iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** X **Month of publication:** October 2026

DOI: https://doi.org/10.22214/ijraset.2026.85139

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

Cloud Resource Scheduling and Interactive Multi-Channel Alert System

Mrs. V. Meena[1], Prasanna M[2], Suriyaa G[3], Vasanthkumar M[4]
[1]*Assistant Professor (Project Guide),* [2, 3, 4]*B.E. Students (Reg. Nos. 6176AC23UCS120, 6176AC23UCS159, 6176AC23UCS171),*
Department of Computer Science and Engineering
Adhiyamaan College of Engineering (An Autonomous Institution), Dr. M.G.R. Nagar, Hosur – 635130, Tamil Nadu, India

Abstract: ***Cloud computing offers on-demand scalability, but developers, academic laboratories and enterprises routinely leave virtual machines, volumes and network resources running after development, testing or maintenance work ends. These forgotten assets silently consume budget, and manual console audits are slow, error-prone and cannot prevent financial leakage at scale. This paper presents the Cloud Resource Scheduling and Interactive Multi-Channel Alert System, an automated governance framework that audits cloud workloads against user-defined operating schedules and idle-utilisation thresholds. When an instance is found running outside its authorised window, the system dispatches prioritised alerts over mobile push, SMS/WhatsApp and e-mail. Every alert carries a cryptographically signed (HMAC-SHA256), time-limited, single-use quick-action link that lets an administrator stop, terminate or extend an instance from a phone without opening the cloud console. If an alert is not acknowledged within a grace period, a fail-safe routine shuts the instance down gracefully. The platform is implemented with Python, Flask, APScheduler, the AWS SDK (Boto3) and MySQL, and adds role-based access control, tamper-evident audit logging and FinOps cost-savings analytics. A working prototype is demonstrated on a sample fleet, and the design is shown to remove the manual-audit, passive-alert and console-access bottlenecks of conventional cloud cost control.***
Index Terms: ***Cloud cost optimisation, resource scheduling, idle resource detection, multi-channel alerting, FinOps, token-based remote remediation, cloud governance.***

I. INTRODUCTION

A. Background

Cloud computing has become the backbone of modern enterprise infrastructure, software deployment and academic research because it provides on-demand scalability, elastic provisioning and operational agility [1]. The same elasticity, however, creates a governance problem. Developers, system administrators and students frequently launch compute instances, persistent storage volumes and networking interfaces for temporary development, testing or maintenance tasks and then forget to stop or terminate them once the work is complete. These orphaned and idle resources keep running in the background, causing continuous consumption, budget overruns and unexpected billing spikes [6], [8].

Conventional controls depend on manual console checks or passive metric dashboards. Both are labour-intensive, slow to react and prone to human error, and the generic e-mail notifications they produce are easily lost in crowded inboxes. Even when an administrator notices an over-running instance, remediation requires logging in to a vendor console, completing multi-factor authentication and navigating several menus, which adds hours of delay during which the meter keeps running.

B. Objectives

The objective of this work is to deliver a lightweight, automated and secure governance framework that prevents unnecessary cloud expenditure. Specifically, the system (i) enforces configurable, policy-driven running schedules for cloud infrastructure; (ii) continuously audits live resources against those schedules and against idle-utilisation thresholds to flag zombie or unmonitored instances; (iii) delivers critical alerts instantly over several channels so that they reach the responsible person; (iv) embeds signed, single-use action links in each alert so that an authorised user can stop, terminate or extend a resource from a mobile device without console login; (v) executes an automatic fail-safe shutdown when alerts remain unacknowledged; and (vi) records every event in an audit trail and quantifies the cost avoided, so that idle consumption can be traced to specific projects, laboratories or sandbox environments.

©IJRASET: All Rights are Reserved

C. Contributions

The main contributions are: a closed-loop pipeline that couples schedule enforcement, idle detection, multi-channel alerting and remote remediation in one platform; a stateless HMAC-signed action-token design with a 15-minute lifetime and single-use enforcement; a grace-period fail-safe that bounds the cost of an unresponsive operator; a FinOps module that converts each intervention into a measurable saving; and a working prototype with a web dashboard, audit-log interface and cloud-account integration page.

II. LITERATURE SURVEY

Recent work on cloud cost control falls into four groups. The first addresses scheduling and cost. Seshanna and Rao [1] align workload allocation with SLA and budget limits and contain cost for steady-state workloads, but are insensitive to short bursts. Wang and Yang [2] use deep reinforcement learning to balance task offloading between edge and cloud, at the price of heavy state-exchange overhead. Lin et al. [5] schedule deadline-constrained workflows by placing non-critical DAG paths on cheaper spot instances, which makes them exposed to spot revocation, and Zhang et al. [7] combine genetic algorithms with local search, though convergence slows as task volume grows. The second group concerns anomaly detection: Jin et al. [3] apply large language models to multi-cloud telemetry for early warning, but inference latency and token cost limit high-frequency use.

The third group is idle-resource governance and FinOps. Devi et al. [6] profile utilisation to detect zombie assets and notify stakeholders, yet decommissioning still depends on a human, leaving a delay between detection and saving. Vance and Robertson [8] enforce lifecycle policy autonomously through tagging compliance, which risks false-positive termination of irregular batch jobs, and Karthik and Subramanian [9] describe a closed-loop FinOps engine whose reliance on vendor-specific APIs hampers portability. The fourth group covers interactive remediation. Sengupta et al. [4] propose token-based verification so that operators can trigger authenticated actions from alert messages, but replay windows open if token lifetimes are misconfigured across unsynchronised nodes, while Farooq and Ananth [10] pair signed actionable alerts with out-of-band incident response and note key-management overhead as the main limitation. Table I summarises these studies.

Table I. Summary of surveyed work and identified gap

Ref.	Focus	Limitation addressed by the proposed system
[1]	Economic/SLA-aware scheduling	No sensitivity to short bursts; no remediation path
[2]	DRL edge–cloud scheduling	Communication overhead; not a cost-governance tool
[3]	LLM-based multi-cloud anomaly alerts	High inference latency and cost; alert-only
[4]	Token-based remote actions	Replay risk if token lifetime is mis-set; no scheduling
[5]	Path-aware workflow scheduling	Spot-instance volatility; batch-DAG only
[6]	Idle-resource detection and notification	Manual decommissioning delays cost realisation
[7]	Hybrid heuristic–metaheuristic scheduling	Slow convergence at scale
[8]	Automated idle-asset FinOps remediation	False-positive terminations; no human override
[9]	Closed-loop FinOps engine	Vendor-specific APIs hinder portability
[10]	Signed multi-channel incident remediation	Key-management overhead; incident-response focus

No single study combines schedule enforcement, utilisation-based idle detection, multi-channel alerting, one-tap signed remediation and an automatic fail-safe with a human override. The present work targets that gap.

III. EXISTING SYSTEM

In the current landscape, organisations and laboratories rely on manual procedures or static dashboards to manage virtualised infrastructure. Operators log in to dense, vendor-specific consoles to find unneeded workloads, and existing alerting is passive: generic, high-volume e-mails that are easily missed. Native cloud platforms do not verify provisioned runtimes against actual working hours, so administrators cannot stop runaway expenses in real time. The principal disadvantages are summarised below.

1) Manual burden. Instance lifecycles depend on human memory and constant console logins; auditing inventory across regions and accounts is error-prone and misses active workloads.

2) Passive, one-way alerting. E-mail and dashboard logs lack urgency and provide no way to act on the resource remotely.

3) Cloud waste. Idle instances, unattached disks and redundant development environments run through nights and weekends without delivering value.

4) Console friction. Acting on a flagged instance means booting a PC, completing multi-factor authentication, locating the resource ID and issuing a stop command.

5) No unified schedule policy. There are no automated guardrails that compare live runtimes with defined operating shifts, giving inconsistent enforcement and no laboratory-level accountability.

IV. PROPOSED SYSTEM

A. Overview

The proposed system is an automated governance architecture that continuously audits cloud workloads, enforces operating schedules and enables instant mobile remediation. A Policy-Driven Scheduler lets administrators define operating windows by project, department or user tag. A Continuous Auditing Service periodically scans the connected accounts, reading instance state, CPU utilisation and runtime. When an instance is running past its schedule or has been idle beyond a threshold, the Interactive Multi-Channel Alert Dispatcher is triggered. Each alert states the resource details and embeds a signed, single-use Quick Action Link for STOP, TERMINATE or EXTEND. If no response arrives before a countdown expires, a safety override performs a fail-safe shutdown.

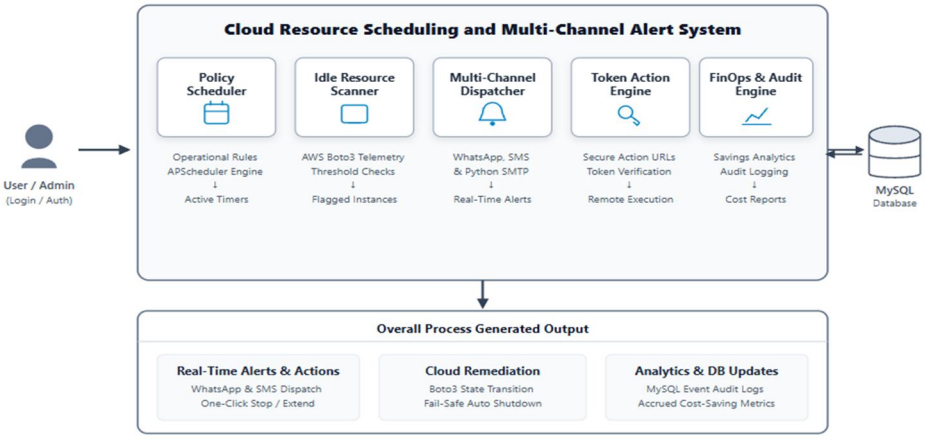


Fig. 1: System architecture of the proposed framework

B. Functional Modules

As shown in Fig. 1, the platform is organised into five cooperating modules around a MySQL data store: the Policy Scheduler, the Idle Resource Scanner, the Multi-Channel Dispatcher, the Token Action Engine and the FinOps and Audit Engine. A user-authentication and dashboard layer sits in front of them and gives authorised staff access to schedules, live workloads, logs and analytics. The flow of data between these modules is shown in Fig. 2: administrators configure schedules, the scanner audits telemetry, flagged instances produce alerts, token actions return state changes through a feedback loop, and the FinOps tracker writes accrued savings and event logs to the database.

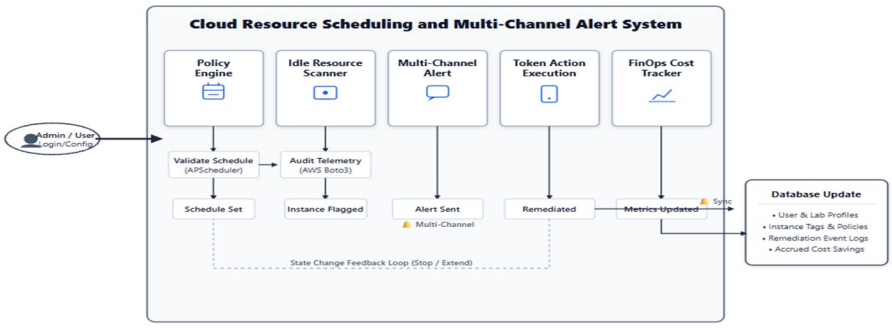


Fig. 2: Data flow diagram of governance and remediation

C. Advantages

The design provides: automated schedule-based cost optimisation instead of manual inspection; interactive remote remediation through secured links delivered straight to the phone; simultaneous multi-channel reachability that mitigates alert fatigue and missed e-mails; rule-based fail-safe shutdowns that bound cost when users are away; and a modular, adapter-based architecture intended to extend to AWS, Azure and GCP and to scale across accounts, teams and laboratories.

D. Problem–Solution Fit and Design Rationale

Four recurring problems motivated the design: uncontrolled cloud spending from forgotten development and sandbox workloads; passive alerts that are missed or buried in noisy inboxes, delaying intervention by hours or days; console-access overhead, since remediating an idle instance requires logging in, completing multi-factor authentication and navigating complex interfaces; and the lack of automated, policy-driven enforcement, which leaves organisations relying on manual vigilance. Each is answered by a specific capability: the policy-driven scheduler manages running hours and prevents off-hours idle runtime; interactive multi-channel alerting provides immediate visibility; signed action links remove the need for console login; and cost analytics with audit logging make every state change transparent for FinOps governance. The ideation phase (Fig. 3) moved from problem identification, through brainstorming of schedule automation, multi-channel alerts and one-click secure actions, to the technology selection summarised in Section V.

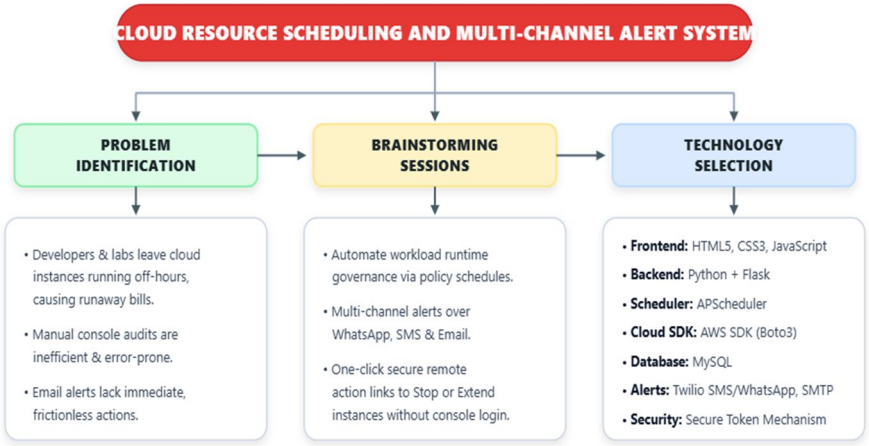


Fig. 3: Ideation and brainstorming outcome

Table II. Existing approach versus proposed system

Aspect	Existing approach	Proposed system
Schedule enforcement	Manual; depends on memory	Automated policy windows (APScheduler)
Alerting	Passive e-mail / dashboard	Push, SMS/WhatsApp and e-mail with context
Remediation	Console login, MFA, navigation	One-tap signed link, no console login
Unattended instances	Run until noticed	Fail-safe shutdown after grace period
Accountability	Little or no traceability	Tamper-evident audit log and savings report

V. IMPLEMENTATION

The system is implemented as a full-stack web application. The front end is written in HTML5, CSS3 and vanilla JavaScript (ES6+) with Fetch-based asynchronous calls to REST endpoints; the back end uses Python and Flask; APScheduler runs background jobs; Boto3 provides EC2 and CloudWatch connectivity; Twilio, SMTP and ntfy handle notifications; and MySQL (with an automatic SQLite fallback for development) stores persistent data. Table III lists the technology stack.

Table III. Technology stack

Layer	Technology	Role
Front end	HTML5, CSS3, JavaScript	Dashboard, forms, status badges, live updates
Back end	Python 3.9+, Flask	REST API, authentication, token verification
Scheduler	APScheduler	Cron/interval jobs; 60-second audit loop; fail-safe timers
Cloud SDK	AWS SDK for Python (Boto3)	EC2 discovery, CloudWatch metrics, start/stop
Notifications	Twilio, Python SMTP, ntfy	SMS/WhatsApp, HTML e-mail, urgent mobile push
Database	MySQL 8.0+ (SQLite fallback)	Users, resources, schedules, tokens, audit trail
Security	PBKDF2-SHA256, HMAC-SHA256	Password hashing; signed single-use action tokens

A. User Registration and Authentication

Operators register with name, official e-mail, mobile number, department and role. JavaScript validates input format and password complexity on the client, and Flask endpoints sanitise input again on the server to guard against injection. Passwords are never stored in clear text: they are salted and hashed with PBKDF2-SHA256 using a 16-byte salt. After a successful login the server creates a session carrying the user identifier, name and role. Role-Based Access Control distinguishes Admin, Manager and Viewer roles: general users register workloads, define schedules and review their team's instances, whereas administrators manage global policies, multi-tenant usage, execution logs and integrations.

B. Policy Scheduler and Discovery Module

The scheduler uses APScheduler inside the Flask process to run cron and interval jobs without blocking HTTP request threads. Administrators define operating policies, for example power-on at 08:00 and shutdown at 19:00 on weekdays with a full shutdown at weekends, or an exact calendar window (DD-MM-YYYY HH:MM) for one-off sessions. Schedules support overnight windows and days-of-week masks. In parallel, the discovery component uses Boto3 to poll EC2 for instance metadata, power state, type, region and tags, so new virtual machines need not be registered manually. Instances without a compliant policy are flagged for review and assigned a conservative default schedule.

C. Idle Asset and Anomaly Scanner

While scheduling prevents off-hours waste, instances that are powered on inside the window but unused also waste money. The scanner queries CloudWatch for average and peak CPU utilisation, network in/out and disk read/write over configurable observation intervals. Using several signals rather than CPU alone avoids false positives for background synchronisation or low-compute streaming. An instance is classed as idle when Eq. 1 holds:

$$\text{idle}(i) = \text{TRUE} \quad \text{if} \quad \text{CPU}_{avg}(w) < \theta_c \quad \text{and} \quad \text{Net}(w) < \theta_n \quad \text{for k consecutive windows w} \tag{1}$$

where θ_c and θ_n are administrator-configured thresholds (for example 5 % CPU and negligible network throughput) and k is the number of consecutive polling windows (for example three 15-minute windows). The scanner also flags out-of-schedule instances, runtime beyond the maximum permitted hours, and resources lacking required tags. For each anomaly it compiles a payload with the instance ID, name, owner, detected idle duration, utilisation metrics and the accumulated cost, estimated from regional hourly rates, and hands it to the dispatcher.

D. Interactive Multi-Channel Dispatcher

The dispatcher runs as an asynchronous pipeline in the Flask back end. It formats each payload into a readable notification giving the cloud provider, instance ID and name, type, department tag, detected idle window and accrued burn rate. WhatsApp and SMS messages are sent through Twilio client libraries to the verified numbers of owners and administrators, an HTML e-mail with telemetry is sent through Python SMTP, and an urgent mobile push (priority 5) with action buttons is sent through ntfy. Every channel carries the signed action links, so the alert itself is the control surface. Delivery status per channel (SENT, FAILED or TRIAL_RESTRICTED) is written to an alert log, and the pipeline is designed to retry or reroute undelivered messages to secondary channels when the primary channel times out.

E. Token Action and Fail-Safe Remediation Engine

For each alert the engine generates action tokens for STOP, TERMINATE and EXTEND. A token encodes the instance ID, action, expiry time and extension length, and is signed with HMAC-SHA256 [11] as in Eq. 2:

$$\text{sig} = \text{HMAC-SHA256}(K, \text{instance_id} \parallel \text{action} \parallel \text{expiry} \parallel \text{minutes}) \quad (2)$$

where K is a server-side secret. Tokens expire after 900 seconds. When an administrator taps a link, a lightweight Flask endpoint verifies the signature and expiry, checks that the token has not been consumed, invokes the corresponding Boto3 call (for example _stop_instances_) or updates the lease timer for an extension, marks the token as used so that it cannot be replayed, and returns a mobile-friendly confirmation page. An EXTEND action grants a temporary lease (60 minutes in the prototype) during which the instance is exempt from enforcement. The complete cycle is given in Algorithm 1.

Algorithm 1. Governance audit loop with fail-safe (executed every 60 s)

```
for each instance r in the fleet with state = running:
    if r has an active extension lease: continue
    obtain schedule S(r); if none: continue
    violation ← (now ∉ S(r).window) or (uptime(r) > S(r).max_runtime) or idle(r) [Eq. 1]
    if not violation: clear alert timer; continue
    if no alert has been sent for r: record t_alert ← now; create STOP / EXTEND tokens (TTL 900 s);
        dispatch alert on all channels; write ALERT_DISPATCHED to the audit trail
    else if (now − t_alert) ≥ G (grace period, 30 min): stop r through Boto3; set state ← stopped;
        write POLICY_SHUTDOWN to the audit trail and update savings
```

The fail-safe therefore guarantees that an unacknowledged alert cannot generate unbounded cost, while operators retain full override through the EXTEND action.

F. FinOps Analytics and Audit Logging

Every operational event, including scheduled shutdowns, manual overrides through tokens, extensions and fail-safe shutdowns, is stored as an append-only record in the audit-trail table with the actor, action and details. The FinOps engine quantifies direct savings for each remediated instance as in Eq. 3:

$$\text{Cost Savings} = \text{Averted Idle Hours} \times \text{Hourly Instance Rate} \quad (3)$$

As a hypothetical illustration, an instance billed at USD 0.10 per hour that is shut down at 19:00 and restarted at 09:00 on weekdays, and stays off at weekends, averts 4 × 14 h + 62 h = 118 idle hours per week, a saving of 118 × 0.10 = USD 11.80 per instance per week; across a laboratory of fifty such instances this reaches USD 590 per week. The actual saving depends on the instance type, region and usage pattern.

Results are aggregated on the dashboard as daily, weekly and monthly savings, active-versus-idle distribution and filterable audit tables, and administrators can export CSV or PDF summaries for departmental chargeback and budgeting reviews.

G. Database Design

MySQL maintains referential integrity across users, compute resources, schedules, alerts and audit records through foreign keys and ACID transactions, which prevents race conditions between scheduler threads and web requests. It also records consumed tokens so that one-click links cannot be reused. Table IV outlines the schema.

Table IV. Principal database tables

Table	Key attributes
users	id, name, e-mail (unique), password_hash, role (Admin / Manager / Viewer)
cloud_resources	instance_id (unique), name, provider (AWS/Azure/GCP), region, state, instance_type, launch_time, lease_expires_at, alert_sent_at
schedules	resource_id (FK, cascade delete), start_time, end_time (daily HH:MM or exact date-time), max_runtime_hours, days_of_week, is_active
alert_logs	resource_id, channel, recipient, status (SENT / FAILED / TRIAL_RESTRICTED), sent_at, details
audit_trails	resource_id, triggered_by, action_taken (ALERT_DISPATCHED, POLICY_SHUTDOWN, STOP, EXTEND_LEASE), timestamp, details

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

VI. SYSTEM REQUIREMENTS

The server requires a 64-bit dual-core CPU at 2.0 GHz or higher (for example an AWS EC2 t3.medium), at least 4 GB RAM and 40 GB or more of SSD storage, running Linux (Ubuntu 20.04/22.04 LTS), Windows Server 2019+ or macOS, with Python 3.9+, a Flask WSGI server such as Gunicorn or Waitress, and MySQL 8.0+. Development uses pip/virtualenv, Git and Visual Studio Code or PyCharm. Client devices need only a modern browser and the ability to receive SMS, WhatsApp or push notifications (1.5 GHz processor, 2 GB RAM; Android 9+, iOS 14+, Windows 10/11 or macOS 11+). A stable broadband, Wi-Fi or 4G/5G connection is needed for telemetry polling through AWS APIs and for dispatching Twilio webhooks.

VII. RESULTS AND DISCUSSION

The prototype was exercised with a sample fleet loaded through the dashboard and a connected AWS account. Fig. 4 summarises the fleet state captured on the dashboard: nine discovered instances, of which three were running, three stopped and three terminated, with eight active compliance policies and an estimated saving of USD 7.87 at the time of capture. Fig. 5 compares the continuous uptime of the three running instances with the 8-hour maximum-runtime policy. Two instances were within the limit (1.19 h and 5.17 h), whereas the batch worker had run for 12.17 h and was therefore flagged as a violation, which is exactly the case that triggers an alert with signed Stop and Extend links and starts the 30-minute grace timer.

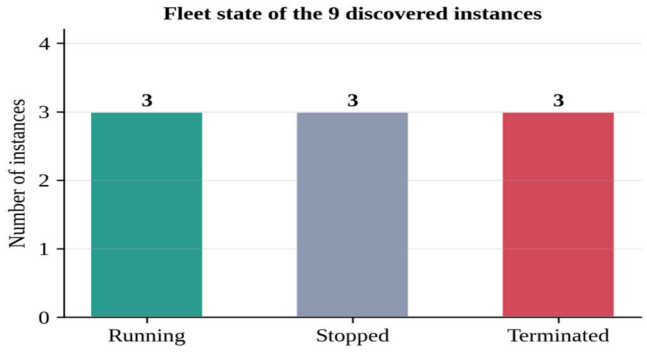


Fig. 4: Fleet state distribution

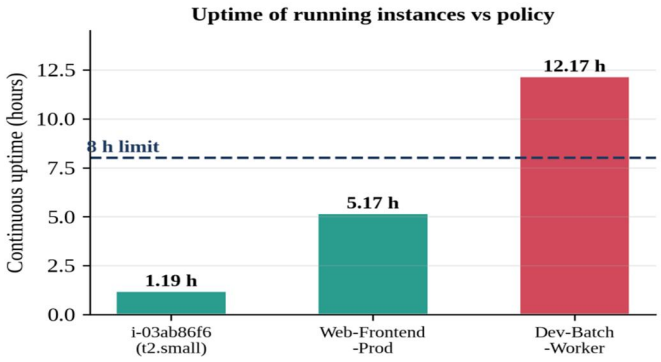


Fig. 5: Instance uptime against the runtime policy

Fig. 6 plots the authorised daily operating windows configured in the schedule module. Both ordinary business-hour windows (for example 09:00–19:00 and 08:00–20:00) and an overnight window (22:30–06:00) were created, showing that the scheduler handles windows that cross midnight; exact date-time windows were also supported. The audit log recorded ALERT_DISPATCHED events for instances running outside their window or beyond the runtime threshold, each noting the start of a 30-minute grace period, together with operator-initiated TEST_ALERT_TRIGGERED events for manual alerts over SMS, WhatsApp and e-mail. Every entry carries a timestamp, actor, action, instance and details, which supports the traceability requirement.

Fig. 7 shows the savings that Eq. 3 yields for a policy that switches instances off at 19:00 and on at 09:00 on weekdays and keeps them off at weekends, which averts 118 idle hours per instance per week. This is an analytical model, not a measurement: the saving grows linearly with the fleet size and the hourly rate, so even modest laboratory fleets accumulate meaningful avoided cost.

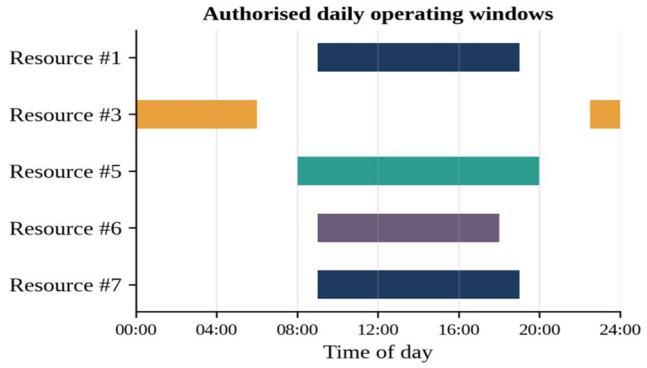


Fig. 6: Configured operating windows

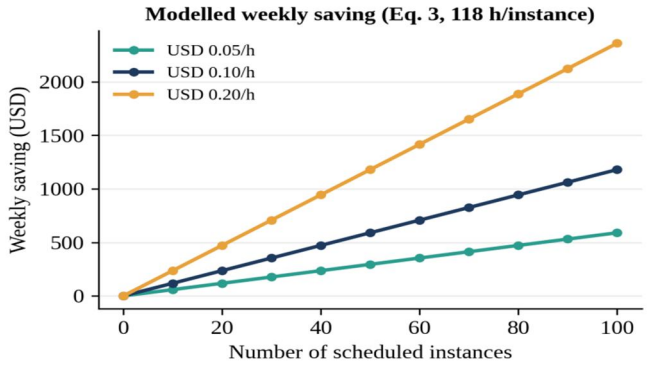


Fig. 7: Modelled weekly saving versus fleet size

©IJRASET: All Rights are Reserved

Table V. Prototype outputs observed during demonstration

Item	Observation
Fleet inventory	9 discovered instances (3 running, 3 stopped, 3 terminated) across AWS, Azure and GCP sample records
Active policies	8 schedules, including daily windows (09:00–19:00, 08:00–20:00, 22:30–06:00) and exact date-time windows
Alert generation	ALERT_DISPATCHED events logged by the scheduler for off-window and over-runtime instances; 30-minute grace period started
Manual test alerts	TEST_ALERT_TRIGGERED events dispatched to SMS, WhatsApp and e-mail by authenticated users
Savings analytics	Dashboard estimate of USD 7.87 saved at capture time (FinOps card)
Account integration	AWS account connected, region us-east-1, status “Connected”, with Test / Discover / Delete actions

A. Discussion

The results show that the pipeline operates end to end: schedule policies are stored, running instances are audited every 60 seconds, violations generate alerts with signed links, and every outcome is logged. Compared with the approaches in Table I, the framework joins detection and remediation in one loop, so the delay between detection and cost realisation depends on how quickly an operator taps a link, and is bounded by the grace period when no one responds. Because the saving of each intervention is computed from averted idle hours (Eq. 3), the effect of the system can be reported per project or laboratory. We note that the figures reported here come from a demonstration fleet rather than a long-running production deployment, so quantitative claims such as reduced mean time to respond remain design goals to be validated in longer field trials.

B. Security Considerations and Limitations

Signed, expiring, single-use tokens limit the damage from forwarded or leaked links, and the 15-minute lifetime narrows the replay window noted in [4]. Several limitations remain. The signing secret must be supplied through protected configuration or a secrets manager and rotated, never embedded in source code. Action links that perform a state change through an HTTP GET request can be triggered by link-preview or security scanners in messaging and e-mail clients, so a confirmation step using POST is advisable. Discovery and remediation are currently implemented for AWS EC2 only, with Azure and GCP shown as integration targets. Finally, idle thresholds should be tuned per workload to avoid terminating irregular batch jobs, the risk identified in [8], which the extension lease and the grace period are intended to soften.

VIII. CONCLUSION AND FUTURE SCOPE

This paper presented a cloud governance platform that combines policy-driven scheduling, utilisation-based idle detection, multi-channel interactive alerting, HMAC-signed one-tap remediation, a fail-safe shutdown and FinOps analytics. By moving remediation to the phone and guaranteeing a bounded response to unattended alerts, the system reduces the manual effort, console friction and passive notification that let idle resources accumulate cost, while the audit trail and savings reports provide accountability for institutions and enterprises.

Future work includes: full Azure and GCP support through unified provider adapters; predictive scheduling that replaces static cron windows with time-series forecasting (for example ARIMA or LSTM) trained on historical usage; bidirectional chatbot remediation through Slack, Microsoft Teams and Telegram using natural-language commands such as “extend 3 hours”; extension of surveillance to containers and serverless services such as EKS, ECS and Lambda provisioned concurrency; and AI-driven FinOps advice on right-sizing, spot substitution and Savings Plans.

IX. ACKNOWLEDGMENT

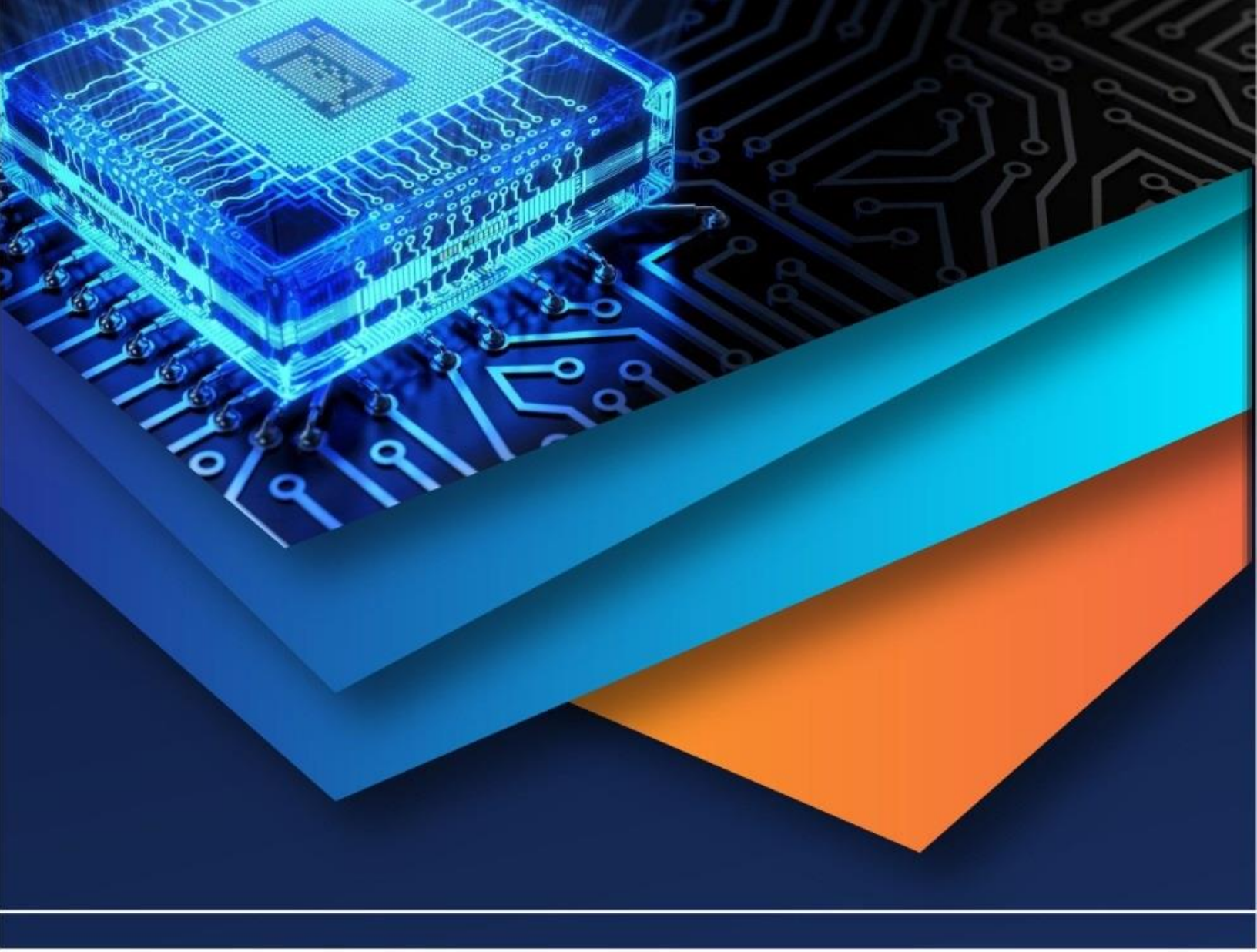
The authors thank Dr. R. Radhakrishnan, Principal, and Dr. G. Fathima, Professor and Head, Department of Computer Science and Engineering, Adhiyamaan College of Engineering, Hosur, for the opportunity and encouragement, and the project coordinator Mrs.V.Meena AP\CSE and faculty of the department for their support.

©IJRASET: All Rights are Reserved

REFERENCES

[1] M. Seshanna and K. B. V. B. Rao, "Economic Impact and Technological Strategies for Resource Scheduling in Cloud Computing," Journal of Cloud Systems and Cost Governance, vol. 14, no. 2, pp. 110–122, 2025.
[2] Y. Wang and X. Yang, "Research on Edge Computing and Cloud Collaborative Resource Scheduling Optimization Based on DRL," IEEE Access, vol. 13, pp. 24150–24162, 2025.
[3] Y. Jin, Z. Yang, J. Liu, and X. Xu, "Anomaly Detection and Early Warning Mechanism in Multi-Cloud Environments Based on LLM," in Int. Conf. Multi-Cloud Reliability and Security, pp. 45–53, 2025.
[4] R. Sengupta, T. Roy, and M. Desai, "Token-Based Interactive Remote Action Mechanisms for Systems Monitoring," IEEE Communications Letters, vol. 29, no. 2, pp. 312–316, 2025.
[5] K. Lin, X. Shen, and M. Zhou, "Path-Aware Scheduling Algorithm for Cost Optimization of Deadline-Constrained Workflows," Journal of Systems Architecture, vol. 148, pp. 103–115, 2025.
[6] S. S. Devi, R. P. Vaijayanthi, et al., "Cloud Cost Optimization Through Idle Resource Detection and Notification," Int. J. Advanced Computer Science and Infrastructure, vol. 16, no. 1, pp. 78–89, 2026.
[7] Z. Zhang, J. Gao, R. Liu, and Q. Tao, "Hybrid Heuristic-Metaheuristic Scheduling for SLA and Cost Optimization in Heterogeneous Clouds," Cluster Computing, vol. 29, pp. 504–519, 2026.
[8] A. N. Vance and C. Robertson, "Automated Idle Asset Remediation and FinOps Governance for Public Clouds," Cloud Governance and FinOps Review, vol. 8, no. 1, pp. 12–25, 2026.
[9] K. R. Karthik and N. Subramanian, "Autonomous Cloud FinOps Engine: Real-Time Resource Auditing and Telemetry-Driven Automated Governance," IEEE Trans. Services Computing, vol. 19, no. 2, pp. 215–228, 2026.
[10] M. E. Farooq and S. P. Ananth, "Multi-Channel Out-of-Band Incident Notification and One-Touch Cryptographic Remediation for Cloud Infrastructure," J. Cloud Computing: Advances, Systems and Applications, vol. 15, no. 1, pp. 42–56, 2026.
[11] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, Internet Engineering Task Force, Feb. 1997.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)