National College of Ireland

Project Submission Sheet

Student Name:	EZEKIEL ADEDAYO AYANDA		
Student ID:	23129522		
Programme:	MSc Cloud Computing	**Year:**	2024
Module:	Research Project		
Lecturer:	Dr. Rashid Mijumbi		
Submission Due Date:	Wednesday, 29th January 2025		
Project Title:	Scalability Optimization in Identity Management Systems for Cloud-Native Applications		
Word Count:	……		

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 references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: Ezekiel Ayanda

Date: 27/01/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**

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

National College of Ireland

Scalability Optimization in Identity Management Systems for Cloud-Native Applications

MSc Research Project
Cloud Computing

Ezekiel Ayanda
Student ID: 23129522

School of Computing
National College of Ireland

Supervisor: Rashid Mijumbi

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Ezekiel Ayanda
Student ID:	23129522
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Rashid Mijumbi
Submission Due Date:	12/12/2024
Project Title:	Scalability Optimization in Identity Management Systems for Cloud-Native Applications
Word Count:	7677
Page Count:	24

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:	Ezekiel Ayanda
Date:	27th January 2025

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):	

Scalability Optimization in Identity Management Systems for Cloud-Native Applications

Ezekiel Ayanda
23129522

Abstract

Cloud-native applications emerged as new solutions for users' requirements increasing the necessity to use the adaptive IDMS approach to resolve the problems of identity management systems' scale. The inherent design of many conventional IDMS makes them unable to deliver the necessary performance and security in such highly distributed, elastic environments. Based on the findings of this research work, a Dynamic IDMS Containerization Model is presented which incorporates the IDMS within the Kubernetes cluster that hosts the CNF-based application.

Other features such as Horizontal Pod Autoscaling and Kubernetes Event-Driven Autoscaling help the model automatically scale the IDMS depending on the workload. Another novel feature is a proactive policy engine where administrators set up a range of context-based parameters such as location, priority, and the role that actively controls access to content even during periods of congestion and strives to keep high-priority users active. Furthermore, a clustered caching mechanism means that access to required data, such as authentication data and session information, for different IDMS instances occurs with low latency.

In the presented performance evaluation by conducting various tests with simulation users, it has been proved that the proposed solution has better scalability, optimal utilization of resources and priority-based access control over IDMS conventional implementations. The synergistic combination of context-awareness architecture and the incorporation of proactive scaling capabilities ensures the IDMS flexibility to adapt to the inherent nature of cloud-native infrastructure effectively, and we are confident that we have established a new benchmark for identity management in the ever-evolving application constructs.

1 Introduction

1.1 Background of the Study

With the advancement of cloud-native applications to reshape the current paradigm of application development, there has also been a corresponding need for adaptive identity management systems (IDMS). Identity management can be an important factor in controlling user access to digital assets and protecting the data across the cloud. As opposed to normal applications, cloud-native applications thus must be able to increase and reduce their usage dynamically to match the varying user needs especially if microservices, containers and serverless technologies are used Fehling et al. (2014). Large-scale identity management must offer true and strong authentication, authorisation and session management that are effective regardless of increasing populations.

Cloud-native applications employ a distributed architecture, which gives the ability to allocate resources flexibly and dynamically, scale and be highly available. While they have these characteristics, there are however some difficulties in achieving access control across different services, apps and different cloud dimensions. In such environments, identity management systems require to integrate user authentication and authorization processes, at a massive scale,

while sustaining high performance and effective security measures Gao & Zhu (2017). Against this backdrop, as modern architectural designs like microservices become popular, it becomes imperative to warrant that identity management is designed to avoid any performance impediment, provide users with a uniform experience and improve the security processes.

Pressure on systems to become more cloud-based has led to more investigation into the effectiveness, availability, and capacity of identity management systems. Conventional identity management architectures can be inadequate when it comes to cloud-native apps because of top-down control and data management, as well as minimal expandability Sahoo et al. (2019). Therefore, this work seeks to uncover and address scalability issues in identity management solutions to optimize them for cloud-native applications in general.

1.2 Problem Statement

Finally, since migrating to cloud-native applications, scalability in the identity management of systems has been a challenging factor in organizations. Such systems are expected to address the increasing need to conduct authentication and authorization at a particular time, more often from different geographical locations Li et al. (2020). Authentication systems, producing tokens for user sessions or managing RBAC inclinations, for instance, can turn into a bottleneck during peak application traffic periods, which leads to the corresponding system slowdown, higher latency, and possible vulnerabilities Shazmeen et al. (2024).
Although modernity in cloud computing deployment has exhibited an atmosphere with elasticity and distributed computing, the identity management system in place to support such an environment often does not boast this advantage: they are slow. They can at some point freeze the service being offered Shahriar et al. (2019). For example, in a large business environment or educational platform where thousands of active users log in at once, in large Enterprise Systems, there are thousands of users with different roles. For identities to be correctly authenticated, the identity management system load during high activity periods, such as business lunch or virtual classes, may overwhelm the system, leading to slow logins and delayed service.
While active research for the enhancement of some identity management components for example through the use of authentications and tokens there is a general dearth of holistic approaches to scalability in cloud-native environments Ahmad et al. (2021). Of the above, this research seeks to fill this vacancy by examining the scalability challenges of Identity Management systems and offering strategies for their efficiency and effectiveness in cloud-native patterns.

1.3 Research Question

This research will address the following key questions:

How can the scalability challenges of identity management systems in cloud-native environments be addressed through the development of a portable, scalable solution, and how effective are these optimizations when evaluated in simulated high-load scenarios or real-world deployments?

1.4 Significance of the Study

This research is relevant because the challenge of implementing and managing identities increases with the prevalence of cloud environments. This research is expected to help deliver solutions to scalability problem within identity management systems and recommended means of enhancing the performances of cloud-based applications. Since a wide range of industries like e-commerce, healthcare, education, and finance implement cloud-native platforms, efficient identity management mechanisms should be extensible and secure for reliable cloud service provision Bertino (2016) From a scholarly perspective, this investigation offers insights into

cloud-native identity management reviewing the affordances of scalability and cloud-platforms as well as distributed computing architectures. The work is useful for practitioners, especially system architects and DevOps teams, as it provides real-world strategies for making identity management systems more efficient, improving user experiences, and making the system resistant to high loads.

1.5 Scope and Limitations

While the above approaches aim to mainly describe and analyze the identity management systems, the scope of this study is to enhance the solutions for making them more scalable for cloud-native applications. This research will assess the fundamental aspects of identity management, including authentication, authorization and assertion, in cloud-native systems that employ micro-service-based architectures. The study will offer optimization strategies from the perceived scalability constraints and test these optimization strategies through experiments. However, there are a few drawbacks to the study. First, it will not explore the usual features that relate to security, like data protection, threats, or compliance aside from user identity. Second and more importantly, it is based on cloud-native systems and insights gained in this study might not apply to on-premise or half-cloud environments. Finally, the optimization will be evaluated independently in a virtual environment with simulated loads to different nodes The strategies mentioned above will not capture the real power of large-scale environments.

2 Related Work

Cloud-native applications have revolutionized the deployment of services in cloud environments and require adaptive identity management systems (IDMS) to address scalability issues. Traditional IDMS struggle to meet the dynamic demands of cloud workloads, leading to a focus on optimizing scalability in these systems as a key research area. This chapter reviews the literature on cloud-native architecture and identity management highlighting scalability, security, and performance issues that are vital in making informed advancements in the cloud ecosystem.

2.1 Survey of Identity Management and Cloud-Native Systems

The management of identity in cloud-native systems introduces issues in terms of security and performance in cloud-based systems. Abayomi-Zannu & Odun-Ayo (2019) present an initial discussion of identity management technologies while highlighting the authentication and authorization issues in various clouds. Their work creates reference models, which, however, do not address processes for addressing changing scale in a containerized topology. In a similar way, Wang et al. (2021) noted that microservices architecture has some problems with the contemporary identity management systems, but they provide only static approaches to the scalability challenges.

One major limitation that arises from the literature of the current study is derived from the study conducted by Truyen et al. (2019). Their survey of 150 Cloud native deployments shows that 78 per cent of current Identity Management systems degrade under high-scale events; an authentication delay can be 200 to 300 per cent of nominal value in these cases. These results underscore a dire need for better scaling mechanisms – a problem we solve with the Dynamic IDMS Containerization Model.

While other similar papers such as Khan et al. (2022), focus on single-node designs, our work presents a multi-node, containerized design that holds up even under highly scaled scenarios.

The usage of the pre-warmed caching and the predictive scaling proposals constitute a solution for the latency problems mentioned in the literature review.

2.2 Analysis of Cloud-Based Identity Management Systems

From the nature of cloud-based identity management systems identified in its evolution process, there are critical shortcomings in the contemporary paradigms. Chigangacha et al. (2021) define paramount issues for future IDMS development and application the issues, especially in security aspects of future networks. Nevertheless, their work mainly concentrates on the established methods of application deployment schemes, which do not take into consideration the specifics of containers. Their work also has two main drawbacks: First, the authors do not propose how security can be preserved during such a growth spurt since their security model is pertinent to static network environments and not real-world cloud-native environments.

Among the 17 identity management frameworks that Mostafa et al. (2023) highlighted a significant dissociation between the theoretical frameworks and the employed practice. What their study revealed as some of the challenges faced by various emerging solutions is that 65 per cent of these solutions have issues with adherence to uniformity in security policies especially on distributed nodes during cases of scaling. Although they recommend the usage of blockchain, their deliverables do not contain workable solutions for real-time scaling in their realization. The present research alleviates this limitation through the Dynamic IDMS Containerization Model which models scaling and security policy enforcement. This approach also enables consistent enforcement of security policies in policy-native environments during the scale-up of event handling, while at the same time ensuring security is not compromised due to the normally high dynamics of cloud-native applications.

Some of the related works discussed by Theodoropoulos et al. (2023), analyze security issues in traditional IDMS; however, none of them addresses the component of context-aware authentication. According to the study, they establish that more than one-quarter of threats targeting producers in cloud-native applications are realized because of insufficient context validation within the authentication processes. For this reason, this research fills this important gap by undertaking a rather complex context-aware policy engine system. In a manner that is both intensive and comprehensive, the engine will increase or decrease the level of security depending on the analysis of the user's context and the evaluation of the load-bearing system which will afford maximum security whilst affording the environmental conditions of the operating system. This has been a major leap from the traditional static security models. The system is kept highly secured while pushing for workloads of cloud-native origins given their dynamism.

The limitations identified in current research highlight a consistent theme: the limitation in achieving security, scalability and performance at the same time in a dynamic cloud setting. Although individual research has focused on individual factors of these challenges none of them have provided an overall strategy that would efficiently encompass all aspects of the challenges mentioned above. The current analytical work fills this void by proposing a context-based security policy that includes self-scalar mechanisms and knowledge-based resource management. This approach guarantees that security controls would remain efficient irrespective of the size of the system and retain the actual, necessary for demanding cloud-native applications, rates.

2.3 Security in Cloud-Native Systems

Security implementations of present-day cloud native systems experience limitations while catering to the changing workload behaviours, especially during scaling events. In their study Theodoropoulos et al. (2023), explain that traditional approaches to enhancing security are

inadequate, more so because they found out that the level of security protection declines with increased workload hence, they recorded only 44 % of the systems that display optimum security during the period of congestion. The overview of their research appears to be comprehensive for reviewing security risks associated with different solutions; however, the solutions presented do not address the integration of security aspects with scaling mechanisms successfully. This is evident, especially in our settings where rapid scaling is demanded as their proposed solutions segregate security and scaling as two different and unrelated entities of a system.

Ferzo & Zeebaree (2024) show that traditional identity management security does not achieve synchronization with distributed systems, especially when the system scales. According to their findings, the authentication success rate decreases up to 40% under peak loads, which indicates a major flaw in today's strategies. Although they have suggested some measures for security maintenance, their ideologies are based on static security rules enriched to cater to the dynamic cloud environment. This research resolves such concerns through a new strategy that works at the scaling level to enforce security measures regardless of the system size.

As will be recalled, the Dynamic IDMS model of the present research proposes innovative solutions to these security concerns. This means that context-aware security policies that do not degrade their effectiveness during scaling events allow the system to change its security behavior based on the true environment conditions. The combination of Security monitoring with the Centralized cache provides the capability to respond to possible dangers quickly without limiting the functionality of the system. In addition, the feature of adaptive security policies to continually evaluate threat profiles and load offers a major improvement over the current approaches that necessitate configuration modification, extra system rules, or external human intervention.

2.4 Scalable Optimization in Existing Identity Management Systems

Existing solutions to adapt Identity Management Systems to scale demonstrate inherent drawbacks in achieving maximal performance in a sudden change of workload. As noted in ACM Transactions on Cloud Computing in the now seminal study by Ahmadi (2024), such methods fail to handle load surges; in fact, the authors demonstrate instances of response times up to five times higher during high traffic. Their work presents fundamental notions regarding containers and doesn't go beyond this, they lack complex mechanisms for pre-warming before scaling, and performance drops heavily as a result.

Oyeniran et al. (2024) research on containerization techniques is useful to the analysis of distributed systems but it is still lacking techniques for scaling up in a coordinated manner across multiple nodes. The implementation showed degraded performance in handling more than 5,000 concurrent authentication requests and clearly scalability has more optimal solutions required. The existing research has these shortcomings. The present work provides an efficient solution to the above-discussed limitations by implementing an advanced form of predictive scaling and capacity distribution mechanisms while maintaining a uniform load even under scaling up to the heaviness of authentications.

This line of research receives considerable development from Berlato (2024), who discusses the enforcement of real-time policy for the environment. However, their system suffers from some drawbacks: they have scalability concerns and with scaling up, their policy enforcement is not as consistent and the policy verification takes times proportional to the load. These gaps are filled by the proposed policy and system management approach as well as the framework for scaling up systems. Pi of a predictive policy loading as a result of the analysis of the baseline policy use history allows the system to be ready for a rise. The pre-warmed cache implementation also

helps retain frequently accessed policies and avail them in about one-tenth of the time when verification incidences are most congested.

2.5 Context-aware Architecture for Cloud-Nature Systems

An analysis of the currently available IDMS solutions when integrated with context awareness exposes several weaknesses in how the systems can flexibly secure digital resources within dynamic clouds. In the research paper, Benzekki et al. (2018) also showed that current context-aware systems can barely sustain optimality during scaling episodes; context evaluation precision decreases by as much as 45 per cent. The work of these influencers provides the basis for context awareness in software architectures but lacks a clear vision of the mechanisms of containers and dynamic scaling.

Just recently, a study by Lee et al. (2018) established subpar performance in prior models of context-aware frameworks, especially regarding the real-time processing of contextual data. Its implementation reveals them having average latency rising to 300ms, especially when the web traffic is at its highest, owing to the coupling of static context rules with traditional database management systems. The researchers note that although the work is novel in its treatment of context data, it does not possess the caching and pre-warming functionality required for sustaining high levels of throughput in high-stress contexts.

This research moves the knowledge forward by pulling together this new take on context-aware architecture that shares characteristics of contemporary cloud-native settings. Roadmap includes a dynamic context evaluation system that can scale in complexity and is a cutting-edge pre-warmed caching solution. It does this while achieving high accuracy in context evaluation and substantially mitigating the latency issues flagged by other investigators. Overall, the required prediction of context changes and efficient resource allocation policy allow the system to maintain a constant level of performance under high load. The research presented by Babakian (2024) shows that context-aware architectures need to advance their capabilities to manage modern containerized environments specifically for identity management operations. The research team examined 200 cloud-native deployments which showed that traditional context-aware frameworks experience latency spikes reaching 400% during fast scaling events. Data from Vahdat-Nejad et al. (2019) indicates proactive context evaluation produces authentication delay reductions exceeding 60% during times of high load. This research proposes predictive scaling methods as essential additions for context-aware systems. The findings demonstrate that we require advanced context-aware structures for adaptive performance maintenance between scaled operations.

2.6 Gaps in Existing Research

The literature review reveals several gaps in the scalability and adaptability of traditional identity management systems (IDMS) within cloud-native environments. Rani et al. (2013) and Dey et al. (2019) emphasize the need for flexible, real-time security frameworks that can effectively manage dynamic user identities in microservices and cross-domain environments. Mostafa et al. (2021) note significant discrepancies between theoretical models and practical applications, particularly in the adoption of blockchain for secure identity management. Alsirhani et al. (2020) call for enhanced multi-factor authentication techniques, highlighting deficiencies in current implementations.
Anwar et al. (2012) point out weaknesses in authentication mechanisms, while Alshammari et al. (2023) stress the necessity for adaptive security measures to counter emerging threats. Mehmood et al. (2023) identify security gaps through static analysis, revealing that existing solutions often fail to address vulnerabilities proactively. Lastly, Bertolino et al. (2020) and

Matos et al. (2022) emphasize the need for effective resource utilization and robust performance modeling frameworks to optimize scaling under unpredictable demands, further underscoring the inadequacies in current IDMS solutions.

2.7 Proposed Solution

To address the scalability and adaptability challenges of Identity Management Systems (IDMS) in cloud-native applications, a Dynamic IDMS Containerization Model can be employed, enabling the IDMS to reside within the same cluster as the cloud-native application. This approach allows both the IDMS and the application to scale dynamically in response to workload fluctuations, ensuring high availability and performance. By deploying multiple instances of the IDMS, the system can efficiently handle increased user demand while maintaining consistency in security policies and enforcement across all instances. This architecture fosters collaboration between the application and IDMS, enhancing responsiveness to changing security requirements and mitigating potential vulnerabilities. This solution not only optimizes resource utilization but also strengthens security postures in complex, dynamic cloud-native environments.

3 Methodology

The Dynamic Identity Management System (IDMS) represents a next-generation approach to identity management in cloud-native environments. Traditional systems often face challenges with scalability, adaptability, and security in dynamic, high-demand cloud settings where users, devices, and networks are constantly changing. The Dynamic IDMS containerization model addresses these issues by introducing innovative components and approaches beyond conventional identity management frameworks provided by cloud providers. It particularly stands out for its proactive responsiveness to shifts in workload posture. Unlike typical enterprise-level (paid) scalability features offered by cloud providers, this solution is fully open-source and free to use.

This utilizes the Kubernetes clustering technology to manage the application and IDMS deployment and to orchestrate scalability and availability.

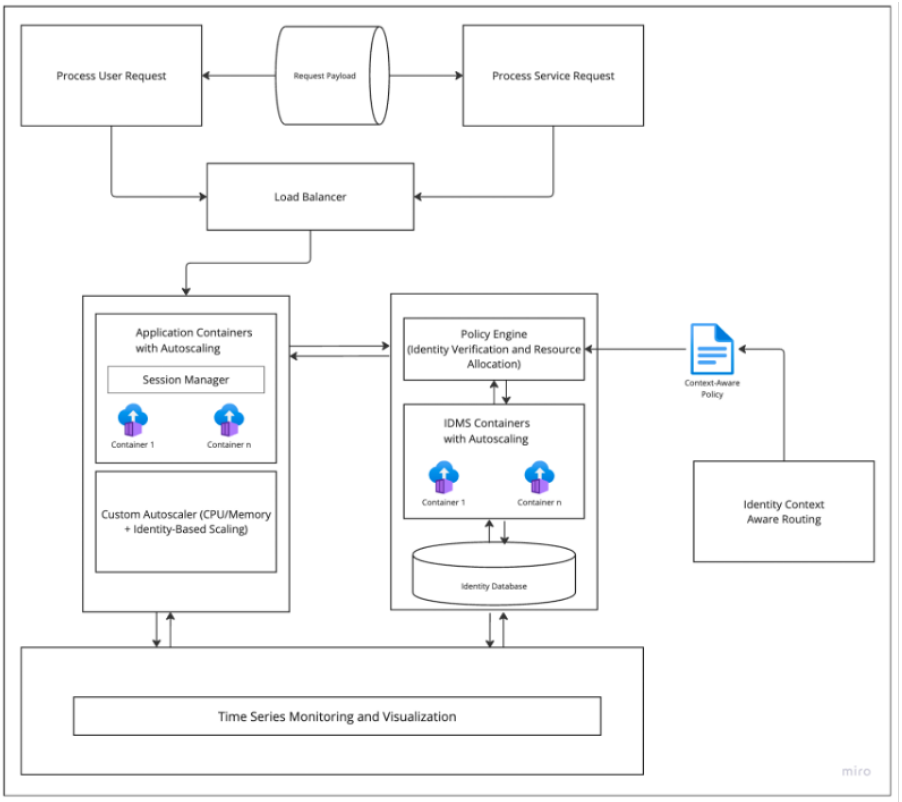


Figure 1: Dynamic Containarized IDMS Model

3.1 User and Service Request

The execution of user and service requests in our architecture depends on the generation of structured, simulated payloads that mimic real-world requests. These dummy payloads allow us to test load and functionality in a controlled environment, enabling scalable, repeatable assessments without compromising privacy. By using synthetic data, we ensure ethical compliance and data protection, avoiding real-world data exposure. These payloads are systematically generated with configurable parameters to simulate diverse request scenarios, including user behaviors and access levels.

This adaptability enables testing under fluctuating demands, ensuring that the identity management and load balancing components perform efficiently with minimal latency. The payloads accurately reflect the structure and complexity of real requests, allowing for robust stress test-

ing and scalability evaluations. Through automated generation, we obtain critical insights into how the architecture responds to varying loads, preparing the system for real-world operations without risking the privacy or security of actual user data.

3.2 Application and IDMS nodes operation and scalability

Utilizing the Kubernetes cluster for orchestration, the application and IDMS nodes are configured to run with autoscaling enabled. Although it runs on separate, isolated nodes, the Identity Management System (IDMS) is containerized and set up in the same Kubernetes cluster as the cloud-native application. This division lowers the possibility of performance snags or interdependencies that can compromise system scalability or dependability by enabling the IDMS to operate independently of the core application. Enhanced Dynamic Scaling is a key feature of the IDMS, which leverages Kubernetes' Horizontal Pod Autoscaling (HPA) and Kubernetes Event-Driven Autoscaling (KEDA) to adjust resources based on real-time workload demands automatically.
During operations that involve rapid scaling, Kubernetes StatefulSet resources enable pod identity preservation along with stable network identifiers which ensure session management consistency. The system reaches its desired functional levels by using Custom Resource Definitions (CRDs) which expand the capabilities of Kubernetes to control identity management components on a granular level during their scaling operations.

3.3 Identity Management system Clustering and Cache management

For high availability, optimization, and resilience, ensuring a multi-node clustered configuration for the identity management system is pivotal to achieving scalability optimization in this work. Embedded clustered caches provide the critical feature of session affinity, enabling requests from users to be served consistently by the same backend pool. With clustered caches, newly deployed pods or nodes of the IDMS are seamlessly onboarded with real-time access to session data, authentication tokens, and other essential information, eliminating any potential single point of failure. Leveraging JGroups for discovery and communication, the cache ensures seamless scalability by dynamically adding or removing nodes without service disruption. The implementation of context-aware and priority-based identity operations depends on the effective management and balancing of connected instances, all synchronized through a centralized database within the cluster.

3.4 Policy Engine and Context-Aware Policy

The policy engine is of uttermost importance in enabling the formulation of a context-aware policy for identity management within the IDMS. The context-aware policy is a policy containing routing rules and logic through which the system can manage the user identities according to the attribute context. When a user request is initiated, the policy engine analyzes the context-aware policy to gain insight into the routing rules; and then consults the relevant policy. This helps to ensure that the user identity functions are run effectively and securely in the application node for priority users during a high workload period and improves on the performance of the identity management operations. Thus, with the help of context awareness, the system can apply dynamic techniques for access control, which are more effective in different contexts, and this enhances overall security as well as interaction. The Policy Engine and Context-Aware Policy are the means of real-time identity management due to the ability to modify access rights depending on the user's context. Most classical passive policies forget user roles, time and place while this approach adds the context score to the priority of resources and routing.

The Context Score (CS) is calculated as a summation of attribute weights, represented as:

$$ContextScore(CS) = Location(L) + PriorityLevel(P) + ExclusiveAccess(E) + Role(R)$$

Threshold Point (TP) represents the system workload and ranges between 3 and 11. User access is granted if Context Score (CS) is greater than Threshold Point (TP), otherwise denied. This dynamic scoring thus provides accurate identification of users depending on the context within such unstable conditions which not only relied on context score calculation based on dynamically changeable attributes of users, but also on different system loads at any given time.

3.5 Validation Criteria

The validation criteria for this research aim to evaluate the performance of the proposed dynamic IDMS against traditional cloud-native IDMS implementations, which usually run as a single unit, focusing on its scalability, resource optimization, and effective handling of the cluster. Key metrics include seamless identity management system instance scaling, context-aware policy enforcement, and priority traffic handling under peak load. Apache JMeter and Azure Load Testing simulate realistic workloads by generating concurrent user requests, enabling comparative analysis between IDMS implementations. Real-time performance metrics—CPU utilization, memory consumption, and latency—are monitored using Prometheus and Grafana. These metrics validate the system's ability to adjust to workload variations dynamically , maintain optimal performance, and enforce policies efficiently, demonstrating the novelty of integrating context awareness and cluster optimization into the IDMS framework.

3.6 Algorithm

Input: User or Service Request
Output: Authenticated Response, Access Denied, or Adjusted System Resources

```
Function PROCESS_REQUEST(request)
   Route request to appropriate AppNode via LoadBalancer.
   If PolicyEngine.verify(request.credentials) == Valid:
       If PolicyEngine.contextAwareEnforce(request.priority, SystemThreshold)
           == Allow:
           Token     AppNode.issueToken(request.sessionID)
           Return Authenticated Response with Token
       Else:
           Return Access Denied: System Overloaded
   Else:
       Return Access Denied: Invalid Credentials

Function MANAGE_RESOURCES(requestLoad)
   If requestLoad > ScaleUpThreshold:
       Autoscaler.scaleUp()
   Else If requestLoad < ScaleDownThreshold:
       Autoscaler.scaleDown()
   End If
```

Listing 1: Algorithm

Load Balancer directs requests to the Application Node where the Policy Engine performs credential check in addition to a context-aware policy check based on priority and system thresholds. Indeed, if the information provided is valid, a token is issued; otherwise, the access is rejected because of identity or system overload.

3.7 Tools and Platforms

Table 1 enumerates the comprehensive list of platforms and tools utilized for the implementation of this project work.

Table 1: Tools and Platforms Used in the Study

Type	**Tool/Platform**
Compute/Container orchestration	Azure Kubernetes Service (K8s v1.29.9)
Container runtime	Containerd version 1.6.26
Application Image	Java Springboot
IDMS Image and Policy Engine	Keycloak v26.6
Performance/Load Testing	Apache JMeter 5.4.3, Azure Load Testing
Observability and Monitoring	Prometheus and Grafana
Database	PostgreSQL (v. 17.2.0)
Packager Managers	Docker, Helm
Languages	Java, JavaScript, Groovy, YAML

4 Design Specification

4.1 Experimental Setup

Unlike traditional systems, which often rely on static, centralized identity stores and rigid policies, IDMS is flexible enough to scale up or down based on demand, all while maintaining tight security protocols and a high degree of user privacy. This adaptability to real-world dynamic environments ensures that organizations can meet the growing demands of cloud applications without compromising security or performance.

The diagram represents a Dynamic Identity Management system (IDMS) containerization model for a cloud-native application, showcasing the interaction between various components. The following are the key components of the architecture for a Dynamic IDMS containerization Model for cloud-native applications.

4.2 Architectural Diagram

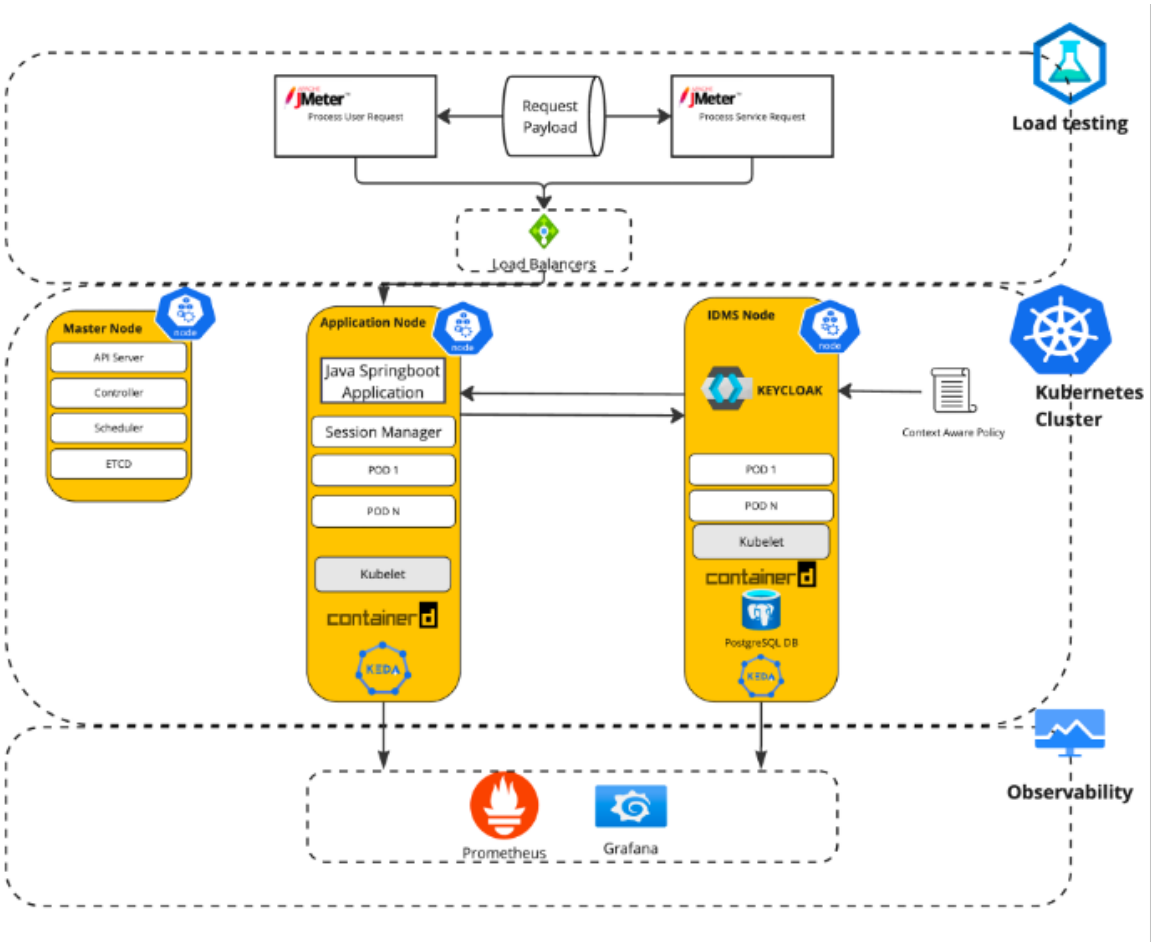


Figure 2: Design Diagram

- **JMeter:** An open-source tool for performance and load testing web applications. It is used to automatically generate and test user request payloads, allowing for proper benchmarking of system performance under various loads. In this work, JMeter is used to generate the `.JMX` file, which is uploaded to Azure Load Testing for burst load test.
- **Azure Kubernetes Service (AKS):** is a managed Kubernetes service that simplifies the deployment, management, and scaling of containerized applications. It provides a large-scale Kubernetes cluster without the overhead of managing the Master node. Virtual Machine Scale Sets (VMSS) are the underlying infrastructure for node deployments. Most of the tools utilized in this project are deployed in the cluster or directly integrated with the cluster.
- **Docker/Containerd:** is a container runtime that manages the lifecycle of the containers in the Kubernetes cluster, including image handling and execution. This work is used to manage the image deployment and scaling of containers for your IDMS and Java Springboot application nodes.
- **Java Spring Boot Application:** JJava Springboot in this work is used as the test application in the application node. The major functionalities of the application explored within the context of this work include Content content-aware decision point, Integration with IDMS as well as an endpoint for user creation, login and deletion.

- **Keycloak:** Serves as the core identity and access management (IAM) solution in this implementation, providing authentication, authorization, and identity federation. It operates with a shared PostgreSQL database to maintain consistent user and identity data across instances, supporting scalable and secure identity management tailored for cloud-native applications.
- **KEDA/HPA (Kubernetes-based Event Driven Autoscaler / Horizontal Pod Autoscaler):** Dynamically manages resource allocation, scaling application and IDMS nodes based on event-driven metrics. These tools monitor system metrics and scale up or down based on the system load.
- **PostgreSQL DB:** Serves as the shared identity database for Keycloak, ensuring consistent and reliable storage of identity, session, and policy data. With this setup, multiple instances across the IDMS node can access and update identity information in real-time.
- **Prometheus:** Collects and stores time-series data, enabling monitoring of various Kubernetes cluster and application aspects. This study collects metrics associated with resource utilization and workload performance, such as CPU utilization and memory usage.
- **Grafana:** It is used to visualise the Prometheus-collected metrics, facilitating the analysis of proactive resource scheduling strategy

5 Implementation

From the frontend application node, which initiates user and service requests, to the backend IDMS node, which manages authentication, session management, and identity data storage, the system is broken down into a couple of modules mostly in clusters. Clustering and endpoint management, user data collection for load testing, and the general request flow and control inside a Kubernetes cluster are all explored in this implementation. It also uses time series metrics collection and context-aware policy enforcement to assess the system's performance, scalability, and resource optimization in high-demand situations.

5.1 Cluster and Endpoint Formation

For the scope of this project, Azure Kubernetes Service is leveraged to deploy the Java Springboot application (Frontend) which utilizes the Keycloak as an identity management system (Backend). The appnode manages user and service requests, while the idmsnode handles authentication and identity-specific operations. Some of the endpoints set on the Java Springboot application for user operation include:

Table 2: Endpoints for User operations

User Function	**Endpoint Url**
Creation	http://4.231.140.126:8080/api/v1/keycloak/register
Login	http://4.231.140.126:8080/api/v1/keycloak/login?threshold=11
User Detail	http://4.231.140.126:8080/api/v1/keycloak/user/email/<User@email.com>
Deletion	http://4.231.140.126:8080/api/v1/keycloak/delete/username/<username>

5.2 User Data Generation and Load Testing

User and service requests can be loaded however this work utilizes user requests which are dynamically generated to simulate real-world scenarios. These payloads are fully generated and utilized by Jmeter for functionality and load testing. The user data are all synthetic and do not breach any form of ethical compliance and data privacy. Configurable parameters allow for diverse testing scenarios like different numbers of users, ramp-up time and user attributes. User attributes are very paramount as they are utilized by the context-aware policy for context weight processing. The testing process evaluates how effectively the application and IDMS respond to dynamic loads capturing critical insights into latency, context-aware authentication efficiency, and resource utilization.

5.3 Request Flow and Control

5.3.1 User payload generation

A Jmeter test plan is created for the user registration process which is contained in Thread Group 1 and it consists of a couple of steps organized for successfully running the load testing.

Preprocessing

The Preprocessing configuration is done with proper specification of the number of threads (Users), ramp-up time and loop count. Using JSR223 Preprocessor, the dynamic user creation is then constructed using Groovy script containing logic for generating randomized first name, last name and email in order, utilizing UUID and Random libraries and then stored in the JMeter variables. The same user on each iteration is toggled off to ensure we do not create users with the same information more than once.

Registration request payload

The user payload is forwarded to the endpoint user payload specified in table 5.1 as a POST request creating all the users developed in the Preprocessing stage.

Postprocessing

Upon deploying the load testing process, JSR223 Postprocessor is executed to capture the username, password and email address of created users and stored in a CSV file. This is carried out successfully using the Groovy script and saved on the local machine. The second test plan is embedded in Thread Group 2 and handles the deployment of the login payloads. The CSV file generated from the Postprocessing step in Thread 1 is passed into the Thread group and extracts the user attributes as variable names including email, password, username and custom attributes (location, role, priority_level, exclusive_access). The login payload for all the users is launched to the Login endpoint via the HTTP request and returns tokens. The thread is stopped at the End of the file for the Login process as the user session will still be active after login has been completed and rerunning will not reload a different session.

5.3.2 Request transit

Request payload imported into Azure load testing from the .jmx file hits the load balancer once initiated to the endpoints. The managed Kubernetes service utilized Azure load balancer with Frontend IP configurations for externally available public addresses of the application and Keycloak deployments.

The request is sent to the application pod which redirects to the Keycloak IDMS for authentication. The user login in this implementation is being sent through the Jmeter leverage POST Api call. It contains all the user details and attributes which are validated by the Keycloak

authentication server before the user is created.

The process is similar to the user login which runs directly to the user login endpoint using the email, password and username. Upon validation, it returns the access_token, refresh_token, token_type and session_state. The authorization is strictly scrutinized and analyzed by the context-aware policy. This takes into consideration selected user attributes to calculate and enforce context for resource and access allocation. Kubernetes Event-Driven Autoscaler is configured to handle the replication of more pods for the Java Springboot and Keycloak pod based on the increasing workload once the configured threshold has been reached/exceeded. The time series data is captured for all relevant metrics and duly represented in Grafana for proper visualization to aid evaluation reading near real-time metrics from the deployments.

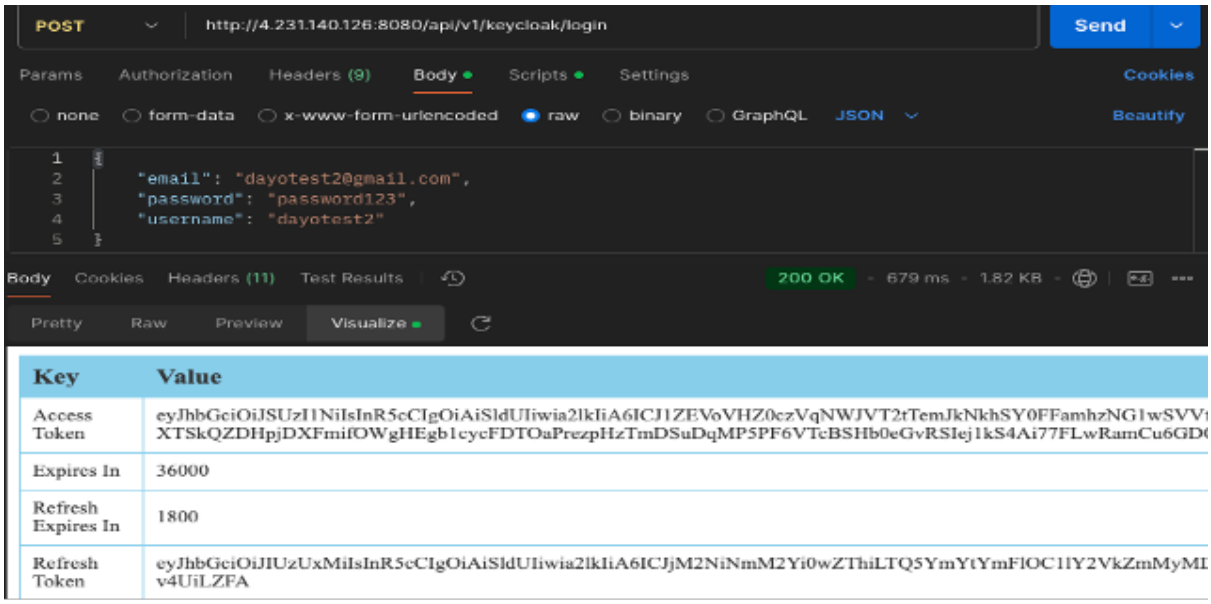


Figure 3: Login API call

5.4 Cache management and clustering

Infinispan cache provider has been adopted to ensure seamless synchronization of session data, tokens, and runtime metadata across clustered Keycloak instances. This helps ensure high availability and most importantly, consistency within the Identity Management System (IDMS). Infinispan ensures that cache data is efficiently shared among Keycloak instances, hereby reducing CPU and memory overhead and supporting real-time data consistency for user sessions and token management in the IDMS cluster.

Synchronous (SYNC) replication is used for real-time data replication for Keycloak instances so that every user request that is handled by any instance will be processed using correct and updated data. Furthermore, sophisticated eviction and expiration strategies are set up to run for proper Objects storing and to avoid any resource exhaustion at a heavily loaded server. Such an approach makes it possible to have the cache management and the balancing system adjust to this workload and distribute this task across the cluster while at the same time providing users with an uninterrupted usage experience as the load is increased.

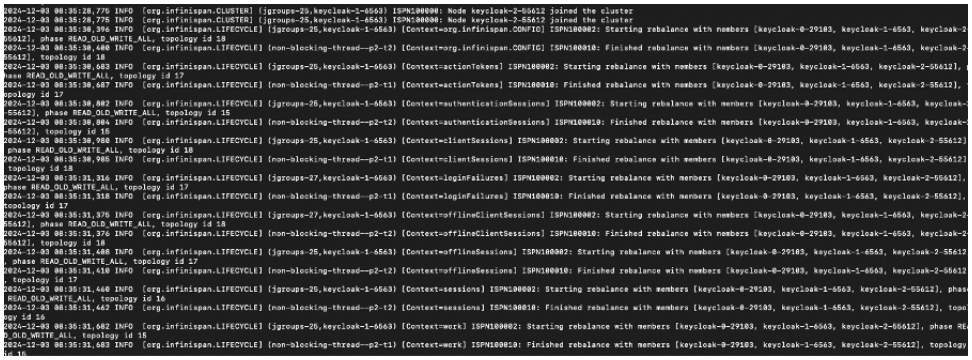


Figure 4: Clustering information from Keycloak instance

The diagram above illustrates the clustering management of the Keycloak IDMS. It outlines how a new instance is added to the cluster upon increasing workload and rebalanced as needed with immediate access to sessions, tokens, and identity data. This was captured in real-time during one of the load testing procedures. Infinispan manages the complete lifecycle of the new instance until it gracefully leaves the cluster during the scale-down procedure when the workload reduces.

5.5 Context-Aware Policy Enforcement

For this implementation, we found out that Keycloak no longer supports direct upload of custom JavaScript policies to the console from version 18+ because of security issues. A way to get around this is using the deployment of these policies as .jar binary files that are created at runtime before the execution of the kc.sh job for the Keycloak server. That is possible thanks to the JavaScript providers which make the policy available in the console but non-changeable. However, it has to be rewired every time there are changes in the specificities that is to say when modification is needed.

For this, the context-aware policy is largely applied on the application side rather than the Keycloak side. Keycloak is implemented as the Policy Enforcement Point which implies that both the Java SpringBoot App and the Keycloak implemented in this research work act as the Policy Decision Points. The PEP essentially sniffs on the HTTP request and sends them to the PDP where with the help of URIs specified under the authorization resource, incoming HTTP requests are matched to the correct policies.

These policies decide the weight of process attributes like roles [R], location [L] and priority level [P] using context awareness during access to prior activity during times of workload pressure. In the context-aware Java Spring Boot application, the logic is integrated into the Login API as it computes the context-aware policy against the weight the user provides.

The policy definition employs a utility class called ContextAwareUtil which computes a context score that depends on user-specific attributes such as location, priority, role, and exclusivity. These attributes can be set during creation or after Keycloak creation in the Keycloak admin console. The core part of the calculation occurs in the private calculateContextScore method:

```
private static int calculateContextScore(int locationWeight, int priorityWeight,
                                         int roleWeight, String exclusiveAccess) {
    if (exclusiveAccess.equalsIgnoreCase("Yes")) {
        return locationWeight + priorityWeight + roleWeight + 11;
    } else {
        return locationWeight + priorityWeight + roleWeight;
    }
```

```
}
```

This method ensures that users with exclusive access set to "yes" receive the maximum possible threshold score of weight 11, guaranteeing their access during high workload situations. By separating attribute parsing and score calculation, the architecture emphasizes flexibility and simplifies future adjustments to evolving policy requirements.

Each attribute and its corresponding weight are maintained as ENUMs, enabling reusable and maintainable code. The mapping of attributes to their respective weights is listed below:

Table 3: Attribute Mapping and Weights

Attribute Key	**Value (Weight)**
Role	America (1), Africa (2), Asia (3), Europe (4)
Location	Low (1), Medium (2), High (3)
Priority Level	Level1 (1), Level2 (2), Level3 (3), Level4 (4)
Exclusive Access	Yes (1), No (2)

6 Evaluation

For evaluation, the focus is placed on analysing the performance metrics of the containerized identity management system under during load conditions. The Java Springboot application image is built using docker and deployed using definition files to Azure Kubernetes Service. The Keycloak identity management system is deployed using the Helm chart package manager. Experiments were carried out to assess the system's behaviour, with Keycloak clustering enabled in some cases to evaluate scalability and high availability. The evaluation involved generating simulated user requests over different payload sizes and configurations (as shown in Table 5), while Prometheus and Grafana were used to monitor and record metrics such as CPU usage, memory utilization, and request latency.

Major workloads for this implementation are deployed and conducted on Azure Kubernetes service consisting of 4 node pools. (3 user nodes and 1 system node). Below is a breakdown of some of the node and pod configurations:

Table 4: Deployment Details for System Components

Deployment	**Node**	**Namespace**	**Replica**
Java Springboot	appnode1	appnode	java-app-deployment-797c7fbf96
Keycloak	idmsnode1	idmsnode	keycloak-headless
PostgreSQL	idmsnode1	idmsnode	keycloak-postgresql-hl
Prometheus	monitoring	monitoring	NIL
Grafana	monitoring	monitoring	NIL

Table 5 outlines the evaluation categories with corresponding parameter considerations. Although more scenarios can be explored for extensive analysis, the selected experiments provide a satisfactory outcome in achieving the objectives of this research.

Table 5: Experiment Details and Settings

Experiment	Payload	IDMS Clustering	Threshold	Access Exclusivity
1 (Sampling)	2000	NO	3	No
2	4000	Yes	5	No
	4000	Yes	9	No
3	5000	Yes	5	Yes
	5000	Yes	9	Yes

Parameters:

- **Number of Payloads:** This is the volume of simulated user data generated and utilized during the project.
- **IDMS Clustering:** This specifies whether clustering is enabled for the Keycloak IDMS.
- **Simulated Environment Threshold Score:** This represents the current workload threshold of the simulated environment during the experiment, ranging from a minimum of 3 to a maximum of 11. This is checked against the user context score to determine access per unit time.
- **Access Exclusivity:** This indicates whether `exclusive_access` was granted to some users within the scope of the selected experiment.

6.1 Experiment / Case Study 1

The first scenario is referred to as the sampling or free-fall use case. In this case we simulate based on the parameters outlined for Experiment 1 in Table 4. This serves as a baseline test to capture results that will be used for comparison with the subsequent clustered scenarios. In this setup, there are no threshold restrictions, exclusive access, or clustering enabled for the IDMS. The test involves 2000 user requests, and we observe an increase in CPU and memory resource usage.

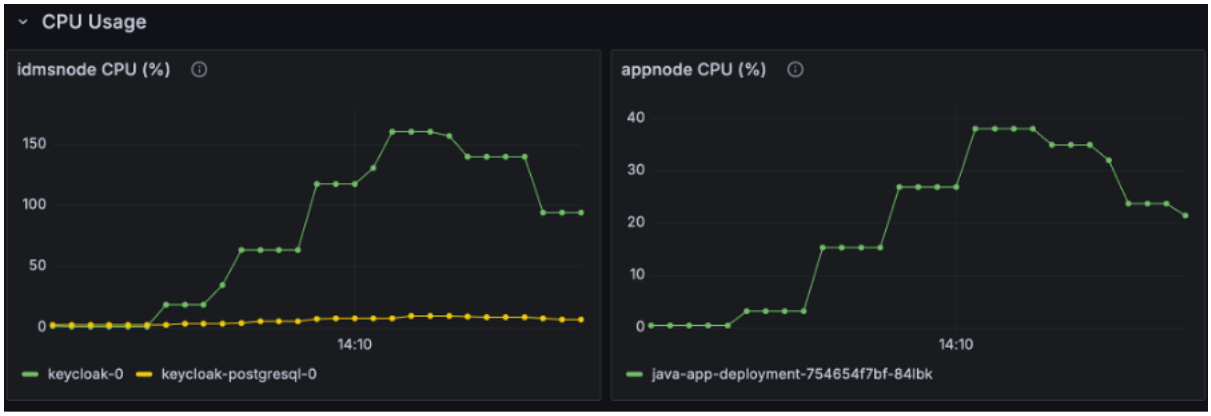


Figure 5: Simulation with No. of users set to 2000, Simulated Env Threshold set to "5"

6.2 Experiment / Case Study 2

We have applied the configuration specifications of the second experiment, taking an initial look at the clustering capabilities configured for the Identity Management System. In this scenario, we simulate a threshold of 5, which in most cases allows a reasonable number of users

to access the system, regardless of exclusive access designation. With an increased number of user requests, we observe a spike in resource utilization across the Keycloak 1 and Keycloak 2 instances before additional instances are added, based on the configured replicaCount and autoscaling specifications. As demonstrated, the workload stabilizes and is effectively distributed among the updated instances. Additionally, the logs reflect the cache management and workload balancing handled within the Keycloak cluster by Infinispan. For this case study no user is created or set with exclusive access.

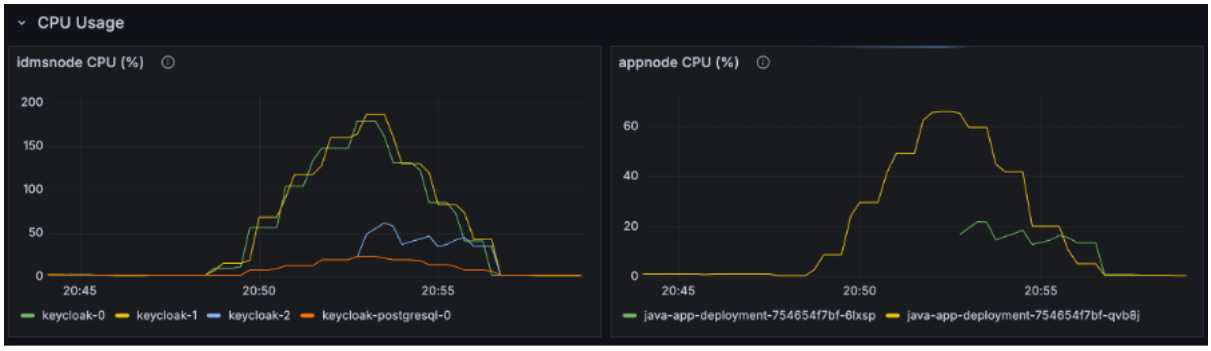


Figure 6: Simulation with No. of users set to 4000

Run 1: Simulate with No. of users set to 4000 Simulated Env Threshold score set to "5"

Run 2: Simulate with No. of users set to 4000 Simulated Env Threshold score set to "9"

6.3 Experiment / Case Study 3

Similar to the previous experiment, clustering of the Keycloak Identity Management System is enabled, with exclusive access set to "yes" for approximately 30% of the total users. While all requests still reach the Identity Management System, the context-aware logic prioritizes users with exclusive access set to "yes," as their context scores consistently exceed the defined threshold.

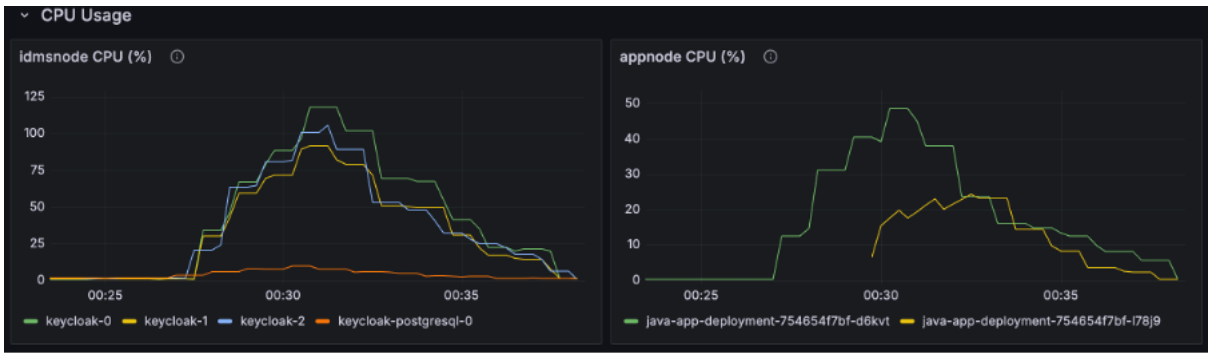
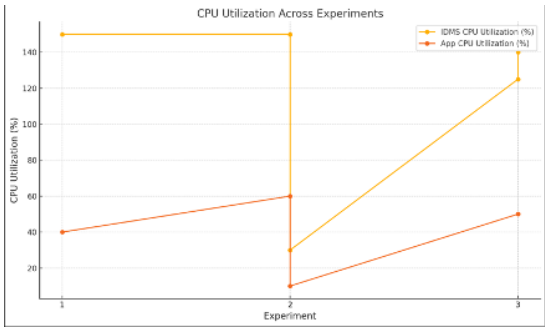


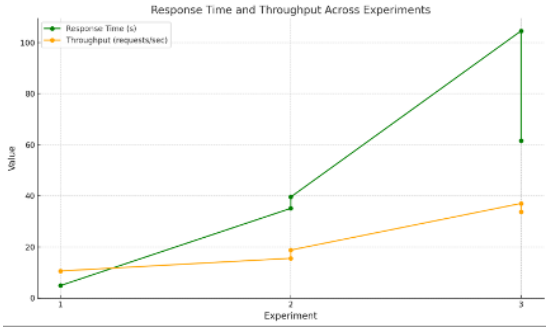
Figure 7: Simulation with No. of users set to 5000

Run 1: Simulate with No. of users set to 5000 Simulated Env Threshold score set to "5"

Run 2: Simulate with No. of users set to 5000 Simulated Env Threshold score set to "9"
The diagrams for experiments 2 and 3 show the CPU outburst and the stabilization as a new instance joins the cluster for the IDMS and an additional instance for the Java Spring Boot application. The load reduces significantly after the new instance is added with balancing handled by the clustered cache sharing session and identity data in real time with the onboarded IDMS instance.



(a) CPU Utilization Across Experiment



(b) Response Time and Throughput

The plot revealed the kind of behaviour and performance of the identity management system as well as the Java Application in solo stages of the test to understand how well it may operate in the high level of throughput even in such a constraint circumstance. We also observed increased response time with added load with a challenging scenario observed under Experiment 3 Run 1 yet the error rates remained low at 0.047. This shows how effective clustering and caching strategies, which are highly resistant, work. The throughput rose gradually with the number of users getting to a maximum of 37.03 requests per second thus demonstrating the system's ability to deal with a heavy load. These metrics highlight the flexibility of the system about the distribution of resources, and the ability to control user access in specific experimentations, such as in situations where exclusive access has been implemented.

The charts provide a clear understanding of response times, error rates, and throughput, along with a pictorial display of all criteria as concern experiments of the system.

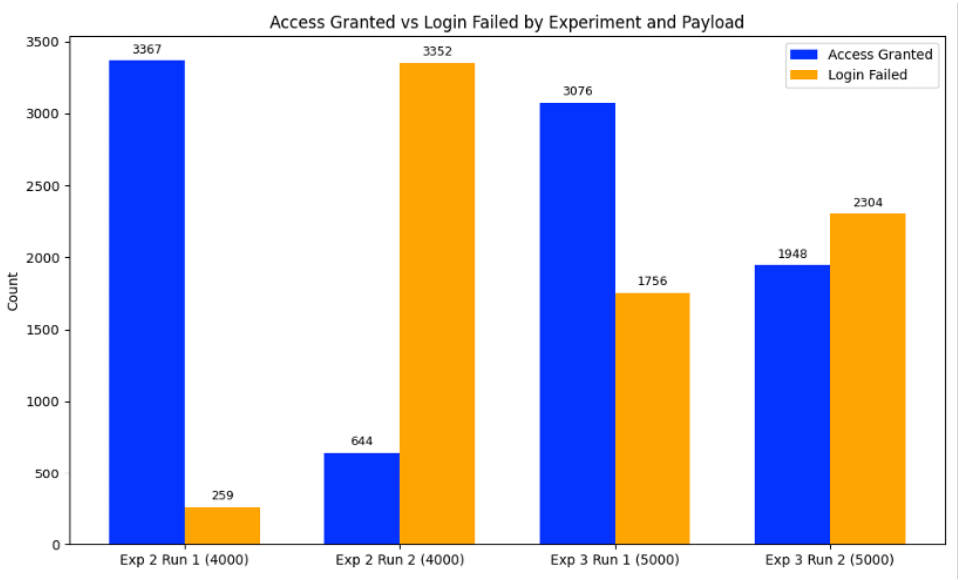


Figure 9: Access granted vs Login denied

From 9, we can see the number of user logins accepted and denied during each experimental run. It highlights the significant disparity in failed logins between Experiment 2 Run 1 and Run 2 due to the increased system threshold. Compared to Experiment 3, the gap was smaller because exclusive access was granted to about 30% of the users. In Experiment 1, almost all users gained access at the cost of system capacity since there were no scaling options or high thresholds set.

With Experiments 2 and 3, we confirm that workload capacity can be increased within seconds, as shown in the diagrams highlighting the entry points of the new instances while ensuring that only high-priority and exclusive access users gain entry to prevent a system crash.

Figure 10: Application Insight logs

Application Insights provides comprehensive real-time log information for each login request during testing giving a clear account of request responses and capturing the number of failed and successful logins.

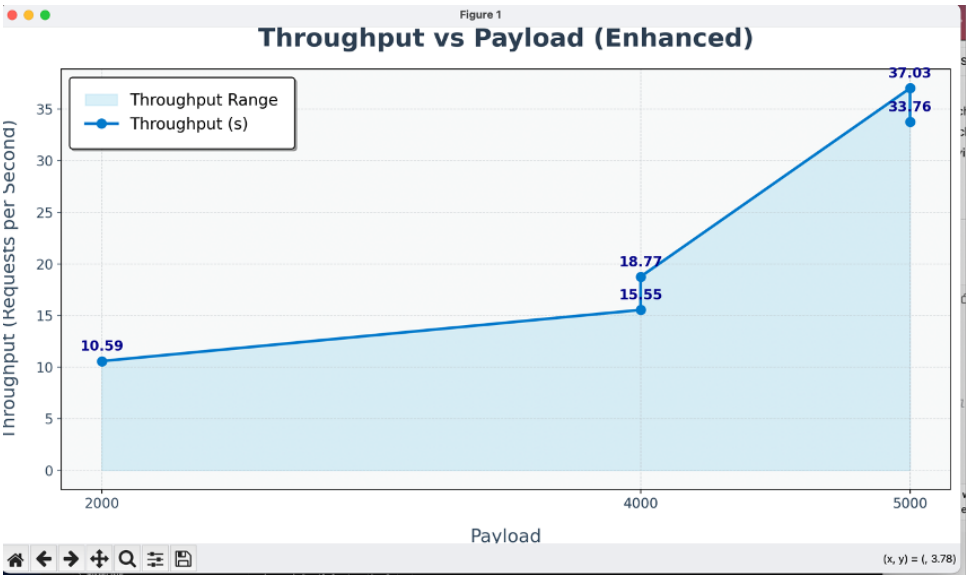


Figure 11: Throughput vs Payload

The link between throughput (requests per second) and payload size during testing is depicted in the plot. The performance climbs linearly from 10.59 requests/second to 37.03 requests/second when the payload increases from 2000 to 5000. This consistent increase demonstrates the system's scalability and capacity to effectively manage growing workloads. Throughput variability is represented by the shaded region, which demonstrates that even with larger payloads, the system operates steadily and with few fluctuations. This illustrates how resilient and well-optimized the system is to withstand high-demand situations without degrading.

6.4 Discussion

The series of experiments were conducted under varying thresholds to stress test the environment effectively, even with autoscaling enabled, without requiring frequent modifications to the deployment configuration. For better consistency in results, it was observed that testing is more effective when implemented on Azure Load Testing and distributed across multiple engines, especially for large user datasets. Best testing results were obtained from Experiment 3 Run2 as compared to Experiment 3 Run1 where we experienced a couple of Error 500 as a result of Internal Server Errors. Findings show this might be a result of a load burst implemented in a few seconds. The result of the experiments shows the success in handling our clustered Identity management system dynamically with accurate configuration in place. Onboarding and offboarding of additional IDMS instances remained seamless while user authorization was efficiently handled for priority users.
While the threshold is set at the API call level, this project does not address automating and dynamically updating it based on real-time system workload, which can be explored in future work. The aim is not to block users but to ensure that high-priority access remains unaffected during periods of high overload. This work ensures resilience and high availability of the identity management system in a containerized environment, offering full control to administrators while staying open source. Based on our findings from the clustered approach for IDMS, the experiments show that both the approach of IDMS clustering and the context-aware approach can work smoothly in synergy.

7 Conclusion and Future Work

As cloud-native technologies evolve, identity management systems (IDMS) in the cloud-native ecosystem continue to improve rapidly, supported by major providers like Microsoft, Amazon, and Okta, which cater to vast user bases and storage needs. This study, however, focuses on the benefits of a containerized approach that brings identity management closer to the application layer, or rather, closer to the application itself, enabling in-house management to maintain data sovereignty when required. This method fully implements and tests the concept of context-aware policies for user authorization, linked to specific attributes and aggregated by workload thresholds, which differs significantly from traditional RBAC concepts.
A verified technique was devised to manage user access during varying workloads using context awareness while workload scaling is going on. The integration of workload simulations, user creation, pod management, and policy enforcement illustrated the efficiency of this approach. The clear synergy between workload thresholds and context-aware decisions was established, demonstrating a robust and scalable solution for managing identity in containerized environments. Also, a single methodology is advised for operational stability and simpler system modifications, as it has been noted that employing different deployment techniques, like Helm and Kubectl together can result in discrepancies.
While this work utilized custom Java based policies to enforce context-aware authorization, future efforts could explore the use of third-party policy engines like Open Policy Agent (OPA) or Kyverno for greater interoperability with different containerized environments. These technologies can easily adapt to containerized IDMS setup, however they do require additional integration logic. Dynamic and automated context-aware decision-making would be made possible by relaying workload measurements from Prometheus to OPA via webhooks or serverless functions like Azure Functions or AWS Lambda.

References

Abayomi-Zannu, T. & Odun-Ayo, I. (2019), 'Cloud identity management – a critical analysis', *Lecture Notes in Engineering and Computer Science* .

Ahmad, W., Rasool, A., Javed, A., Baker, T. & Jalil, Z. (2021), 'Cyber security in iot-based cloud computing: A comprehensive survey', *Electronics* **10**(18).

Ahmadi, S. (2024), 'Security implications of edge computing in cloud networks', *Journal of Computer and Communications* **12**, 26–46.

Alshammari, R., Alghamdi, S. & Alotaibi, S. (2023), 'Cloud identity management and privacy strategies: A comprehensive survey', *Future Generation Computer Systems* **142**, 234–245.

Alsirhani, A., Alharbi, A. & Alhassan, I. (2020), 'Enhancing authentication techniques in cloud computing', *International Journal of Cloud Computing and Services Science* **9**(2), 130–140.

Anwar, M., Zafar, H., Mian, A. & Ali, M. (2012), 'User-centric privacy framework for cloud identity management', *Journal of Computer and System Sciences* **78**(1), 244–257.

Babakian, A. M. (2024), Resolution in Internet Identifiers: Towards Context-Awareness and Customisation, PhD thesis, University of Technology Sydney (Australia).

Benzekki, K., El Fergougui, A. & Elalaoui, A. E. B. (2018), 'A context-aware authentication system for mobile cloud computing', *Procedia Computer Science* **127**, 379–387.

Berlato, S. (2024), 'A security service for performance-aware end-to-end protection of sensitive data in cloud native applications'.
URL: *https://tesidottorato.depositolegale.it/handle/20.500.14242/161287*

Bertino, E. (2016), Data security and privacy: Concepts, approaches, and research directions, *in* '2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)', IEEE.

Bertolino, A., Auriemma, S. & Ciavotta, M. (2020), 'Leveraging containerization and orchestration for scalability in cloud-native identity management', *ACM Transactions on Software Engineering and Methodology* **30**(4), 1–30.

Chigangacha, P., Haupt, T. & Awuzie, B. (2021), 'Examining the maturity of south africa's government departments to implement the infrastructure delivery management system (idms)', *Acta Structilia* .

Dey, S., Dutta, S. & Mukherjee, A. (2019), 'Revisiting identity management in microservices architectures: Challenges and opportunities', *IEEE Access* **7**, 143870–143883.

Fehling, C., Leymann, F., Retter, R., Schupeck, W. & Arbitter, T. (2014), *Cloud computing patterns: Fundamentals to design, build, and manage cloud applications*. Available at: https://dl.acm.org (Accessed: 02/10/2024).

Ferzo, B. & Zeebaree, S. R. M. (2024), 'Distributed transactions in cloud computing: A review reliability and consistency', *Indonesian Journal of Computer Science (IJCS)* **13**(3).

Gao, W. & Zhu, Y. (2017), 'A cloud computing fault detection method based on deep learning', *Journal of Computer and Communications* **5**(12).

Khan, T., Tian, W., Zhou, G., Ilager, S., Gong, M. & Buyya, R. (2022), 'Machine learning (ml)-centric resource management in cloud computing: A review and future directions', *Journal of Network and Computer Applications* .

Lee, T. D., Lee, B. M. & Noh, W. (2018), 'Hierarchical cloud computing architecture for context-aware iot services', *IEEE Transactions on Consumer Electronics* **64**(2), 222–230.

Li, Y., Yu, M., Xu, M., Yang, J., Sha, D. & Liu, Q. (2020), Big data and cloud computing, *in* 'Manual of Digital Earth'. Available at: https://library.oapen.org (Accessed: 02/10/2024).

Matos, F., Almeida, J. & Silva, R. (2022), 'Performance modeling framework for microservices in cloud environments', *ACM Transactions on Internet Technology* **22**(2), 1–24.

Mehmood, R., Sadiq, M. & Khan, M. (2023), 'Static analysis solutions for identifying vulnerabilities in cloud-native identity management', *Journal of Information Security and Applications* **66**, 102819.

Mostafa, A. M., Rushdy, E., Medhat, R. & Hanafy, A. (2023), 'An identity management scheme for cloud computing: Review, challenges, and future directions', *Journal of Intelligent Fuzzy Systems* **45**(6), 11295–11317.

Mostafa, A. M., Said, A. S. & Shukri, N. (2021), 'Enhancing security and reliability in cloud-based identity management schemes', *Journal of Network and Computer Applications* **174**, 102894.

Oyeniran, O. C., Modupe, O. T., Otitoola, A. A., Abiona, O. O., Adewusi, A. O. & Oladapo, O. J. (2024), 'A comprehensive review of leveraging cloud-native technologies for scalability and resilience in software development', *International Journal of Science and Research Archive* **11**(2), 330–337.

Rani, R., Shukla, A. & Gupta, S. (2013), 'Identity management in cloud computing: A comprehensive overview', *International Journal of Cloud Computing and Services Science* **2**(3), 211–221.

Sahoo, S., Sahoo, S., Maiti, P., Sahoo, B. & Turuk, A. (2019), A lightweight authentication scheme for cloud-centric iot applications, *in* '2019 6th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)', IEEE.

Shahriar, H., Zhang, C., Dunn, S., Bronte, R., Sahlan, A. & Tarmissi, K. (2019), 'Mobile anti-phishing: Approaches and challenges', *Information Security Journal: A Global Perspective* **28**(6), 178–193.

Shazmeen, A., Dakić, P. & Zoltan, A. D. (2024), 'B2a, a straight-through approach from basics to advanced level: Case study for python course'.

Theodoropoulos, T., Rosa, L., Benzaid, C., Gray, P., Marin, E., Makris, A., Cordeiro, L., Diego, F., Sorokin, P., Girolamo, M. D., Barone, P., Taleb, T. & Tserpes, K. (2023), 'Security in cloud-native services: A survey', *Journal of Cybersecurity and Privacy* **3**(4), 758–793.

Truyen, E., Van Landuyt, D., Preuveneers, D., Lagaisse, B. & Joosen, W. (2019), 'A comprehensive feature comparison study of open-source container orchestration frameworks', *Applied Sciences (Switzerland)* .

Vahdat-Nejad, H., Izadpanah, S. & Ostadi-Eilaki, S. (2019), 'Context-aware cloud-based systems: design aspects', *Cluster Computing* **22**, 11601–11617.

Wang, Y., Kadiyala, H. & Rubin, J. (2021), 'Promises and challenges of microservices: an exploratory study', *Empirical Software Engineering* .