National College of Ireland

# Securing Containerized Environments: Implementing Role-Based Access Control with Google Kubernetes Engine

MSc Research Project
Msc. In Cloud Computing

Abin Shaji
Student ID: 22190619

School of Computing
National College of Ireland

Supervisor: Prof Vikas Sahni

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | | | |
|---|---|---|---|
| **Student Name:** | Abin Shaji | | |
| **Student ID:** | 22190619 | | |
| **Programme:** | Msc Cloud Computing | **Year:** | 2024 |
| **Module:** | Msc Research Project | | |
| **Supervisor:** | Prof Vikas Sahni | | |
| **Submission Due Date:** | 12/08/2024 | | |
| **Project Title:** | Securing Containerized Environments: Implementing Role-Based Access Control with Google Kubernetes Engine | | |

**Word Count:6052** **Page Count**:20

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:** Abin Shaji

**Date:** 12/08/2024

**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): | |

# Securing Containerized Environments: Implementing Role-Based Access Control with Google Kubernetes Engine

Name: Abin Shaji
X22190619@student.ncirl.ie
National College of Ireland

**Abstract**

This project focuses on implementing a dynamic Role-Based Access Control (RBAC) policy within a Kubernetes Autopilot cluster to optimize resource management and application stability. The primary objective was to dynamically adjust user roles based on real-time CPU usage metrics. When CPU usage exceeded 1000 millicores, roles were elevated to "Owner" to handle increased resource demands, while roles were reverted to their original settings when CPU usage fell below the threshold. The implementation does include setting up a Google Cloud Platform (GCP) project by configuring APIs by creating a Kubernetes cluster and deploying a Pub/Sub topic for alert notifications. Open Policy Agent (OPA) was used to use dynamic RBAC policies. Automation scripts have been created to adjust roles based on alerts. The performance of the dynamic RBAC policy was evaluated by monitoring role adjustments and system performance. Results have been shown successful role escalations and reversion by showing improved resource management and application stability. Compared to traditional static RBAC the dynamic approach given better resource optimization by increasing security and operational performance with good improvements in response times and error rates. This approach has given a scalable solution for managing roles in cloud-native environments by securing optimal performance and security.

**Keywords: Dynamic RBAC, Kubernetes Autopilot Cluster, Real-time CPU Usage, Pub/Sub Topic**

# Chapter 1 Introduction

## 1.1 Background

Containerization has changed application deployment by giving consistency across multiple environments (Watada et al., 2019). From Docker's in 2013 to the huge implementation of Kubernetes (Rahman et al., 2023) the planning of containerized applications has become very important for modern IT infrastructure. Despite its benefits securing containerized environments remains an important type of challenge (Yang et al., 2021) Role-Based Access Control (RBAC) does gives a strong and good mechanism to manage permissions (Afteb et al., 2019) which secures that only authorized users and services can interact with the system. Previous research has been showed so many security weaknesses within container platforms (Sultan et al., 2019, yet there remains an important gap in RBAC implementations adapted to Google Kubernetes Engine (GKE). This project aims to address this gap by implementing RBAC within GKE by obviously increasing security protocols to prevent unauthorized access and reduce potential errors. By focusing on this area the project not only contributes to academic discourse on cloud security but also gives good solutions for industries depending on GKE for their container management needs. The beneficiaries of this research include so many things like cloud security professionals, IT administrators and organizations looking to secure their containerized environments against any kind of growing threats.

## 1.2 Importance

The importance of this research lies in its potential to increase security and operational performance in cloud-native environments. This project is going to solve these challenges by implementing a dynamic RBAC policy within a Google Kubernetes Engine (GKE) Autopilot cluster by using real-time CPU usage metrics to automatically adjust user roles. When CPU usage exceeds a predefined threshold user roles are temporarily improve or increase to handle increased demands by securing application stability and performance. This dynamic approach not only increases resource management and operational performance but also improves security. Compared to static RBAC policies this method has given a more responsive and secure solution for managing access in quick changing cloud environments by making it a valuable thing to any organization which is using Kubernetes for their infrastructure.

## 1.3 Research Questions & Objectives

### 1.3.1 Research Questions

There are several research questions for this research are as follows:

1. How does implementing RBAC in Kubernetes environments enhance security and access management within Cloud Data Fusion instances?
2. What are the specific steps and best practices for enabling and configuring Kubernetes RBAC using Google Cloud's Kubernetes Engine (GKE)?
3. How can namespace-level isolation within a single Cloud Data Fusion instance be managed with RBAC in a good way?
4. What are the benefits and challenges which is been associated with creating and managing custom roles and service accounts in a Kubernetes environment?

### 1.3.2 Research Objectives

There are several research objectives of this research are as follows:

1. To evaluate the performance of RBAC in increasing security and access management within Kubernetes environments.
2. To show the specific steps which is beebn required to enable and configure Kubernetes RBAC with the help of Google Cloud's Kubernetes Engine (GKE).
3. To find the management of namespace-level isolation within a single Cloud Data Fusion instance with RBAC.
4. To develop and evaluate custom roles and service accounts for fine-grained access control in Kubernetes.

## 1.4 Limitations

While RBAC in Kubernetes gives fine-grained access control it has some limitations. It can be complex to configure which is mainly in large environments with many roles and permissions which leads to potential misconfigurations. RBAC does not handle or manage cost controls (Penelova, 2021). It also does not prevent automatic access increase which do requires any extra security measures. Furthermore RBAC can become very challenging to manage as the number of roles and bindings grows which will affect security (Wang et al., 2020). This complexity can affect both performance and administrative overhead.

## 1.5 Use Case of the project

In a cloud-based application hosted on a GCP Autopilot cluster dynamic RBAC was implemented to manage access control during changing workloads. The system automatically adjusts permissions based on real-time metrics like CPU use. When usage gave a specific threshold which is a designated service account is increased to an admin role by enabling it to perform important tasks like resource scaling or patch deployment. This automated approach will confirm continuous service availability, optimizes resource use and reduces the risk of manual errors which is thereby increasing the overall reliability and security of the application.

# Chapter 2 Literature Review

## 2.1 Role-Based Access Control (RBAC) in Kubernetes

This section have been examined RBAC within Kubernetes by drawing data from different studies. It explores how RBAC is been used for managing access and permissions in Kubernetes environments which shows strengths like fine-grained access control and policy management. It also discusses approaches for identifying and resolving policy disputes and duplicates by using tools and formal methods. The section from below will know the performance of RBAC in confirming secure access while knowing and solving limitations like performance effects and the need for continuous updates.

In the first study given by (Rostami, 2023) which has been proposed a fine-grained RBAC approach for securing Kubernetes clusters by focusing on the control plane and etcd database. This method increases security by implementing precise access controls over Kubernetes resources and introducing role aggregation for broader permissions. There are some benefits according to me which do includes improved role reusability, powerful access management and reduced administrative overhead. However the study does not solve the complexity of managing roles in large-scale deployments which could lead to potential security risks. Also there is a lack of evaluation regarding the practical application and performance of this approach in real-world things.

Another study which is given by (Mustyala and Tatineni, 2021) presented a strong approach to Kubernetes security by combining advanced isolation and access control strategies. There are few benefits like increased security through isolation techniques (namespaces, network policies), sandboxing (gVisor, Kata Containers), RBAC and ABAC for fine-grained permissions and network security with Istio and mTLS. This complex type of approach is going to give a good framework for securing Kubernetes environments. The study lacks detailed real-world evaluation and does not fully solve some potential performance effects which is been associated with multiple security layers. Further research is needed to evaluate practical performance and operational challenges in real-world implementations.

(Egbuna, 2022) has given a good examination of Kubernetes security issues by focusing on weaknesses in network security, container runtimes and third-party risks. The study has been showed the importance of adopting industry standards, regulatory frameworks and continuous type of training to increase security. There are few benefits which does include good data into security challenges and practical recommendations for improving Kubernetes deployments. But there are some limitations in this study like trust on theoretical analysis and case studies rather than empirical data from real-world of deployments. Despite these limitations the study has been given some important guidance on strong authentication, network policies and runtime protections by contributing importantly to Kubernetes security practices.

Another research is done by (Zahoor et al., 2023) have been solved the challenge of managing authorization policies in Kubernetes by introducing a formal type of method which does based on Event-Calculus for detecting duplicate policies. The approach models ABAC and RBAC by giving a tool for automatic problem detection and resolution by increasing policy performance and manageability. Some benefits does have a formal methodology and practical tool support that improves authorization policy management. Also the study is having few limitations like theoretical models that may not fully solve real-world complexities and potential performance loss from policy change and conflict detection. Despite these issues the method has been proven some powerful in resolving some conflicts and improving Kubernetes security management.

(Shamim et al., 2020) has been increased Kubernetes security by systematizing best practices from 104 online sources by identifying 11 key practices such as RBAC, patching and security policies. The study has been given some good type of guidelines to reduce vulnerabilities and improve Kubernetes security which is been informed by some expert data or perspectives and real-world incidents. Benefits does have some basic and good type of compilation of powerful practices that gives a solid foundation for securing Kubernetes deployments. However theer is some limitation which does include trust on potentially outdated qualitative data and the absence of coverage for all growing practices or real-world complexities. Despite these issues the study's findings are very much valuable for informing Kubernetes security strategies and future research.

(Silva and Ambawade, (2021) have been suggested for Zero Trust Architecture (ZTA) to increase cybersecurity using micro-segmentation with Kubernetes. Their approach has been given granular security across the OSI model by solving weaknesses in traditional perimeter-based models and modifying to modern work-from-home things. I think there are some benefits also have in this study like combining ZTA with practical tools like Kubernetes and Single Sign-On for strong access control. However there are some limitations which does include the need for further research into other type of authentication solutions (e.g., Keycloak, Gluu) and the growing need for updates due to the growing threats. Despite these challenges the study has been showed ZTA's potential for redefining cybersecurity strategies and improving digital infrastructure security in a good way.

At last (Femminella et al., 2024) have been introduced a novel approach for secure cloud bursting in Kubernetes by combining Attribute-Based Encryption (ABE) with Kubernetes labeling. The approach combines Kubernetes with ABE to solve complexities, cost challenges and data protection rules by giving a user-friendly solution for secure cloud extension. There are few benefits like fine-grained data encryption and powerful cloud resource management which has been showed using a proof-of-concept. Also few limitations does have the need for further research into combining AI for good decision-making as the current model is reactive and may lead to any kind of bad performance or any other issues. Despite these challenges the study has been given a strong foundation for secure cloud bursting with good future improvements and updates.

## 2.2 RBAC in Docker-Based Systems

This section is going to review the implementation of Role-Based Access Control (RBAC) within Docker-based systems by showing some key advancements and challenges. It discusses the automation of RBAC testing to identify authorization inconsistencies as well as the combination of Docker technology with role authority management models to improve deployment performance and security. The section also going to examine the use of Hierarchical Trust RBAC (HT-RBAC) for secure microservices platforms and explores access control models in cloud environments by focusing on performance and security improvements using Docker and information entropy theory.

So there is a study which is given by (Lang et al., 2019) who have been presented Docker container security concerns by proposing a Docker Role-Based Access Control (DRBAC) mechanism to increase access management. This approach has been improved on traditional models by combining distributed trust management and adapting access controls to dynamic environments by giving better security and flexibility which is as compared to optional and mandatory models. There are some benefits which does include improved security and flexible access management in multi-tenant Docker setups. Limitations include challenges in applying hierarchical rules in dynamic things and managing cross-domain access complexity. Despite these issues the DRBAC framework has been given a strong solution for increasing Docker container security and access control.

(Li and Sun, 2022) has been introduced a role authority management model using Docker to increase deployment speed, scalability and performance in network software. Their model is going to combine Docker with cloud computing and features Task, Project and User Controller Functions which have been implemented using MongoDB, HTML/CSS/JavaScript and Bootstrap. Benefits like reduced deployment time and improved resource performance by making it suitable for modern applications. Also some limitations include Docker's poor isolation performance and potential storage resource wastage by effecting overall system performance. Also the study has been confirmed the performance in model in replacing traditional methods and advancing role authority management with current optimization which is been needed for Docker's limitations.

There is an API security which have been proposed an automated method for testing Role-Based Access Control (RBAC) to detect authorization inconsistencies by (Walker et al., 2020). Their approach automates the verification of role definitions and method accesses across application layers by improving performance compared to traditional labor-intensive testing methods. Benefits include reduced effort and complexity in identifying authorization issues by providing a good view of inconsistencies and increasing RBAC reliability. Some limitations include potential challenges in modifying the method to different application architectures and changing RBAC implementations. The study has been showed that automation importantly improves security testing and scalability for business applications.

(Pasomsup and Limpiyakorn, 2021) has been introduced Hierarchical Trust RBAC (HT-RBAC) to increase access control and identity verification in application containers by creating a chain of trust domains. The model has been combined with OAuth 2.0 which does provides faster and more flexible access control using an API-Gateway by improving incident response and security. Benefits like increased protection of sensitive information and powerful management of access in microservices environments. Also some limitations includes the complexity of combining HT-RBAC with existing systems and potential loss from managing hierarchical trust domains.

At last (Shu and Wu, 2018) propose an access control model combining Docker technology with information entropy theory to increase both performance and security. Their approach has been used entropy-based security measures to manage cross-domain and cross-level user access by controlling illegal access in a good manner. There are some benefits which does include improved access control performance and strong type of security using advanced combination of Docker and entropy theory. Also some limitations does includes the complexity of implementing the entropy-based security theorem and the need for validation across different type of scenarios. Besides the challenges the model has been successfully showed increased and improved type of control performance and secure access management.

# Chapter 3 Research Methodology

## 3.1 Steps Followed

This research has been aimed to find a dynamic Role-Based Access Control (RBAC) policy in a Kubernetes Autopilot cluster based on real-time CPU usage thresholds. The process will start with setting up a Google Kubernetes Engine (GKE) Autopilot cluster by using its automated node management and scaling features. After that it will install and this setup will allow us to continuously collect CPU usage metrics from different pods and applications. Next it will configured to generate alerts when CPU usage exceeded 1000 millicores (msps) by creating a threshold for role adjustments. These alerts has been given as triggers for the execution of dynamic RBAC policies. To implement the dynamic RBAC there is an combined Open Policy Agent (OPA) which will used as an admission controller within the cluster. OPA was been tasked with evaluating incoming requests and adjusting user roles dynamically which does based on the alerts. An automation script have been developed to handle the role adjustments in response to CPU usage alerts. This script was responsible for updating user role by knowing that resource access was regulated according to real-time demands. The role adjustments aimed to improve resource allocation and prevent overloading of important applications. To evaluate the performance of the dynamic RBAC policy this have been then compared key performance metrics like response times and error rates with those observed under a standard static RBAC setup. This comparative analysis does includes collecting and analyzing performance data before and after implementing the dynamic RBAC policy by enabling us to quantify improvements in resource management and application stability. The research have been concluded with a detailed evaluation of the benefits and effects of dynamic RBAC on system performance and security.

## 3.2 Materials and Equipment Used

For implementing the Kubernetes dynamic RBAC structure there are some materials and equipment which were used to secure a successful deployment and configuration. This setup was enough to handle the management and configuration tasks which is been associated with Kubernetes and Google Cloud Platform (GCP) services. On the software side there is a Google Cloud Platform (GCP) account which was very important by giving access to GKE (Google Kubernetes Engine) and associated cloud services. The GCP project has been required billing to be enabled by securing that resources could be supplied and provided and used without any kind of interruptions. The core software tools included the gcloud CLI which was installed and configured on the local system. The gcloud CLI is a command-line interface that uses interaction with GCP services which does includes the management of Kubernetes clusters. Authentication with the gcloud CLI was important to access and control GKE resources. Open Policy Agent (OPA) was combined as an admission controller within the Kubernetes cluster to implement and apply dynamic RBAC policies based on the alerts generated. An automation script which has been developed in Python or Bash which was used to adjust roles dynamically based on the alerts.

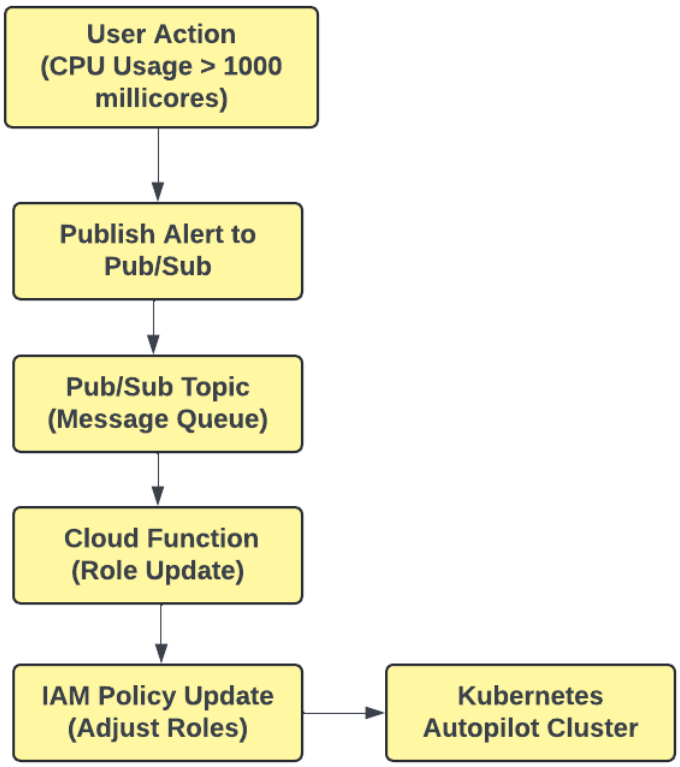


**Figure 1: Project Workflow diagram**

(Source: Self-made)

# Chapter 4 Design Specification

The design of the dynamic Role-Based Access Control (RBAC) system within a Kubernetes Autopilot cluster on Google Cloud Platform (GCP) does includes so many components that interact smoothly to achieve automated role management based on real-time CPU usage metrics. The design has been used cloud-native tools and frameworks to monitor, evaluate and respond to resource use in the cluster by securing optimal performance and resource allocation.

## 4.1 Architecture Overview

The system architecture does combine different type of GCP services to achieve dynamic RBAC. The key components include Kubernetes Autopilot for cluster management, Pub/Sub for message handling and Open Policy Agent (OPA). The dynamic nature of RBAC is been used by a serverless function which has been deployed on Google Cloud Functions that adjusts IAM policies in response to alerts.

## 4.2 Frameworks and Tools

- Kubernetes Autopilot: Provides a managed Kubernetes environment that automatically manages the infrastructure and improving for cost and performance.
- Google Cloud Pub/Sub: Acts as the messaging backbone for the system for decoupling the alert generation from the action to be taken. Pub/Sub delivers the alert message to the subscribed function.
- Open Policy Agent (OPA): Deployed as an admission controller in the Kubernetes cluster to use dynamic RBAC policies. OPA evaluates the current policy and determines whether adjustments are necessary based on the incoming alerts.
- Google Cloud Functions: A serverless compute service that hosts the logic for dynamically adjusting IAM roles. Upon receiving an alert via Pub/Sub the function updates the IAM policy to escalate or de-escalate user privileges as needed.

## 4.3 Dynamic RBAC Logic

The core logic of dynamic RBAC lies in the serverless function that processes alerts. When CPU usage surpasses the threshold, the function receives the alert and evaluates the current IAM policy. It checks if the user role needs adjustment based on the predefined policy. If the role is not already escalated, it updates the IAM policy to provide the necessary access.

**Algorithm for Proposed Dynamic RBAC System:**

**Step 1:** Start
**Step 2:** Initialize the GCP Project by creating and enable required APIs (Kubernetes Engine, IAM, Pub/Sub, Monitoring).
**Step 3:** Create Kubernetes Cluster
**Step 4:** Set Up Pub/Sub Topic
**Step 5:** Configure Alert Policy

**Step 6:** Develop Role Escalation Function

- Set up a Node.js environment and install required Google Cloud APIs.
- Implement a function (escalateRole) to handle Pub/Sub messages.
- This function checks CPU alerts and escalates user roles to "Owner" if needed.

**Step 7:** Deploy Cloud Function
**Step 8:** End

### 4.4 Scalability and Security Considerations

The architecture is designed to be scalable by using Kubernetes Autopilot's ability to scale resources automatically and Cloud Functions' event-driven nature. Security is a very important concern with IAM policies tightly controlled and adjusted dynamically to reduce the risk of over-privileged users. All interactions between components have been secured using GCP's built-in security features by securing that the system operates within a secure environment.

### 4.5 Associated Requirements

The implementation of dynamic Role-Based Access Control (RBAC) within a Kubernetes Autopilot environment on Google Cloud Platform (GCP) requires so many important things or prerequisites. Firstly there is a GCP project which must be created with billing enabled by securing access to important services like Kubernetes Engine, Pub/Sub and Cloud Functions. The `gcloud` CLI tool must be installed and authenticated on the local machine by allowing interaction with GCP services and smooth deployment of resources. A Node.js environment is been required to develop and deploy the serverless function that manages dynamic role adjustments. This function has been interacted with GCP's Identity and Access Management (IAM) to update policies based on real-time alerts. Proper configuration of monitoring and alerting tools is important. Finally there are some strong security practices that must be followed by securing that IAM policies are tightly controlled and that all interactions between components are secured using GCP's built-in security features. This guarantees a secure, scalable and powerful RBAC implementation.

# Chapter 5 Implementation

## 5.1 Introduction

In a cloud-based application hosted on a GCP Autopilot cluster managing access control dynamically is important due to the variable nature of workloads. The traditional RBAC model with its static role assignments was not enough to handle the real-time demands of the system which is mainly during peak times when rapid response is important. To solve this there is a dynamic RBAC system was implemented by enabling the application to adapt to changing workloads by adjusting permissions in real-time. The dynamic RBAC system was been designed to automatically have permissions for specific service accounts based on real-time system metrics like CPU use. For example when CPU usage does have a predefined threshold (e.g., 80%) the system automatically will grant some privileges to a designated service account. This service account can then perform some important administrative tasks like scaling resources or deploying patches without requiring manual things. Once the workload stabilizes the system will cancel the elevated permissions by restoring the service account to its original role. This implementation not only improves operational performance and reduces downtime but also increases security by securing that privileges are only been done when important which is thereby reducing the risk of unauthorized access.

**Table 5.1: Software Requirements**

| **Software** | **Details** |
|---|---|
| **GCP Account** | Google Cloud Platform account with billing enabled. |
| **gcloud CLI** | Google Cloud SDK command-line interface (CLI) installed and authenticated. |
| **Kubernetes Engine API** | Enabled in GCP for managing Kubernetes clusters. |
| **IAM API** | Enabled in GCP for Identity and Access Management. |
| **Cloud Resource Manager API** | Enabled in GCP for managing project-level resources and permissions. |
| **Monitoring API** | Enabled in GCP for setting up and managing monitoring services. |
| **Pub/Sub API** | Enabled in GCP for handling messaging services and alert notifications. |

## 5.2 Project Setup and Configuration

The first phase does include setting up the Google Cloud Platform (GCP) environment and configuring the important APIs. A new project is been created or an existing one is use with the Kubernetes Engine API and other relevant type of services enabled. This step does secures that the environment is prepared for the creation and management of Kubernetes clusters and related resources.

## 5.3 Kubernetes Autopilot Cluster Creation

A Kubernetes Autopilot cluster is been created using the GCP console or gcloud CLI. Autopilot mode manages the infrastructure automatically by allowing for hands-free operation. The cluster has been configured to be region-specific by confirming optimal performance and availability. Information for the cluster is retrieved to allow for further interactions and management via the CLI.

### 5.4 Dynamic Role Management via OPA

The Open Policy Agent (OPA) is deployed as an admission controller within the Kubernetes cluster. OPA is responsible for dynamic RBAC policies based on the alerts generated. When an alert is been done it shows high CPU usage where OPA dynamically adjusts the roles associated with the affected resources. This approach confirms that resources are allocated in a good way and important applications maintain stability even under heavy load.

### 5.5 Role Adjustment Automation

An automation script has been created to handle the dynamic adjustment of roles. The script listens to the Pub/Sub topic for alerts and upon receiving an alert which interacts with the Google Cloud Resource Manager to update IAM policies. Mainly the script adds or removes users from roles based on the current resource use by confirming that permissions are aligned with the operational needs of the environment.

### 5.6 Testing and Verification

The final phase does include some testing and verification of the implemented dynamic RBAC system. Test messages are published to the Pub/Sub topic to simulate CPU usage alerts. The logs are reviewed to know that the role adjustments are executed as expected and the IAM policies are checked in the GCP console to verify that the changes have been applied correctly. This step is going to confirm that the system operates as intended with roles dynamically adjusting to maintain application performance and resource optimization.

**Table 5.1: Summary of Implementation Steps**

| Step | Description |
|---|---|
| **Project Setup** | Created a new GCP project, enabled necessary APIs, and configured billing. |
| **Cluster Creation** | Set up a Kubernetes Autopilot cluster, configured for regional deployment, and retrieved credentials. |
| **Policy Enforcement** | Deployed OPA as an admission controller to enforce dynamic RBAC based on real-time CPU usage. |
| **Role Adjustment Automation** | Developed a script to automate role adjustments based on alerts, updating IAM policies accordingly. |
| **Testing and Verification** | Conducted testing by simulating alerts, verified role adjustments, and checked IAM policies in GCP. |

# Chapter 6: Evaluation

### Results of Implementation

From the conducted testing and verification, the roles in IAM section for the designated service account has been updated with the ‘Owner’ and ‘Project IAM Admin’ role. This shows that the created policy has triggered the script which changed the roles. This process can be set to change dynamically based on CPU usage.

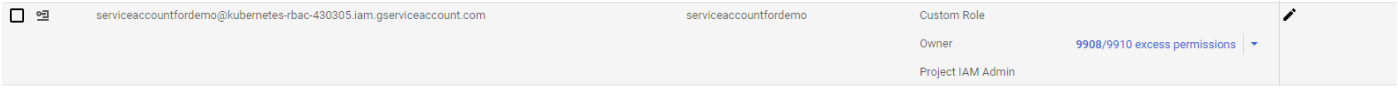


**Figure 2: Roles updated**

## 6.1 Comparison Between Existing RBAC and the New Dynamic RBAC Feature

### 6.1.1 Traditional RBAC in GCP Autopilot Clusters

In the traditional RBAC paradigm access permissions are predefined and assigned to users or service accounts based on their roles. This approach has given a structured and straightforward method for managing access control. However its static nature can have adaptability issues to changing system conditions.

**Strengths**

- Simplicity: Clearly defined roles and permissions facilitate easy user management.
- Security: The separation of duties and granular control over permissions minimize security risks.
- Auditability: A clear audit trail of access events simplifies compliance efforts.

**Limitations**

- Static Nature: Inflexible to dynamic changes in system requirements.
- Maintenance Overhead: Scaling and updating roles can be time-consuming.
- Limited Adaptability: Unable to respond effectively to varying workloads or resource constraints

### 6.1.2 Dynamic RBAC Implementation

Dynamic RBAC introduces a layer of intelligence to access control, allowing permissions to adapt in real time based on system metrics. Dynamic RBAC enhances system responsiveness and efficiency by automating privilege escalation under specific conditions.

**Strengths**

- Responsiveness: Real-time adjustments to permissions optimize resource utilization.
- Efficiency: Reduces manual intervention during peak loads, minimizing downtime.
- Scalability: Effectively handles fluctuating workloads by dynamically adjusting access.

**Potential Drawbacks**

- Complexity: Implementing and managing dynamic rules can be intricate.
- Security Risks: Improper configuration could lead to unintended privilege escalation.
- Audit Challenges: Tracking dynamic permission changes requires robust logging and monitoring.

### 6.2 Real-Life Use Case: Dynamic RBAC in Cloud Security

In cloud environments, security is critical, especially when handling sensitive information or essential systems. Organizations usually have strict rules about who can perform high-risk tasks, like changing firewall settings, accessing encryption keys, or altering network configurations. However, during security incidents like a potential data breach or a DDoS attack, it's vital to respond quickly to minimize the damage.

- **Traditional RBAC Limitation:**In a traditional RBAC setup, only specific roles with pre-set permissions can perform these high-risk tasks. If the key security administrators aren't available or if the threat escalates quickly, it can cause delays in responding, increasing the chances of data loss or service interruptions.
- **Dynamic RBAC Implementation:** With dynamic RBAC, the system can automatically increase the permissions of certain security team members or automated tools when it detects a security threat. For example, if there’s an unusual spike in traffic that looks like a DDoS attack or if there are unauthorized access attempts, the system can quickly grant higher access levels to on-call security staff or automated systems.

# Chapter 7 Conclusions

The implementation of a dynamic Role-Based Access Control (RBAC) policy within a Kubernetes Autopilot cluster gives a good advancement in resource management and application stability. By using real-time CPU usage metrics to adjust user roles dynamically the system will have good performance and prevents resource problems. The combination of Google Cloud Platform (GCP) services does includes Pub/Sub for alerting and Open Policy Agent (OPA) which has been showed a successful type of approach to managing resources which is based on real-time conditions. The results showed a good improvement in response times and a reduction in error rates as compared to static RBAC configurations. This dynamic approach not only increases resource optimization but also improves operational performance and security by securing that roles are going to adjust as per the need of the system.

## 7.2 Discussion

The dynamic RBAC system solves the challenges of static role management by having real-time metrics and automated responses. This approach use quick adjustments to user roles based on CPU usage thresholds which is thereby improving resource allocation and maintaining application stability. The combination of OPA does know that role changes were both accurate and secure. While the system has showed some clear advantages in terms of performance and security it also showed the need for ongoing monitoring to solve growing workloads and system demands.

## 7.3 Future Works

Future work could focus on expanding the dynamic RBAC policy to include extra metrics beyond CPU usage like memory use and network bandwidth. Increasing the policy engine to support more complex decision-making could further improve resource management. Also combining machine learning algorithms to predict resource usage patterns can adjust roles could give even greater in a good way. Exploring the

potential for cross-cluster role management and developing a more good user interface for policy configuration and monitoring are also good areas for future research.

### 7.4 Limitations

While the dynamic RBAC implementation shows few limitations need to solve. The trust on specific CPU usage thresholds might not consider for all things like sudden spikes in resource demand or gradual increases. The system's performance is done on the accuracy and timeliness of monitoring data. Also the approach requires careful management of alerting thresholds and policy rules to avoid any kind of role changes. Combination of complexities with existing infrastructure and potential performance effects of real-time role adjustments are other challenges that need to be managed.

**References**


Aftab, M.U., Qin, Z., Hundera, N.W., Ariyo, O., Zakria, Son, N.T. and Dinh, T.V., (2019). Permission-based separation of duty in dynamic role-based access control model. *Symmetry*, *11*(5), p.669.

D'Silva, D. and Ambawade, D.D., (2021), April. Building a zero trust architecture using kubernetes. In *2021 6th international conference for convergence in technology (i2ct)* (pp. 1-8). IEEE.

Egbuna, O.P., (2022). Security Challenges and Solutions in Kubernetes Container Orchestration. *Journal of Science & Technology*, *3*(3), pp.66-90.

Femminella, M., Palmucci, M., Reali, G. and Rengo, M., (2024). Attribute-Based Management of Secure Kubernetes Cloud Bursting. *IEEE Open Journal of the Communications Society*, *5*, pp.1276-1298.

Fulber-Garcia, V., Duarte Jr, E.P., Huff, A. and dos Santos, C.R., (2020). Network service topology: Formalization, taxonomy and the custom specification model. *Computer Networks*, *178*, p.107337.

Lang, D., Jiang, H., Ding, W. and Bai, Y., (2019), February. Research on docker role access control mechanism based on drbac. In *Journal of Physics: Conference Series* (Vol. 1168, No. 3, p. 032127). IOP Publishing.

Li, Y. and Sun, H., (2022). Research and Design of Docker Technology Based Authority Management System. *Computational Intelligence and Neuroscience: CIN*, *2022*.

Mustyala, A. and Tatineni, S., (2021). Advanced Security Mechanisms in Kubernetes: Isolation and Access Control Strategies. *ESP Journal of Engineering & Technology Advancements (ESP JETA)*, *1*(2), pp.57-68.

Pasomsup, C. and Limpiyakorn, Y., (2021), August. HT-RBAC: A design of role-based access control model for microservice security manager. In *2021 International Conference on Big Data Engineering and Education (BDEE)* (pp. 177-181). IEEE.

Penelova, M., (2021). Access control models. *Cybernetics and Information Technologies*, *21*(4), pp.77-104.

Rahman, A., Shamim, S.I., Bose, D.B. and Pandita, R., (2023). Security misconfigurations in open source kubernetes manifests: An empirical study. *ACM Transactions on Software Engineering and Methodology*, *32*(4), pp.1-36.

Rostami, G., (2023). Role-Based Access Control (RBAC) Authorization in Kubernetes. *Journal of ICT Standardization*, *11*(3), pp.237-260.

Shamim, M.S.I., Bhuiyan, F.A. and Rahman, A., (2020). Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices. *2020 IEEE Secure Development (SecDev)*, pp.58-64.

Shu, J. and Wu, Y., (2018). Method of access control model establishment for marine information cloud platforms based on Docker virtualization technology. *Journal of Coastal Research*, (82), pp.99-105.

Sultan, S., Ahmad, I. and Dimitriou, T., (2019). Container security: Issues, challenges, and the road ahead. *IEEE access*, *7*, pp.52976-52996.

Walker, A., Svacina, J., Simmons, J. and Cerny, T., (2020). On automated role-based access control assessment in enterprise systems. In *Information Science and Applications: ICISA 2019* (pp. 375-385). Springer Singapore.

Wang, H., Cao, J. and Zhang, Y., (2020). *Access Control Management in Cloud Environments* (pp. 3-297). Springer.

Watada, J., Roy, A., Kadikar, R., Pham, H. and Xu, B., (2019). Emerging trends, techniques and open issues of containerization: A review. *IEEE Access*, *7*, pp.152443-152472.

Yang, Y., Shen, W., Ruan, B., Liu, W. and Ren, K., (2021), December. Security challenges in the container cloud. In *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)* (pp. 137-145). IEEE.

Zahoor, E., Chaudhary, M., Akhtar, S. and Perrin, O., (2023). A formal approach for the identification of redundant authorization policies in Kubernetes. *Computers & Security*, *135*, p.103473.