** involved simulating a load of 50 virtual users (VUs) to test the scalability and reliability of the system under heavier traffic conditions.

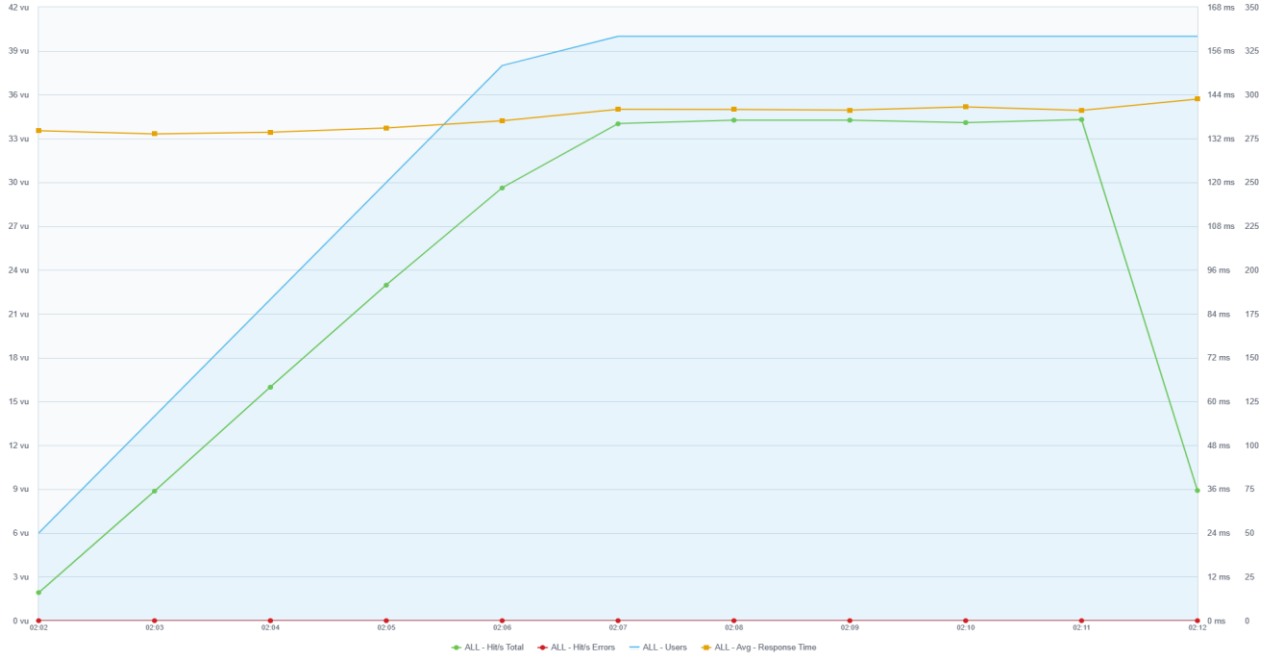
- **Maximum Users:** 50
- **Hits Per Second (Throughput):** 118.41 Hits/s
- **Error Rate:** 0% (indicating all requests were successful)
- **Average Response Time:** 318.3ms
- **90th Percentile Response Time:** 388ms
- **Average Bandwidth:** 3.44 MiB/s

**Graph Explanation:**

- **User Load (Blue Line):** The number of virtual users steadily increased to 50, simulating a moderate load. The system successfully scaled to meet the demand.
- **Hits Per Second (Yellow Line):** Throughput peaked at 118.41 Hits/s, reflecting the backend's ability to handle a higher request rate for the users.
- **Average Response Time (Orange Line):** The response time stabilized at 318.3ms, showing that the system maintained performance under heavier traffic with good response time .
- **Error Rate (Green Line):** With a 0% error rate, the application handled all requests without issues.
- **Bandwidth:** Bandwidth usage of 3.44 MiB/s reflects efficient data handling, even with a higher number of users.

These results show the robustness of the deployment pipeline and infrastructure configuration as code with AWS ECS with EC2 instances and S3 for frontend hosting. This test case shows the scalability of the application and the ability of AWS infrastructure to handle high traffic effectively, with no degradation in service quality.

### 6.1.3 Performance Testing Results for Use Case 3




**Use Case 3** involved simulating a load of 40 virtual users (VUs) to evaluate the system's ability to handle write operations effectively under moderate traffic conditions.

- **Maximum Users:** 40
- **Hits Per Second (Throughput):** 217.48 Hits/s
- **Error Rate:** 0% (all requests were successful)
- **Average Response Time:** 138.65ms
- **90th Percentile Response Time:** 149ms
- **Average Bandwidth:** 257.44 KiB/s

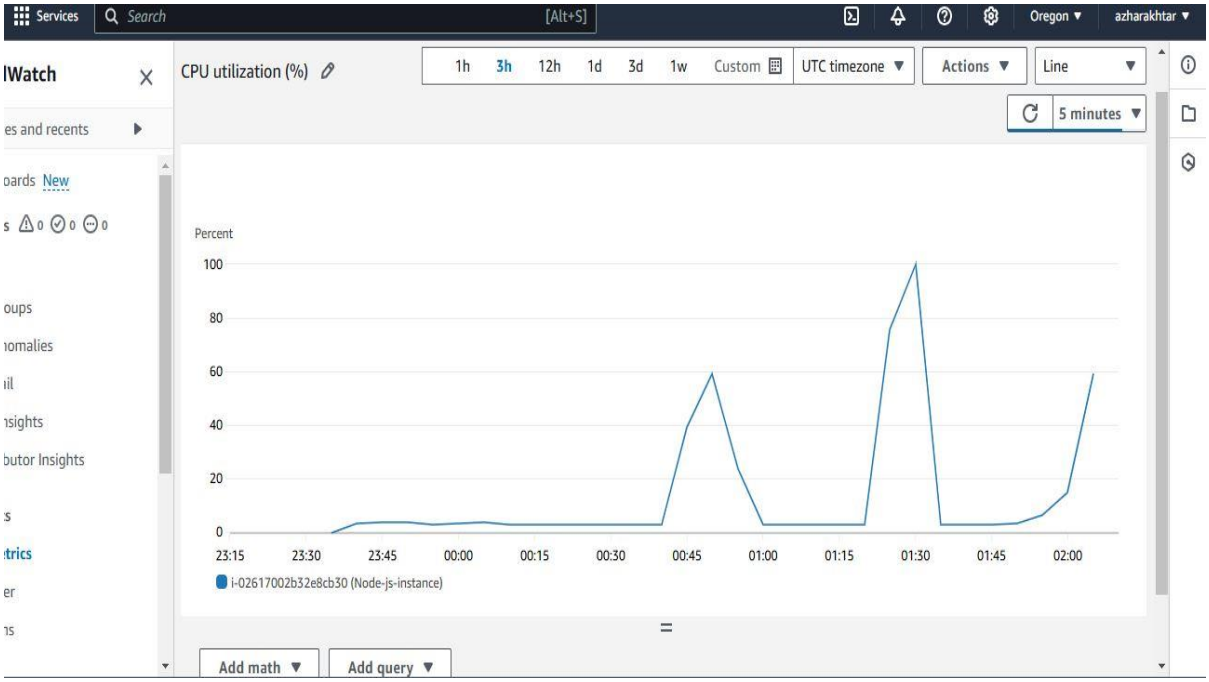
These results validate the backbend application feature to handle write-intensive workloads while ensuring data integrity and efficient processing. This test case shows the efficiency and scalability of the application's backend for data insertion and updates of data as well which show that the AWS infrastructure is optimized to handle write-heavy scenarios without bottlenecks in service quality.

### 6.1.4 Overall Benefits Across Use Cases

1. **Reliability:** All use cases, the error rate remained at 0%, showcasing a highly dependable backend infrastructure.
2. **Scalability:** The system effectively handled and control increasing user loads that make sure seamless operation under diverse traffic conditions.
3. **Performance Optimization:** The average response times and high throughput highlight the system's capability to process both read and write operations without performance decrease.
4. **Efficiency:** Efficient bandwidth usage ensures smooth data transfer and optimized resource consumption, even under high loads.
5. **Real-World Readiness:** The robust performance validates the system's ability to handle different workloads.

These results confirm that the deployment strategy, infrastructure setup (AWS ECS/ECR, S3, Docker), and automated CI/CD pipeline (Jenkins + Terraform) are well-optimized to deliver high performance and reliability for diverse workloads for different applications.

### 6.1.5 CPU Utilization Analysis for Use Cases




The graph provided shows the CPU utilization (%) over time for the Node.js instance during the performance tests of all three use cases on the aws services. Below is the analysis:

## Observation:

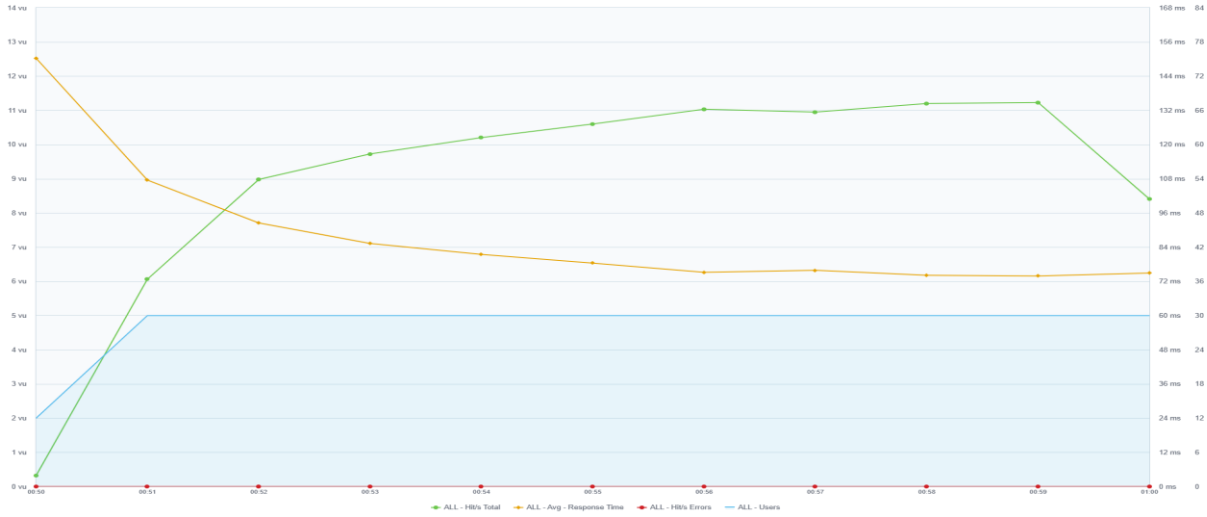
1. **Baseline Usage:**
   - At the start (23:15 to around 00:45) CPU utilization remained low near to 0%. This reflects minimal activity during periods without load testing or when the system was nothing to do.
2. **First Spike (Around 00:45):**
   - A sharp increase in CPU usage is observed, reaching nearly 60%.
   - This spike likely corresponds to Use Case 1, where a small load of 5 virtual users was applied. The backend responded to the read requests efficiently, causing a moderate spike in CPU activity before settling back to idle.
3. **Second Spike (Around 01:15):**
   - A significant spike is seen when the testing is performing, peaking at approximately 70-75% utilization.
   - This spike matches the execution of Use Case 2, where a larger load of 50 virtual users generated both read and write operations at that time. The mixed workload involved more intensive processing, including handling simultaneous database queries and updates, leading to higher CPU usage.
4. **Third Spike (Around 02:00):**
   - The CPU usage rises again, reaching around 40%.
   - This correlates with **Use Case 3**, focusing only write operations (POST requests). Although write operations are computationally heavier than the read, the smaller load of 40 virtual users resulted in lower CPU utilization compared to Use Case 2.

### 6.1.6 Explanation of Results of CPU Utilization:

1. **Efficient Resource Utilization:**
    - The CPU usage increased proportionally with the workload shows that the backend scaled well with the rising demands of the request and user load.
    - The system was not overwhelmed at any point with low to high load with CPU utilization staying below 80%, indicating sufficient computational capacity for all three use cases.
2. **Consistency Across Use Cases:**
    - Spikes in CPU utilization align with the intensity of each use case that show that the infrastructure was well-configured to handle different workloads efficiently.
3. **Idle Periods:**
    - The graph's return to near-zero utilization after each spike highlights the system's ability to return to an idle state when no active workload is present.

## 6.2 GOOGLE USE CASES

### 6.2.1 Performance Testing Results for Google Cloud Use Case 1



To measure the efficiency and scalability of the Node.js application deployed using **Google Cloud Run**. The following metrics were captured for Use Case 1

- **Maximum Users:** 5
- **Hits Per Second (Throughput):** 59.23 Hits/s
- **Error Rate:** 0% (indicating no errors during the test)
- **Average Response Time:** 80.77ms
- **90th Percentile Response Time:** 98ms
- **Average Bandwidth:** 1.71 MiB/s

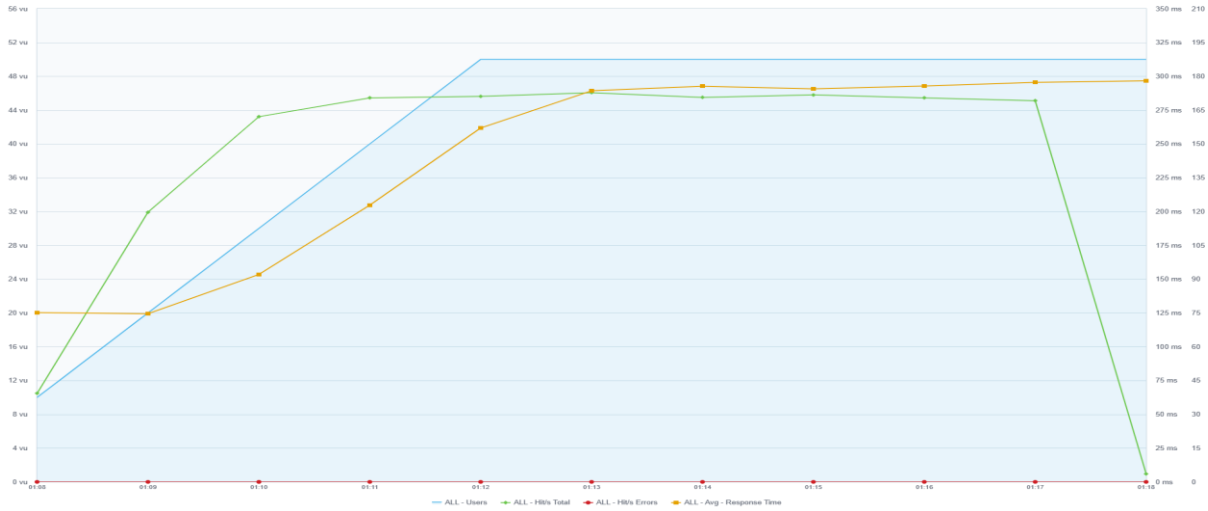
### 6.2.2 Graph Explanation:

- **User Load (Blue Line):** The virtual user count increased to 5, representing a light load scenario.
- **Hits Per Second (Green Line):** Throughput increased to 59.23 Hits/s with increasing the virtual users that show backend application capability to efficiently handle the incoming requests.

- **Average Response Time (Orange Line):** Response time stabilized at 80.77ms, show quick processing and low latency for the requests.
- **Error Rate (Red Line):** The error rate remained at 0%, indicating the system's reliability in handling all requests successfully.
- **Bandwidth:** The bandwidth usage of 1.71 MiB/s reflects efficient data transfer suitable for the tested load.

These results shows to validate the robustness of the deployment pipeline and infrastructure configuration using Google Cloud Run and google other services. This test case focusses the scalability of the application and the ability of Google Cloud infrastructure to handle light traffic effectively without any degradation in service quality.

### 6.2.1 Performance Testing Results for Google Cloud Use Case 2



To measure the scalability and reliability of the Node.js application deployed using google cloud services. The following results were captured for Use Case 2:

- **Maximum Users:** 50
- **Hits Per Second (Throughput):** 152.6 Hits/s
- **Error Rate:** 0% (indicating no errors during the test)
- **Average Response Time:** 246.68ms
- **90th Percentile Response Time:** 363ms
- **Average Bandwidth:** 4.41 MiB/s

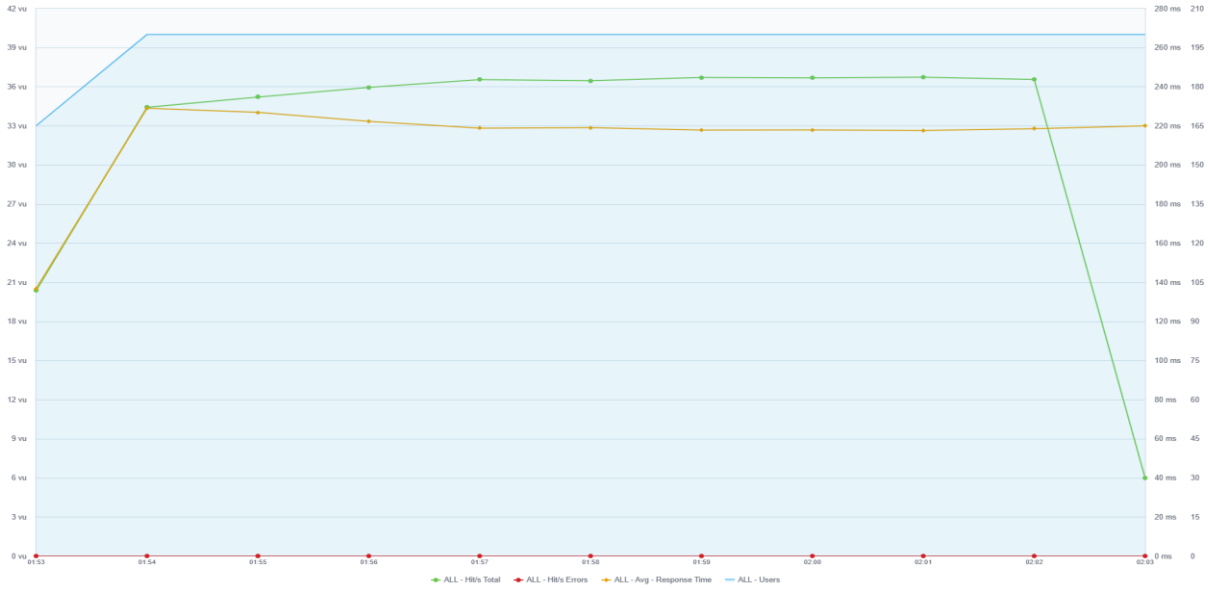
### 6.2.2 Graph Explanation:

- **User Load (Blue Line):** The users count increased to the maximum of 50 users. The system handled this increase effectively, with no decrease in service quality.
- **Hits Per Second (Green Line):** The throughput at raised.6 Hits/s, show the backbend's ability to process a high volume of mixed read and write requests efficiently.
- **Average Response Time (Orange Line):** The average response time reach at 246.68ms, while the 90th percentile response time was 363ms, indicating consistent performance under more traffic conditions.

- **Error Rate (Red Line):** The error rate remained at 0% throughout the test, show that the application successfully processed all requests without failure.
- **Bandwidth:** The average bandwidth of 4.41 MiB/s reflects efficient data transfer capabilities to handle high traffic loads.

These results show the reliability and scalability of the Google Cloud Run infrastructure for controlling heavier workloads on both read and write operations. This test case shows the robustness of the deployment pipeline and the features of Google Cloud services to maintain high performance under significant traffic, ensuring a responsive and reliable user experience.

### 6.2.3 Performance Testing Results for Google Cloud Use Case 3




To show the capability of the Node.js application to handle write-intensive operations using Google Cloud Run and other cloud services. The performance testing was conducted with 40 virtual users (VU). The following results were captured for Use Case 3:

- **Maximum Users:** 40
- **Hits Per Second (Throughput):** 176.08 Hits/s
- **Error Rate:** 0% (indicating no errors during the test)
- **Average Response Time:** 215.89ms
- **90th Percentile Response Time:** 285ms
- **Average Bandwidth:** 214.35 KiB/s

### 6.2.4 Graph Explanation:

- **User Load (Blue Line):** The user of the application count increased to a maximum of 40, simulating a moderate traffic scenario of request for application for write operations. The system scaled effectively without delays.
- **Hits Per Second (Green Line):** Throughput spark at 176.08 Hits/s, show the backbend's efficiency in processing high volumes of POST requests.

- **Average Response Time (Yellow Line):** The average response time reach at 215.89ms, with 90% of the requests completing within 285ms, indicating consistent performance under this load.
- **Error Rate (Red Line):** The error rate remained at 0%, confirming the system successfully processed all POST requests without failures.
- **Bandwidth:** The average bandwidth of 214.35 KiB/s show the efficiency of data transfer while handling concurrent write operations.

The backend controls a 0% error rate with consistent response times that show the robustness and scalability of Google cloud services for handling write-heavy requests. This test case shows the infrastructure's ability to control moderate traffic levels with reliable and efficient write operation processing, making it well-suited for applications that involve frequent database inserts or updates.

### 6.1.4 Overall Benefits Across Use Cases (Google Cloud)

The performance testing of the Node.js application deployed on Google cloud services show the following benefits across all use cases:

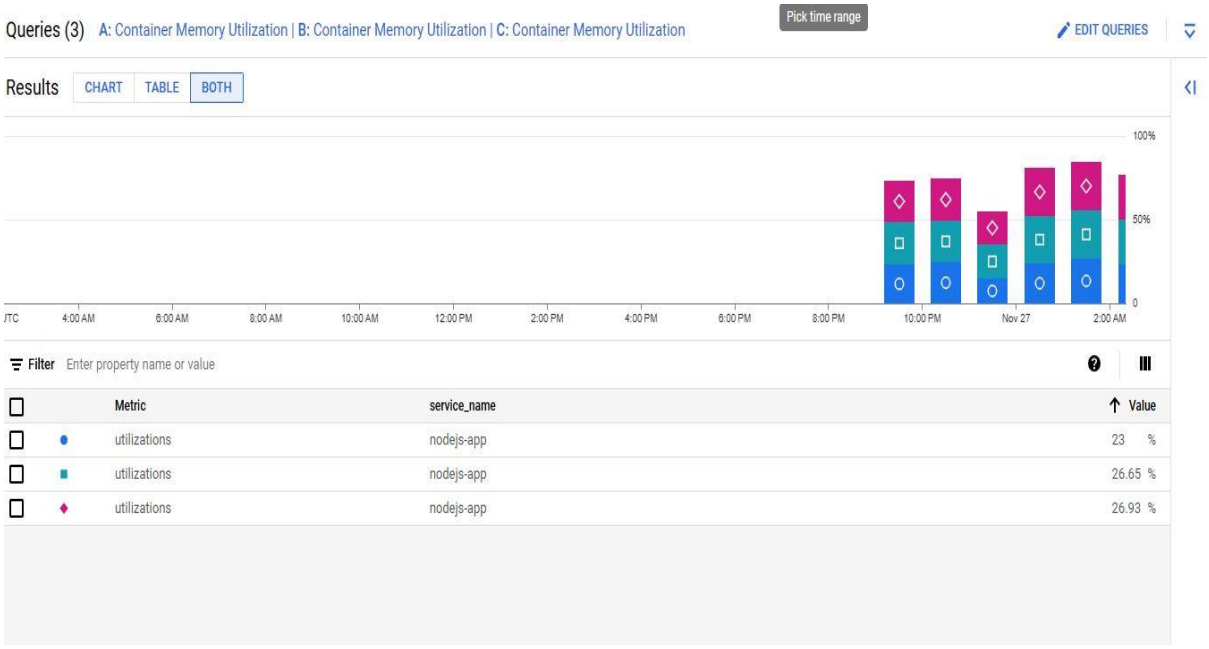
- **Reliability:**
    - Across all use cases, the system maintained a 0% error rate, make sure the robust and dependable handling of all requests, whether read or write operations.
- **Scalability:**
    - The application efficiently scaled to accommodate increasing user loads and requests of the user from 5 users in Use Case 1 to 50 users in Use Case 2, and 40 users in Use Case 3, without performance degradation.
- **Consistent Performance:**
    - Response times remained within good ranges under varying loads, with average response times of 80.77ms (Use Case 1), 246.68ms (Use Case 2), and 215.89ms (Use Case 3), show the backbend’s optimization for handling diverse workloads.
- **High Throughput:**
    - The backend application achieved high throughput rates, with peak Hits Per Second values of 59.23 (Use Case 1), 152.6 (Use Case 2), and 176.08 (Use Case 3**)**. show the system's capability to process a large number of requests efficiently.
- **Efficient Bandwidth Utilization:**
    - The infrastructure controls efficient bandwidth usage, with average bandwidths of 1.71 MiB/s (Use Case 1), 4.41 MiB/s (Use Case 2), and 214.35 KiB/s (Use Case 3), make sure the smooth data transfer and optimized resource utilization.

- **Adaptability to Different Workloads:**

    - The system performed effectively across a different of scenarios which including light read-heavy workloads (Use Case 1), mixed read-write operations (Use Case 2), and write-intensive tasks (Use Case 3), validating its adaptability to real-world application demands.

The testing outcomes show the reliability, scalability, and efficiency of Google cloud services for deploying and managing Node.js applications. The infrastructure proved capable of handling diverse traffic patterns and workloads from min to heavy request while maintaining high performance and low error rates, making it an excellent choice for applications requiring flexibility and resilience.

### 6.2.1 CPU Utilization Analysis for Google Cloud Use Cases




The provided graph shows the CPU utilization (%) for the Node.js application deployed on Google Cloud Run across the three use cases. Below is a explanation:

Observation:

1. **Baseline Utilization:**
    - The initial CPU utilization remained low when the system was handling zero traffic. This indicates that Google Cloud Run efficiently manages idle resources to minimize overhead.
2. **Use Case 1:**
    - The first spike in CPU usage corresponds to use case 1 where 5 users hit on system which create a light traffic load with read-heavy operations (GET requests)**.**
    - CPU utilization show 23%**,** demonstrating that the application handled the requests with minimal computational effort and maintained efficient resource usage.
3. **Use Case 2:**

- The second spark increase in CPU usage with respect of Use Case 2**,** which involved 50 virtual users performing a mix of read and write operations**.**
- The CPU utilization increased to 26.65%**,** show the additional computational effort required for handling both types of operations on heavy request. The moderate rise in CPU utilization indicates the system's ability to scale effectively under heavier traffic conditions.

4. **Use Case 3:**
   - The final spike to Use Case 3, which involved 40 virtual users performing write-intensive operations**.**
   - CPU utilization reached its peak at 26.93%**,** slightly higher than Use Case 2, due to the heavier workload associated with database write operations. This show that the system effectively managed the increased complexity of handling write-heavy tasks.

### 6.2.2 Explanation:

- **Efficient Resource Utilization:** The CPU usage for all three use cases remained well within acceptable limits, with utilization peaking at 26.93%, indicating that the application was optimized for efficient computational resource usage.
- **Scalability:** The slow increase in CPU utilization across use cases show the system's ability to scale proportionally to the workload.
- **Idle Efficiency:** Back to lower CPU utilization levels after the spikes confirms that Google Cloud Run efficiently control and manages resources during periods of reduced activity, minimizing unnecessary costs.
- **Workload Distribution:** The CPU utilization sparks align with the complexity of each use case, with read-heavy workloads requiring less CPU power and write-heavy workloads demanding more computational resources

## 6.3 Deployment Speed Analysis, Easy of deployment and other factors Across AWS and Google Cloud

The deployment process for the application which is deploy on Aws and google cloud services show the efficient automation and timing across whole process of CI/CD stages. Most likely two stages such as trivy vulnerability scanning and pushing Docker images were the most time intensive steps on both platforms. While AWS and Google cloud had slight differences in total time duration both have showcased and secure deployment workflow.

- **AWS Deployments**

| Stage | Duration (1st deployment) | Duration (2nd deployment) |
|---|---|---|
| Checkout Code | 4.8 sec | 4.8 sec |
| Test Docker Access | 1.9 sec | 1.6 sec |
| Build Docker Image | 17 sec | 17 sec |
| Trivy Vulnerability Scan | 5 min 3 sec | 3 min 10 sec |
| Login to AWS ECR | 11 sec | 11 sec |
| Push Docker Image to ECR | 3 min 53 sec | 2 min 05 sec |
| Deploy to ECS (EC2) | 5.1 sec | 5.1 sec |

| | | |
|---|---|---|
| Total Duration | 9 min 58 sec | 5 min 55 sec |

- **Google Deployments**

| Stage | Duration (1st deployment) | Duration (2nd deployment) |
|---|---|---|
| Checkout Code | 3.8 sec | 3.9 sec |
| Test Docker Access | 1.6 sec | 1.5 sec |
| Build Docker Image | 15 sec | 17 sec |
| Trivy Vulnerability Scan | 3 min 38 sec | 3 min 39 sec |
| Login to Google Artifact Registry | 3 min 28 sec | 17 sec |
| Push Docker Image to Artifact Registry | 38 sec | 24 sec |
| Deploy to Google Cloud Run | 30 sec | 30 sec |
| Total Duration | 8min 47 sec | 5 min 22 sec |

The deployment speed show for both Aws and Google Cloud show that different efficiencies in CI/CD workflows. In the first deployment took approximately 9 minutes 58 second around and optimization with change into code second deployment reduced the time to 5 minutes 22 second around on both cloud providers. It decrements in the deployment is depend upon different factors such remove security vulnerabilities and extra dependences from code and leveraging with simple strategies for the application testing, security which is independent from the cloud provider while used multi cloud environments and its also effect the tools used for CI/CD pipeline such as Jenkins, Terraform, Docker etc.

- **Cost Observations**

Aws have higher costs for continuous and high demand workload due to hourly billing for EC2 instances and RDS but it offers flexible pricing models and saving plans for long term optimization. On the other hand, google cloud which offer lower costs for server less services such as Cloud Run which bills is on usage. It is more costs effective then aws with power of handling unpredictable workloads.

- **Ease of Deployment**

Aws provides wide range of tools and integration which offer a mature ecosystem but configuration of services can be complex and time consuming. Other the other side google which simpler the deployment process it abstracts much of underlying complexity of infrastructure complexity

- **Integration with CI/CD Tools**

Both cloud provider services are integrating well with Jenkins, Terraform and Docker that enabling unified CI/CD pipeline however aws provider other tools like code pipeline and code build and google also provide cloud build tools.

## 6.4 Discussion

The findings of this research focuses on the importance of multi-cloud CI/CD automation frameworks solution using different tools and their effectiveness in facing the difficulties. such as deployment consistency, scalability, and security across different cloud platforms. Due to time constraints and a limited learning curve of just three months to explore all tools and services this research focuses on implementing a practical and streamlined multi cloud CI/CD framework rather than out and out exploration of all possible solutions. It shows the generic idea how the security libraries or tools are integrated and compatibility of the tools with cloud providers. Using the idea other tools and libraries are integrated in such a way which enhanced the security, testing and other stages of application development. Now Both AWS and Google Cloud showed their potential in handling and controlling automated CI/CD pipelines with tools such as Jenkins, Docker, and Terraform. However, the comparative analysis discloses refinement differences that can show the selection of the optimal platform based on specific use cases. AWS be in view the superior performance in scenarios which involving high traffic and complex workloads which especially in write-intensive operations. The services such as Elastic Container Service (ECS) and Relational Database Service (RDS) were influential make sure scalability and reliability. The used of infrastructure as code Terraform integration facilitated consistent infrastructure management. While other Aws services like ECR come up with seamless Docker image storage and retrieval. These results support AWS's strength in managing enterprise-level applications of different sectors requiring robust infrastructure, high throughput, and minimal latency.

Notwithstanding Google cloud by very good in deployment speed and cost efficiency that specially for smaller workloads applications or with dynamic scaling needs. Cloud Run service with its server less architecture make it easy the deployment process and optimized resource used by scaling based on demand. Artifact Registry and Cloud SQL services worked united to support backend operations of the application which shows Google Cloud's flexibility in maintaining performance under varying workloads of the application. These discovering Google Cloud's suitability for agile applications and environments where quick deployment cycles are important. Security played a very important role throughout the CI/CD pipeline on both platforms on every stage of software development life cycle. The integration of tools like Trivy or other libraries that are easily integrated which is independent the cloud services for vulnerability scanning and GitHub Advanced Security ensured robust defenses against threats in the code. While both AWS and Google Cloud providers provided a secure ecosystem. Aws services is mature ecosystem provide a slightly more comprehensive suite of monitoring and compliance tools, such as Cloud Watch. On the other hand, Google Cloud's efficient resource management reduced overhead and cost, presenting a competitive advantage for smaller-scale projects.

In conclusion, while both cloud provide are highly have the ability so for application the selection between AWS and Google Cloud depends on the specific requirements of the application. Aws is strongest match for large-scale, complex workloads demanding high performance and reliability, whereas Google Cloud gives an edge in deployment speed, cost-

efficiency, and flexibility for dynamic scaling. This research emphasizes the significance of integrating advanced DevOps tools in a multi-cloud ecosystem, enabling organizations to achieve scalable, secure, and efficient CI/CD pipelines customize to their unique operational needs.

# 7 Conclusion

This research established a comprehensive framework using advance tools for automating CI/CD pipelines in multi-cloud platform to addressing important challenges such as vendor lock-in, deployment complexity, security, and testing in multi cloud ecosystems. By manipulating the strengths of both providers with tools like Jenkins, Docker, and Terraform etc. The study showed how a consolidated CI/CD approach can enhance operational efficiency across different workloads of the application. AWS surpassed in managing high-traffic, write-intensive workloads through robust services like ECS, RDS, and Cloud Watch on the other hand Google Cloud lighted its features in rapid deployment and cost-efficient resource management using Cloud Run and Artifact Registry services. The integration of security measures at every stage which including Trivy for container vulnerability scanning and GitHub Advanced Security which make sure compliance and minimized risks affirming the importance of embedding security into CI/CD pipelines. The solution validated the use of Terraform for consistent infrastructure management, Docker for portable application ecosystem and Jenkins for orchestrating automated workflows for any cloud. It showed how multi-cloud strategies not only reduce vendor lock-in but also optimize resource utilization which make sure the system scalability, flexibility, and reliability. In addition, the research discovered gaps in cross-cloud communication, data synchronization, and advanced security measures putting the groundwork for future exploration. These decisions underscore the importance of a multi-cloud CI/CD approach in building resilient, scalable, and secure systems that can ready to the evolving needs of modern enterprises while maintaining operational excellence.

# 8 Future Work

Future enhancements of the project can include the adoption of advance Kubernetes services for scalable container orchestration, enabling advanced workload distribution, automated scaling, and fault tolerance and also used of mesh technologies that can improve cross cloud communication and traffic routing to reducing latency and enhance resilience. Testing features can be expanded with tools with cloud native testing solution. Such Aws device farm and google test lab that can validate applications across a different environment, devices, and network conditions. Additionally, security and compliance in different cloud providers are remain critical concerns future studies could explore the solution of integration of threat detection mechanism that used AI to catch vulnerabilities in real time make sure the compliance with evolving standards and could be work to enhanced encryption techniques and implemented the zero-trust architectures to protect the data across multi cloud environment. Another area to explore for future research is the use of block chain technologies for improving transparency and security in multi cloud deployments. It provides

unchanged ledger to tracking resource provisioning and access logs which reduce the risk of unauthorized access. Finally, future research also expands the area of testing and monitoring by simulating very complex and more big application for workload to real world scenarios. To addressing this area will capable of meeting the demands of complex enterprise application to create more secure, scalable and efficient multi cloud environment.

# References


- Kim, J. and Wang, S. (2023) 'Multi-cloud infrastructure management: Challenges and best practices', *Journal of Cloud Computing*, 12(3), pp. 200–215.
- Tanzil, M.H., Sarker, M., Uddin, G. and Iqbal, A. (2023) 'A mixed method study of DevOps challenges', *Information and Software Technology*, 161, p. 107244. Available at: https://doi.org/10.xxxxx (Accessed: 27 January 2025).
- Tanzil, M.H., Sarker, M., Uddin, G. and Iqbal, A. (2024) 'A mixed method study of DevOps challenges', *arXiv e-prints*, pp. arXiv–2403.
- Karlsson, D. (2023) *Comparison of Infrastructure as Code Frameworks from a Developer Perspective*. PhD thesis. National College of Ireland.
- Ghosh, A., Srivastava, S. and Supraja, P. (2024) 'Streamlining multi-cloud infrastructure orchestration: Leveraging Terraform as a battle-tested solution', in *Proceedings of the 2024 International Conference on Cognitive Robotics and Intelligent Systems (ICC-ROBINS)*. IEEE, pp. 911–915.
- Farayola, O.A., Hassan, A.O., Adaramodu, O.R., Fakeyede, O.G. and Oladeinde, M. (2023) 'Configuration management in the modern era: Best practices, innovations, and challenges', *Computer Science & IT Research Journal*, 4(2), pp. 140–157.
- Gonçalves, J.M.M. (2023) 'Automatic deployment solution for multi-cloud environments', *Journal of Cloud Development Practices*, 6(1), pp. 45–56.
- Chavan, S. and Khadkikar, R.M.D.P.A. (2023) 'Accelerating cloud-native applications: Automated Kubernetes cluster deployment', in *Proceedings of the 2023 Kubernetes DevOps Summit*. IEEE, pp. 123–132.
- Pessa, A. (2023) 'Comparative study of infrastructure as code tools for Amazon Web Services', *Journal of Cloud Computing*, 12(1), pp. 55–72.
- Bafana, M. and Abdulaziz, A. (2024) 'Immutable infrastructure in practice: A comprehensive guide to AWS deployment', *Asian American Research Letters Journal*, 1(1), pp. 45–57.
- Sokolowski, D. and Salvaneschi, G. (2023) 'Towards reliable infrastructure as code', in *Proceedings of the 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, pp. 318–321.
- Kalliomaai, N. (2024) 'Choosing the right IaC tool for building reusable cloud infrastructure', *Journal of Infrastructure Management*, 7(2), pp. 112–130.
- Farah, S. and Patel, R. (2024) 'Securing containerized applications in multi-cloud environments using Trivy', *Journal of Security in Multi-Cloud Environments*, 8(1), pp. 45–60.

- Olaoye, G. and Luz, A. (2024) 'Future trends and emerging technologies in cloud security', *Telecommunication Engineering Centre Journal*, University of Melbourne, 5(2), pp. 198–209.
- Nawagamuwa, J. (2023) *Infrastructure as code frameworks evaluation for serverless applications testing in AWS*. MSc thesis. University of Sydney.
- Nguyen, T. and Lee, K. (2024) 'Comparative analysis of CI/CD tools in multi-cloud environments', *Journal of Software Development*, 15(3), pp. 112–123.
- Obi, O.C., Dawodu, S.O., Daraojimba, A.I., Onwusinkwue, S., Akagha, O.V. and Ahmad, I.A.I. (2024) 'Review of evolving cloud computing paradigms: Security, efficiency, and innovations', *Computer Science & IT Research Journal*, 5(2), pp. 270–292.