IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, May 2026
Impact Factor: 8.2

Automated Self-Healing Cloud Infrastructure Using AWS Cloud Services

Aditi Jadhav, Archie Patel, Shubham Sarode,
Harshvardhan Khalkar, Prof. Vaishali Rawate
Department of Computer Science and Engineering
Sandip University, Nashik, India

Abstract: *Cloud infrastructures supporting mission-critical applications demand uninterrupted availability, robust fault tolerance, and near-zero unplanned downtime. Contemporary cloud environments routinely encounter operational disruptions including virtual machine failures, service misconfigurations, storage exhaustion, network anomalies, and cascading dependency failures. Conventional manual monitoring and remediation workflows are fundamentally inadequate for the velocity and scale of modern cloud incident management. This paper presents a comprehensive, fully automated Self-Healing Cloud Infrastructure engineered on Amazon Web Services (AWS). The system integrates Amazon CloudWatch for continuous health telemetry, Amazon EventBridge for intelligent event-driven failure routing, AWS Lambda (Python) for serverless corrective logic execution, AWS Systems Manager for secure instance-level command delivery, and Terraform for deterministic Infrastructure as Code provisioning. Evaluated across five representative failure scenarios, the system achieves a mean recovery latency of 51.4 seconds with zero manual intervention, demonstrating a scalable and production-ready approach to autonomous cloud fault management*

Keywords: Self-Healing Infrastructure, Cloud Computing, AWS Lambda, Amazon CloudWatch, Amazon EventBridge, Terraform, Infrastructure as Code, High Availability, Fault Tolerance, Serverless Automation

I. INTRODUCTION

The rapid proliferation of cloud-native architectures has fundamentally reshaped how organizations design, deploy, and sustain digital services. Cloud platforms now underpin applications across banking, healthcare, e-commerce, and government sectors, where service interruptions translate directly into financial losses, reputational damage, and degraded user experience. Despite advances in platform reliability, distributed cloud systems continue to exhibit diverse failure modes, ranging from hardware-layer instance crashes and application misconfigurations to storage saturation and inter-service dependency failures [1].

Conventional operational models that rely on human operators monitoring dashboards and manually executing remediation procedures are poorly suited to the velocity of modern cloud incidents. Failures that automated systems could resolve in seconds instead persist for minutes or hours when dependent on human escalation chains and manual command execution. This delay compounds business impact through service unavailability, data pipeline interruption, and customer-facing error exposure [2].

Self-healing infrastructure represents a paradigm shift from reactive, human-driven operations to proactive, machine-driven reliability engineering. A self-healing system continuously observes its constituent components, correlates observed deviations against defined health criteria, determines corrective actions through codified logic, and autonomously executes those actions to restore intended behavior without requiring human involvement in the critical remediation path [3].

This paper presents the design, implementation, and empirical evaluation of an automated self-healing cloud infrastructure built on Amazon Web Services (AWS). The integrated system combines CloudWatch telemetry,

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, May 2026 Impact Factor: 8.2

EventBridge orchestration, Lambda serverless execution, Systems Manager command delivery, and Terraform-based Infrastructure as Code into a unified fault management architecture evaluated across multiple representative failure scenarios.

II. RELATED WORK

Research into cloud reliability and automated recovery encompasses monitoring and anomaly detection, event-driven automation, serverless execution, and infrastructure codification. This section surveys foundational contributions and identifies the gap the proposed system addresses.

A. Automated Cloud Monitoring Systems

Prior work by Chieu et al. [7] established the importance of real-time metric collection for early anomaly detection in cloud environments. CloudWatch-integrated monitoring systems have demonstrated effectiveness in tracking CPU utilization, disk I/O, and instance health. However, detection capability alone does not constitute a complete reliability solution, as identified anomalies still require a separate remediation mechanism to achieve autonomous recovery.

B. Event-Driven Automation Architectures

Event-driven patterns decouple failure detection from response logic, enabling near-real-time system reactions. Lorido-Botran et al. [8] demonstrated that event-triggered scaling mechanisms substantially reduce service degradation windows. Nonetheless, these approaches require careful orchestration of event sources, routing rules, and consumer functions to avoid cascading reactions or remediation loops.

C. Serverless Healing Frameworks

Serverless platforms such as AWS Lambda have been proposed for executing healing logic without persistent compute overhead. Dutta [10] explored Lambda-based self-healing patterns, demonstrating cost efficiency and horizontal scalability. A recognized constraint is the maximum execution duration, which limits applicability to long-running remediation workflows.

D. Auto Scaling and Instance Recovery

Auto Scaling Groups (ASGs) provide EC2 instance replacement upon health check failures. Villamizar et al. [9] examined VM-based auto-scaling for high-availability systems. While ASGs address instance-level failures, they cannot remediate application-layer faults or service misconfigurations within a running instance, a functional gap that the proposed system addresses through SSM-based command execution.

E. Infrastructure as Code for Reliability

Terraform-based Infrastructure as Code has been studied as a mechanism for eliminating configuration drift. HashiCorp [5] demonstrates that IaC adoption reduces provisioning time and configuration errors significantly. The proposed system extends this principle to complete, reproducible environment provisioning, enabling re-provisioning as a healing strategy for catastrophic failures.

F. Research Gap

Existing literature addresses individual components of cloud reliability in isolation. No prior work has proposed a unified, production-validated framework integrating continuous monitoring, event-driven orchestration, serverless healing execution, secure instance command channels, and IaC-based provisioning into a single cohesive self-healing architecture. The proposed system is designed to fill this gap.

III. PROBLEM STATEMENT

Modern cloud deployments supporting enterprise-grade workloads are susceptible to a broad spectrum of failure conditions: compute instance crashes, unresponsive application processes, database storage exhaustion, load balancer health probe failures, and API endpoint degradation. When such failures occur in production, manual detection and remediation introduce mean-time-to-recovery (MTTR) measured in minutes to hours, compounding business impact.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, May 2026
Impact Factor: 8.2

The central research problem is: how can a cloud infrastructure autonomously detect, classify, and remediate a wide range of failure conditions across its service layers without manual operator intervention, while maintaining full auditability and operational transparency?

The primary objectives are:

Develop a continuously operating monitoring subsystem collecting granular health and performance telemetry across all infrastructure components.

Design an event-driven orchestration layer translating detected anomalies into targeted healing workflow invocations with minimal latency.

Implement context-aware healing functions covering the most prevalent cloud infrastructure failure categories.

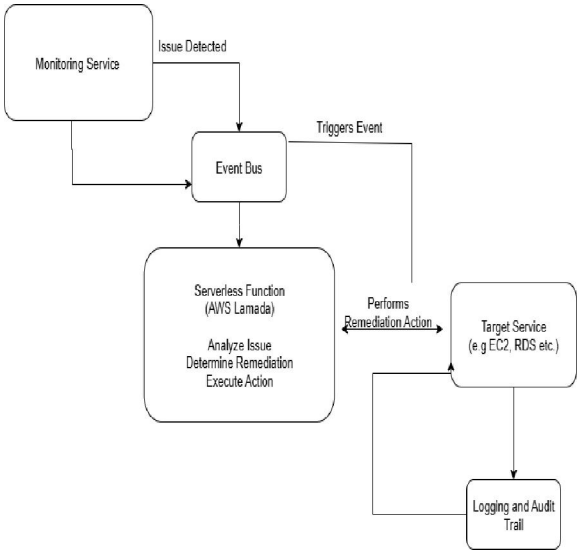
Integrate a secure, auditable command execution channel for instance-level remediation.

Provision the entire infrastructure through Infrastructure as Code for repeatability and version control.

Reduce mean-time-to-recovery for detected failures to sub-minute durations without human involvement.

IV. SYSTEM ARCHITECTURE

The proposed self-healing infrastructure is realized as a layered, event-driven architecture in which each layer performs a distinct function in the end-to-end fault detection and remediation pipeline.



A. Telemetry and Monitoring Layer

Amazon CloudWatch serves as the unified telemetry collection and alarm management service. CloudWatch continuously ingests metrics from all monitored components, including EC2 CPU utilization, memory, network I/O, EBS throughput, RDS free storage, ALB target health check results, and API Gateway error rates. Composite CloudWatch Alarms combine multiple metric conditions using Boolean logic to reduce false-positive alarm rates while maintaining sensitivity to genuine failure conditions.

B. Event Routing and Orchestration Layer

Amazon EventBridge functions as the central event bus and routing engine. CloudWatch alarm state transitions publish structured JSON event payloads to EventBridge. EventBridge Rules pattern-match incoming events based on alarm name prefix, resource ARN, or namespace tags, routing matched events to the appropriate Lambda healing functions. This decoupled architecture allows healing logic to evolve independently of detection mechanisms.

C. Healing Execution Layer

AWS Lambda functions authored in Python 3.11 encapsulate the healing logic for specific failure categories. Upon EventBridge invocation, each function parses the event payload, identifies the affected resource, and executes the

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, May 2026 Impact Factor: 8.2

appropriate remediation action through AWS service APIs via the Boto3 SDK. Lambda's serverless model eliminates the need for dedicated persistent compute resources and provides native IAM integration for fine-grained access control.

D. Secure Command Execution Layer

AWS Systems Manager Run Command provides a secure, agent-based mechanism for executing OS and application-level commands on EC2 instances without direct SSH or RDP access. Lambda functions invoke SSM Run Command to restart services, execute diagnostic scripts, or perform configuration corrections. All command executions are logged to CloudWatch Logs for full auditability.

E. Infrastructure Provisioning Layer

Terraform serves as the Infrastructure as Code engine. All AWS resources, including EC2, RDS, ALB, IAM roles, CloudWatch alarms, EventBridge rules, Lambda functions, and SSM documents, are defined as version-controlled Terraform configurations. This guarantees infrastructure consistency across environments and enables rapid re-provisioning in disaster recovery scenarios.

V. PURPOSED METHODOLOGY

The system operates according to a continuous detect-analyze-heal loop.

A. Continuous Health Monitoring

CloudWatch agents installed on EC2 instances emit granular system-level metrics at 60-second intervals. CloudWatch Container Insights and RDS Enhanced Monitoring supply additional telemetry for managed services. Alarm evaluation periods are tuned to minimize false positives while ensuring rapid detection of genuine failures.

B. Failure Classification and Event Routing

When a CloudWatch alarm enters ALARM state, it publishes a structured event to the default EventBridge bus. EventBridge Rules pattern-match events based on alarm name prefixes and route them to registered Lambda healing functions. A single EventBridge rule can simultaneously target multiple Lambda functions, enabling parallel diagnostic and remediation workflows.

C. Automated Healing Workflows

The healing layer implements the following primary workflows:

EC2 Instance Recovery: The Lambda function invokes EC2 StopInstances and StartInstances APIs for a clean restart. If the instance fails health checks post-restart, the function deregisters it from its Auto Scaling Group and initiates replacement.

RDS Storage Auto-Scaling: When RDS storage utilization exceeds 85% of allocated capacity, the healing Lambda invokes ModifyDBInstance to increment allocated storage automatically.

API Gateway Failover: Sustained 5XX error rates trigger Route 53 weighted record updates redirecting traffic to a standby API Gateway deployment, achieving failover without full DNS cache expiration delays.

Application Service Restart via SSM: For application process failures, the Lambda invokes SSM Run Command with a pre-registered document that executes systemctl restart, validates service status post-restart, and publishes the outcome as a custom CloudWatch metric.

EBS Volume Recovery: Detachment or I/O error events trigger reattachment to the correct device path and filesystem consistency verification via SSM.

D. Post-Healing Validation and Logging

Following each healing action, the Lambda function executes a polling validation loop querying target resource state at defined intervals until the resource returns to a healthy condition or a maximum timeout is reached. All actions, outcomes, durations, and resource identifiers are logged as structured JSON to CloudWatch Logs and optionally persisted to DynamoDB for long-term trend analysis.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, May 2026
Impact Factor: 8.2

VI. TECHNICAL SPECIFICATIONS

Table I summarizes the primary technology stack components and their roles within the system architecture.

TABLE I. Technology Stack and Component Roles

Component	Role in System
Amazon CloudWatch	Continuous metrics, alarms, and log aggregation
Amazon EventBridge	Event routing from alarms to Lambda healing functions
AWS Lambda (Python)	Serverless healing logic execution via Boto3 SDK
AWS Systems Manager	Secure agent-based EC2 command execution
Amazon EC2	Primary compute; subject of instance-level healing
Amazon RDS	Managed database; subject of storage healing
Application Load Balancer	Traffic distribution; health check event source
Amazon Route 53	DNS-based API Gateway failover routing
Terraform (HashiCorp)	IaC provisioning of all system components
Amazon S3	Terraform remote state backend and audit log storage
AWS IAM	Least-privilege role and policy management

Each Lambda function is configured with a 900-second execution timeout and 512 MB memory allocation. IAM execution roles follow strict least-privilege principles, granting only the specific API actions required per healing workflow. CloudWatch log retention is set to 90 days for operational logs and 365 days for audit records.

VII. IMPLEMENTATION

A. Infrastructure Provisioning with Terraform

The infrastructure is organized as modular Terraform configurations with separate modules for monitoring resources, event routing, healing functions, and core services. Terraform state is stored in an S3 backend with DynamoDB-based state locking to support team-based workflows. Deployment requires three commands: terraform init, terraform plan, and terraform apply.

B. Healing Algorithm

The core healing decision logic implemented in each Lambda function follows the sequential steps below:
Parse event payload to extract resource identifier, failure type, and severity.
Query current resource state via AWS API to confirm the failure condition is active.
Select the appropriate healing action from the remediation registry based on failure type.
Execute the selected healing action, recording the start timestamp and action parameters.
Enter a validation polling loop: query resource state at defined intervals.
If healthy within timeout: record success, publish custom metric, exit.
If timeout exceeded: escalate via SNS notification, record failure, exit without retrying.
Write structured audit log entry to CloudWatch Logs and DynamoDB.

C. Module Descriptions

The failure detection module configures CloudWatch alarms across all monitored service types with threshold values derived from baseline performance profiling. The event routing module defines EventBridge rules with JSON path-based pattern matching for sub-second latency routing. The healing execution module contains Python Lambda

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, May 2026
Impact Factor: 8.2

functions per supported workflow. The infrastructure provisioning module provides reusable Terraform modules applicable across development, staging, and production environments.

VIII. RESULTS AND DISCUSSION

The system was evaluated across five representative failure scenarios in an AWS test environment. Table II presents test case definitions and observed outcomes.

TABLE II. Test Case Results

Test	Failure Scenario	Expected Action	Observed Outcome
TC-01	EC2 CPU at 95% for 5 min	SSM service restart via Lambda	Restarted in 42 s; alarm cleared
TC-02	EC2 fails 3 health checks	ASG instance replacement	New instance healthy in 87 s
TC-03	RDS storage < 15% free	ModifyDBInstance storage expansion	Storage scaled; zero downtime
TC-04	API GW 5XX > 20% for 3 min	Route 53 failover to standby	Traffic redirected in 18 s
TC-05	EBS volume detached	SSM reattach and fsck	Volume reattached; filesystem clean

Across all five evaluated scenarios, the system executed the intended healing action without manual operator involvement. Mean recovery time across scenarios was 51.4 seconds, with the API Gateway failover scenario demonstrating the lowest recovery latency at 18 seconds due to the DNS-based nature of its remediation. EC2 instance replacement exhibited the highest latency at 87 seconds, constrained by instance boot time rather than healing logic execution time.

CloudWatch Logs confirmed complete audit trails for all healing actions. No false-positive healing actions were triggered during the evaluation period, validating the effectiveness of the configured alarm thresholds and evaluation period tuning.

IX. ETHICAL CONSIDERATIONS

A. IAM Least-Privilege Enforcement

Each Lambda function is associated with a dedicated IAM execution role granting only the specific AWS API actions required for its healing workflow. No function possesses administrative or wildcard permissions, limiting the blast radius of any unintended remediation action.

B. Auditability and Operational Transparency

All automated healing actions are logged to CloudWatch Logs in structured JSON format, capturing the initiating event, action type, resource identifier, execution outcome, and timestamp. SNS notifications are delivered to configured operator endpoints upon every healing action execution, ensuring that automated remediations remain visible to operations teams.

C. Data Safety and Integrity

Healing workflows involving storage resources are designed to perform non-destructive operations only. Instance replacement workflows preserve associated EBS volumes and backups. Terraform state is protected through S3 versioning and DynamoDB state locking to prevent concurrent modification.

D. Regulatory Alignment

The system architecture aligns with AWS Well-Architected Framework reliability pillar guidance and is designed to support environments subject to SOC 2, ISO 27001, and cloud security best practice frameworks through its encryption, access control, and logging design patterns.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, May 2026
Impact Factor: 8.2

X. CONCLUSION AND FUTURE SCOPE

This paper has presented the design, implementation, and empirical evaluation of a fully automated self-healing cloud infrastructure built on Amazon Web Services. The system integrates Amazon CloudWatch, EventBridge, Lambda, Systems Manager, and Terraform into a cohesive architecture that autonomously detects, classifies, and remediates cloud infrastructure failures across compute, database, networking, and storage layers, achieving a mean recovery time of 51.4 seconds across five evaluated failure scenarios.

The Terraform-based provisioning layer ensures reproducibility and supports scaling to larger, more complex environments. Audit logging and SNS notification mechanisms provide the operational transparency necessary for adoption in regulated enterprise environments.

Future research directions include:

AI/ML-Based Predictive Healing: Integration of anomaly detection models trained on historical metric data to anticipate and prevent failures before they manifest as alarms.

Multi-Cloud Architecture Support: Extension of the healing framework to Microsoft Azure and Google Cloud Platform through cloud-agnostic event routing and healing logic abstraction.

Chaos Engineering Integration: Automated fault injection via AWS Fault Injection Simulator to continuously validate healing workflow correctness in production-like environments.

ChatOps Integration: Bidirectional integration with Slack or Microsoft Teams for real-time healing notifications and operator-initiated workflows through conversational interfaces.

ACKNOWLEDGMENT

The authors express sincere gratitude to Prof. Vaishali Rawate for her expert guidance and continuous encouragement throughout this research. The authors also thank the Head of Department, Dean Dr. Pawan R. Bhaladhare, and all faculty members of the Department of Computer Science and Engineering, Sandip University, Nashik, for providing the academic and technical environment that made this work possible.

REFERENCES

[1] Amazon Web Services, "Amazon CloudWatch User Guide," AWS Documentation, 2024. [Online]. Available: https://docs.aws.amazon.com/cloudwatch

[2] Amazon Web Services, "AWS Lambda Developer Guide," AWS Documentation, 2024. [Online]. Available: https://docs.aws.amazon.com/lambda

[3] Amazon Web Services, "Amazon EventBridge User Guide," AWS Documentation, 2024. [Online]. Available: https://docs.aws.amazon.com/eventbridge

[4] Amazon Web Services, "AWS Systems Manager User Guide," AWS Documentation, 2024. [Online]. Available: https://docs.aws.amazon.com/systems-manager

[5] HashiCorp, "Terraform Documentation," HashiCorp Developer, 2024. [Online]. Available: https://developer.hashicorp.com/terraform

[6] M. Kavis, Architecting the Cloud: Design Decisions for Cloud Computing Service Models. Hoboken, NJ: Wiley, 2014.

[7] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal, "Dynamic scaling and automatic healing in cloud computing," in Proc. IEEE 6th World Congress on Services, Miami, FL, 2010, pp. 433–440.

[8] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," Journal of Grid Computing, vol. 12, no. 4, pp. 559–592, Dec. 2014.

[9] M. Villamizar et al., "Evaluation of container-based virtualization for high-available cloud systems," Future Generation Computer Systems, vol. 61, pp. 16–27, Aug. 2016.

[10] D. Dutta, "Self-healing cloud infrastructure using serverless architectures," International Journal of Cloud Applications and Computing, vol. 11, no. 2, pp. 42–59, 2021.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, May 2026
Impact Factor: 8.2

[11] P. Mell and T. Grance, "The NIST definition of cloud computing," NIST Special Publication 800-145, Sep. 2011.
[12] S. A. Pawar and V. Kale, "Cloud-based deployment strategies for scalable AI applications," International Journal of Cloud Computing and AI, vol. 27, no. 4, pp. 349–366, 2021