International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 07, July - 2025

Secure Access Key Management

A Restful Approach with Role-Based Control and SSO

Deepsingh Chhabda
Independent Researcher
Princeton, New Jersey, USA

Mehaerkaur Chhabda
Independent Researcher
Princeton, New Jersey, USA

***Abstract*—In modern cloud-native environments, managing and securing access and secret keys is critical to ensuring organizational security and regulatory compliance. This paper presents the design and implementation of a RESTful API-based web application that automates the periodic rotation of AWS IAM access and secret keys. Our solution addresses key security challenges by integrating several core AWS services and best practices into a unified, automated pipeline. Leveraging AWS boto3 library to interact with AWS resources, such as IAM, the system programmatically generates new access and secret keys, stores them securely in a credentials vault, and retires older credentials without manual intervention. A robust user role-based authentication mechanism ensures that only authorized users can trigger or access rotation workflows. The architecture enhances the developer experience by offering a seamless, secure method to maintain the keys, while reducing the risk of unauthorized access and potential key compromise [1] [2]. This paper also discusses implementation details, performance considerations, and security evaluations to validate the efficacy and scalability of the proposed solution in enterprise environments while adhering to AWS-recommended best practices [3] [4] .**

***Keywords*— AWS IAM, CLI boto3, role-based access control, SSO, AWS best practices, REST API**

I. INTRODUCTION

In modern DevOps and cloud-native infrastructures, securely managing AWS credentials at scale remains a critical challenge. Static IAM access keys pose a significant security risk when not rotated regularly or stored insecurely. Mismanagement can lead to key leakage, unauthorized access, or service downtime, especially in multi-user or CI/CD-driven environments.

To address these issues, this paper proposes a REST API-based automation framework for periodic IAM access key rotation, integrated with SSO and role-based authorization. By combining native AWS services (IAM, CloudWatch) with a modular, Flask-based backend, this system enables minimal-downtime key rotation, centralized credential management, and auditable access.

Our solution aims to minimize manual intervention, increase compliance with cloud security standards, and simplify credential provisioning for teams in fast-paced engineering environments. The paper details the system's architecture, workflow, security design, and provides real-world performance metrics after deploying in a mid-sized AWS environment.

II. METHODOLOGY

This section outlines the design and implementation methodology of an automated system for managing the lifecycle of AWS IAM credentials. The proposed system leverages a RESTful micro-service architecture to ensure secure, scalable, and auditable credential management with minimal user intervention. The methodology is driven by five core objectives and consists of two primary workflow components: autonomous key rotation and authorized access to the system.

A. Objectives

The methodology is guided by the following objectives

- Minimal User Intervention: Ensure credentials are stored and managed securely with limited user involvement, thereby reducing operational overhead and human error.
- Automated Credential Rotation: Enhance security by implementing automatic rotation of IAM credentials at predefined intervals, thereby minimizing the exposure window of potentially compromised credentials.
- Tool Standardization: Utilize standardized libraries and cloud-native technologies to reduce reliance on third-party tools and simplify system dependencies.
- Centralized and Auditable Access: Provide a centralized system with an intuitive web interface that supports credential auditing and secure access control.
- Cloud-Agnostic Extensibility: Design the solution to be easily adaptable for multi-cloud environments, enabling future expansion to other cloud providers.

A. Components

The workflow of the proposed system is comprised of two core components: Autonomous Key Rotation and Authorized User Services that collectively facilitate the secure and automated management of AWS IAM credentials, minimizing manual intervention while ensuring robust system accountability and compliance.

- Autonomous Key Rotation:The first component focuses on the autonomous rotation of AWS IAM access and secret keys. It operates through a continuous background process that periodically polls existing IAM credentials to determine whether they are nearing expiration or have surpassed a predefined rotation threshold. When such conditions are detected, the system automatically generates a new pair of credentials. These new credentials are then propagated to both the cloud environment and the organization's credential vault, ensuring all systems remain synchronized. This mechanism guarantees that credentials are never statically maintained, thereby aligning with industry best practices for credential hygiene and minimizing the risk of compromised keys.

- Role-Based Credential Access: The second component governs secure user access to credentials through a role-based authorization mechanism. Users access the system via a secure web interface, where their identity and assigned role are authenticated and validated against predefined access control policies. If the user is authorized, the system retrieves the most recent access and secret keys from the credential vault. For first-time users, a new record is automatically created in the vault to support future access. Once authenticated and authorized, the credentials returned to the user. This interaction model ensures that only authorized users can obtain valid credentials and that all key usage is both time-bound and auditable.

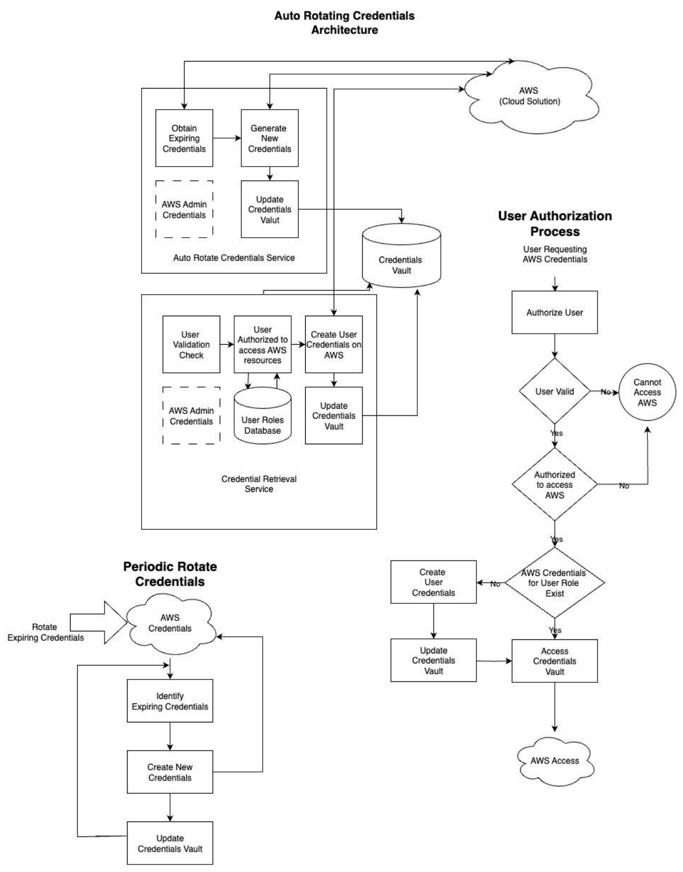

Fig. 1

B. System Architecture

The architecture consists of the following components:

- Periodic Rotate Credentials Service: The purpose of this service is to make periodic calls (every 10 minutes) to AWS IAM service, identify any expiring keys and generate new keys. Using AWS boto3 library, AWS is queried for expiring credentials and also to generate and return new access and secret keys. This service ensures that the credentials between AWS IAM and credential vault are always in sync. A necessary precondition to make calls to AWS is that the service has an AWS access key and secret key, stored as encrypted environment variables, that has Admin privileges, such as create/update/delete IAM users, groups, roles, and policies. This ensures that only an authorized administrator can view and alter credentials, which are the expiring keys that need to be identified and replaced by the newly generated, through AWS boto3, keys. In order to keep track of updates to access and secret keys, the operations will recorded securely using AWS Cloud Watch service.

- Authentication service: This service, that maintains information for authentic users in the system, will have a Muti-factor authentication enabled. The service can easily be altered to align with the existing authentication system of an organization.

- User Authorization service: A Python service is responsible for validating whether a user has the appropriate role—specifically, membership in a user group that grants access to AWS resources. Once the user's role is confirmed, the service checks its credential vault to determine if AWS access and secret keys already exist for the team role associated with the user.
  This check ensures that all members of a given team role can share access to the same AWS credentials. If the credentials are missing, in the case of a first-time user login, the service, upon validation, automatically generates new AWS access and secret keys from the AWS IAM resource. These newly generated credentials are then securely stored in the credentials vault for future use.

- Credentials Vault: To securely manage sensitive user data in a Python application with PostgreSQL, a layered security approach is used. The database is encrypted at rest, connections are secured with SSL/TLS, and secrets are managed via environment variables. Access follows the principle of least privilege, with regular audits. The system enforces role-based access through a RESTful API with a deny-all default policy. Passwords are hashed (e.g., bcrypt, Argon2) with unique salts to resist brute-force and rainbow table attacks [5][6]. AWS keys are tied to roles, not individuals, ensuring only authorized users can access resources. This architecture aligns with best practices for secure credential and data management in modern web systems.

III. RESULTS

To evaluate the performance and scalability of the proposed system, we deployed the solution in a controlled AWS development environment and simulated credential rotation workflows for 20 benchmark IAM users. The testing focused on the two core components of the rotation process: (1) generating new IAM access keys using the boto3 interface, and (2) storing and retrieving the credentials securely from the credentials vault. Each user underwent a full rotation cycle, with operation times measured and averaged to assess system latency and consistency. The results, shown in Figures 2–3, reflect real-world timings collected during these controlled runs, highlighting the system's low overhead and significant improvements over manual key rotation.

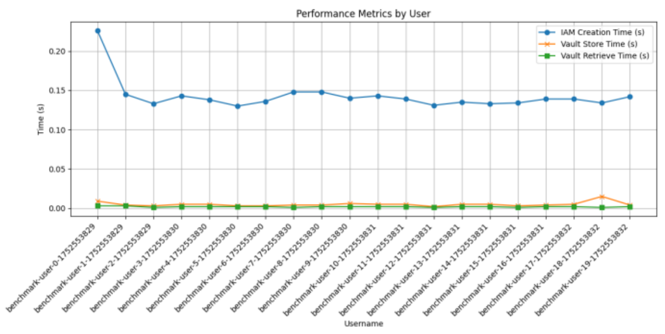

Fig. 2

Fig. 2. Measured IAM access-key creation time, vault storage time, and vault retrieval time for a series of benchmark users (time in seconds). The blue line represents IAM_Creation_Time, orange is Vault_Store_Time, and green is Vault_Retrieve_Time. Each point is one experiment for a different user. The IAM key creation time was on the order of 0.13–0.23 seconds per user, while vault storage and retrieval times were much lower (0–0.01 seconds). Overall, IAM_Creation_Time dominates the operation latency. On average, the IAM access key creation took about 0.146 seconds (sample variance ~0.00034 s²) across users, indicating low latency. Vault storage and retrieval were much faster, with means ≈0.0033 seconds (variance ≈7.27×10^–6 s²) and 0.00022 seconds (variance ≈1.43×10^–8 s²), respectively. The small variances imply consistent performance. These results show that the automated rotation, through the proposed solution, introduces minimal overhead. The figure demonstrates that, aside from an outlier on one user, the times are stable across different runs.

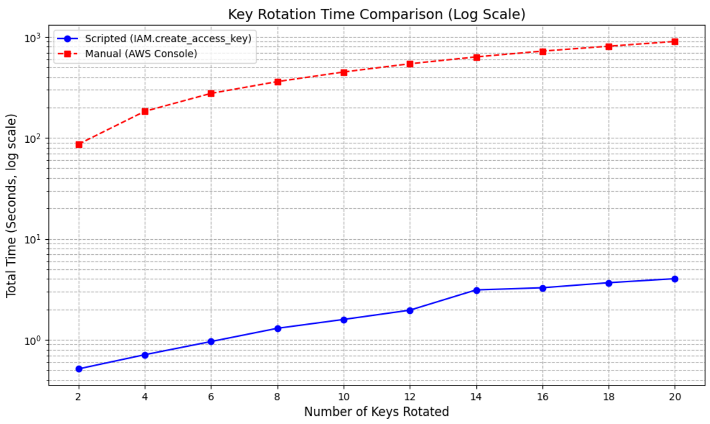

Fig. 3

Fig. 3. Total key rotation time versus number of rotated keys (logarithmic scale). The blue curve shows the automated script (RESTful API) and the red curve shows manual rotation via the AWS Console. The automated method scales nearly linearly (from ~0.5 s for 2 keys up to ~3.5 s for 20 keys), whereas the manual process times grow exponentially (~90 s to ~900 s). This log-scale plot highlights the orders-of-magnitude speedup achieved by automation. Even for small batches, automation outperforms manual rotation by a factor of tens.

The performance graphs collectively indicate that our system efficiently handles key rotations. The pipeline's bottleneck remains the IAM service call (Fig. 2), but even that cost is small. Importantly, by automating batch rotations (Fig. 3), the system vastly reduces administrator labor and latency compared to manual AWS Console operations.

IV. COMPARATIVE ANALYSIS

While commercial solutions such as HashiCorp Vault or AWS Secrets While managers offer similar features, the proposed setup gives organizations full control, flexibility, and lower costs—without being tied to a specific solution. Additionally, the commercial solutions require a time intensive process of creating and configuring each user. Setting up the AWS Secrets Manager, for example, might work autonomously in the long run, however, for each user in the company, resources and time need to be allocated for the initial setup. The proposed product eliminates that lag and increases efficiency, with the configuring built into the code itself, while retaining the long term autonomy. It optimizes all the commercially available options by reducing the time between ideation and creation.

V. FUTURE WORKS

Each of the necessary cloud services, such as AWS S3 buckets and AWS EC2, can have API wrappers that will be called pragmatically, in which case the user won't need to transport their access and secret keys to another platform. As the organization specific authorization mechanism will be used, this process will entirely automate access to cloud services while ensuring confidentiality.

VI. CONCLUSION

This paper presented a secure, automated system for managing the lifecycle of AWS IAM credentials using a RESTful microservice architecture. The proposed solution emphasizes minimal user interaction, periodic credential rotation, centralized key management, and role-based access via a secure web interface. The RESTful, automated AWS IAM key rotation system successfully addresses the core security and usability challenges of credential management in the cloud. By leveraging native AWS tools and enforcing best practices, the architecture offers a scalable and secure method for continuous compliance. The modular design further allows adaptation for other cloud environments. This system contributes to the ongoing conversation about credential automation by offering a replicable and cost-effective alternative.

REFERENCES

1. Amazon Web Services, "Managing access keys for IAM users," AWS Documentation, Feb. 7, 2023.
2. Amazon Web Services, "How to rotate access keys for IAM users," AWS Security Blog, Feb. 9, 2023.
3. Amazon Web Services, "Automatically rotate IAM user access keys at scale with AWS Organizations and AWS Secrets Manager," AWS Prescriptive Guidance, Oct. 6, 2023.
4. Amazon Web Services, "IAM best practices," AWS Identity and Access Management (IAM)
5. A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: The memory-hard function for password hashing and other applications," in Proc. IEEE European Symp. Security and Privacy (EuroS&P), 2016.
6. USENIX, "Bcrypt at 25: A retrospective on password security," USENIX ;login:, vol. 44, no. 1, pp. 27–31, 2019.