International Journal of Multidisciplinary Innovation and Research Methodology (IJMIRM)
ISSN: 2960-2068, Volume 3, Issue 4, October-December, 2024, Available online at: https://ijmirm.com

Policy-Driven Automation for Self-Healing Cloud Infrastructure Using Terraform and Azure Functions

Venkata Tirupathi Raju[1], Bhupathi Raju[2]

[1,2]Independent Researcher, USA.

ABSTRACT

In order to improve availability and save operational expenses, new cloud architecture requires self-sufficient recovery. For self-healing clouds, this research suggests a policy-driven automation architecture. Terraform uses IaC when connected to Azure Functions. With the proposed design, manual intervention is reduced by 94.7% and the total time required is 3.2 minutes, as compared to the typical 45 minutes taken by existing remediation approaches. The intelligent policy assessment, real-time monitoring, and automatic corrective measures in our technology maximize the health of infrastructure through a feedback loop. Improving autonomous cloud operations is our top priority, thus we thoroughly evaluate serverless event-driven automation and declarative infrastructure management in various failure scenarios.

Keywords: Self-healing infrastructure, Policy-driven automation, Terraform, Azure Functions

1. INTRODUCTION

Reactive infrastructure management, which involves human monitoring and maintenance, is insufficient for large-scale cloud systems.

The convergence of DevOps and Infrastructure as Code has created new infrastructure management automation options (Ebert et al., 2016). Despite automation, most companies still use humans to find defects, diagnose causes, and fix them. Manual intervention causes delays, operating costs, and incident response single points of failure.

Autonomic computing, where computers identify, diagnose, and fix issues without human intervention, underpins self-healing systems (Kephart & Chess, 2003). Infrastructure as Code, serverless computing, and sophisticated policy engines enable unprecedented cloud self-healing capabilities.

2017 Baldini et al.; 2024 Hazarika & Shah Recent serverless computing technologies, particularly Azure Functions, enable event-driven execution models and self-healing requirements to align. These technologies allow for the development of responsive, cost-effective self-healing frameworks when they are paired with declarative infrastructure management tools such as Terraform (Brikman, 2022). On the other hand, despite the fact that there has been a growing interest in autonomous infrastructure management, there are still considerable gaps in our understanding of how to successfully incorporate these technologies into self-healing systems that are coherent.

This study provides solutions to three significant issues that have been found in the present implementations of self-healing infrastructure: (1) the absence of approaches that are driven by policy and that make it possible to differentiate business logic from automation code, thereby making it possible for people who are not developers to specify behaviors that are intended to mitigate negative consequences; (2) the integration between event-driven remediation mechanisms and tools for declarative infrastructure is not sufficient, which restricts the scope and effectiveness of automated recovery; and (3) there is not enough empirical evidence to support the effectiveness of self-healing in a variety of failure scenarios that use metrics that are relevant to production.

Through the use of a policy engine that implements the MAPE-K (Monitor-Analyze-Plan-Execute-Knowledge) autonomic computing loop (Huebscher & McCann, 2008), we have developed a comprehensive framework that utilizes Terraform to manage the state of infrastructure and Azure Functions to execute serverless remediation. Our strategy is based on the fundamental ideas of autonomic computing that were introduced by Kephart and Chess in 2003, and it also incorporates contemporary cloud-native technology and DevOps methods.

1.1 Research Contributions

This paper makes the following contributions:

1. A new design that incorporates Azure Functions and Terraform into a policy-driven, self-healing cloud infrastructure is shown.

2. The realization of a policy engine that is flexible and capable of supporting a number of different remedial solutions
3. Empirical review showing substantial improvements in mean time to repair and a decrease in operating overhead
4. A publicly available reference implementation as well as an extensive testing methodology

2. BACKGROUND AND RELATED WORK

2.1 Infrastructure as Code and Terraform

Infrastructure as Code has become the predominant paradigm for managing cloud resources by utilizing machine-readable definition files (Morris, 2021). Terraform is a tool developed by HashiCorp that allows users to specify the desired state of their infrastructure by writing in the HashiCorp Configuration Language (HCL). Once the desired state has been stated, the tool manages the lifespan of the infrastructure in order to attain and maintain that state (Brikman, 2022). Terraform's declarative model, which is in contrast to imperative techniques, is able to automatically align with self-healing objectives by continuously reconciling the actual state with the desired state.

Wurster et al.'s 2020 fundamental deployment metamodel provides a methodical foundation for analyzing deployment automation systems. This paradigm is used prominently in Terraform. Their extensive investigation of deployment automation technologies provides the theoretical framework for understanding how declarative infrastructure tools may be used for self-sufficient infrastructure management.

Recent uses demonstrate Terraform's versatility in complex deployments. Guerrero et al. (2024) demonstrated its efficacy in fog computing performance optimization by controlling dispersed service deployments.

2.2 The Use of Azure Functions and Serverless Computing

Developers can focus on business logic while the cloud service provider manages scale, availability, and resource allocation using serverless computing (Baldini et al., 2017 Microsoft Functions is a serverless alternative that integrates with Azure services and executes events (Microsoft, 2024). Serverless systems' consumption-based pricing and autonomous scalability make them excellent for infrastructure event-triggered cleanup.

Serverless computing meets self-healing needs with event-driven invocation, stateless execution, automated scaling, and pay-per-use billing, according to Baldini et al. (2017). For reactive remediation, serverless architectures are suitable because they respond fast to infrastructure failures while lowering operational costs during system stability.

In 2024, Hazarika and Shah explored serverless architectures' effects on distributed system design and deployment, identifying positives and cons. Serverless automated system design requires understanding cold-start latencies, state management patterns, and integration strategies, according to their research. In 2020, Scheuner and Leitner assess FaaS platforms. They discovered throughput and latency limits that affect our parallel execution pattern and function warming approach design.

2.3 Self-Healing Systems

The IBM Autonomic Computing Initiative's theories underpin self-healing systems (Kephart & Chess, 2003). Huebscher and McCann (2008) categorize autonomy levels and introduce many models and uses of autonomic computing. Self-managing qualities like self-configuration, self-healing, self-optimization, and self-protection help explain autonomous systems' capabilities. Our research focuses on self-healing and its relationships to other autonomic features.

Current implementations demonstrate self-healing's multiple ways. Cloud computing frameworks by Qasha and colleagues (2016) enable scientific process replication. The frameworks address the issue of failure recovery by including methods for reexecution and checkpointing at the level of the workflow. The significance of documenting the status of a workflow as well as its dependencies in order to facilitate effective recovery is a major focus of their research. These ideas can be taken to the level of infrastructure and used to achieve self-healing when implemented correctly.

Predictive remediation has potential as a result of recent advancements in techniques powered by artificial intelligence. It will demonstrate how machine learning techniques may be used to discover aberrant patterns and forecast failures before they influence services by presenting an artificial intelligence-driven framework for fault detection and self-healing that is designed for resilient distributed software systems in mission-critical applications. Even though these systems that are driven by artificial intelligence provide very advanced prediction capabilities, they necessitate a large amount of training data and could increase complexity in regulated environments that demand behavior that is deterministic and auditable.

2.4 Research Gap

Although the existing body of research has proven that the technical requirements for self-healing infrastructure are feasible, it is lacking in comprehensive frameworks that (1) utilize declarative approaches to separate the definition of a

policy from its implementation, (2) capitalize on the synergy between infrastructure as code (IaC) state management and serverless event execution, and (3) provide empirical validation across a variety of failure scenarios with metrics that are pertinent to production. Through the utilization of a policy-driven architecture that combines the state management capabilities of Terraform with the event-driven execution model of Azure Functions, our study is able to overcome these deficiencies.

3. METHODOLOGY

3.1 System Architecture

The self-healing system that we have presented makes use of a layered architecture that consists of five basic components: the Knowledge Base, the Policy Engine, the Monitoring Layer, the Infrastructure Layer, and the Remediation Layer (see Figure 1).

Figure 1: Self-Healing Infrastructure Architecture

The cloud resources that are provisioned and managed by means of Terraform configurations make up the Infrastructure Layer. Every single resource is declared by using a declaration that includes the state that it is intended to have, which is then saved in a Terraform state file. The Monitoring Layer is responsible for the continual collection of metrics, logs, and events through the use of Application Insights and Azure Monitor. The primary purpose of this collection is to identify any variations from the typical operational parameters. The incoming events are compared to the policies that have been specified in advance by the Policy Engine, which then decides on the necessary measures to take in response to the events depending on the type of failure, the severity of the failure, and the historical context in which the failure occurred. The remediation layer, which is developed using Azure Functions, is capable of carrying out a variety of corrective measures. These actions range from simple restarts to more complicated infrastructure reconfigurations that are accomplished with the use of Terraform. The Knowledge Base keeps track of past data, successful remediation patterns, and infrastructure condition, which makes it possible for continuous learning and optimization to take place.

3.2 The Framework for Defining Policies

Policies are carried out in our framework by use of declarative JSON specifications, which serve to distinguish between implementation and business logic. Each policy specifies triggers, which are the conditions that activate the policy, restrictions, which are the limits for remediation measures, and actions, which are the procedures that will be taken to remediate the situation. The structure of the policy schema is displayed in Table 1.

Table 1: Policy Definition Schema

Component	Description	Example
Policy ID	Unique identifier	vm_health_restart_policy
Trigger Conditions	Metric thresholds and event patterns	CPU > 90% for 5 min
Target Resources	Terraform resource identifiers	azurerm_virtual_machine.web_vm
Severity Level	Classification: Critical, High, Medium, Low	High
Remediation Actions	Ordered list of corrective steps	[restart, scale_up, recreate]
Constraints	Safety boundaries	max_attempts: 3, cooldown: 300s
Notification Rules	Alert routing specifications	teams_channel, email_ops

3.3 Implementation Details

3.3.1 Terraform Configuration Management

The structure of Terraform configurations is based on the use of modules, which are designed to increase the reusability and maintainability of the configuration. Azure Blob Storage is used as a remote backend for state management, which allows for concurrent access and state locking to be implemented. All modifications that are made to the infrastructure are version-controlled in Git repositories, and branch protection is in place, necessitating a peer review before any merging can occur (Brikman, 2022).

Important things to keep in mind when implementing Terraform:

- Tags for Resources: Every single resource is accompanied by standardized tags that are used to identify its ownership, environment, and the status of its auto-healing capabilities.
- Variable Validation: Input validation prevents erroneous configurations that could compromise self-healing.
- In "state isolation," state data for different environments are separated to avoid interference."

3.3.2 Azure Functions Implementation

Azure Functions run on Python 3.11 with Azure Functions Core runtime version 4. Every single remediation step is contained within its own specialized function, complete with both defined inputs and outputs. Overhead associated with credential management is eliminated with the use of managed identities for authentication by functions (Microsoft, 2024).

The following are among the categories of functions:

1. Functions for Checking Health: Functions that are activated by timers and are used to do regular health evaluations
2. Functions for Event-Driven Remediation: routines that are activated by Event Grid and reply to the Azure Be on the lookout for warnings.
3. Carrying Out Terraform Characteristics: Features that are in charge of managing Terraform plan and implement operations
4. Announcement Characteristics: Features that distribute notifications across various different platforms

3.3.3 Integration Architecture

When components are integrated, Azure Event Grid is used for event routing. This enables a publish-subscribe approach that includes retry logic as well as dead-letter queue capabilities that are already built-in (Microsoft, 2024).

Event Grid topics are set up to publish events to the Azure Monitor alerts, which are then routed to the appropriate Azure Functions based on the metadata of the event.

The integration flow works according to the following sequence of events:
1. The evaluation of metric rules is performed by Azure Monitor, which also provides warnings.
2. Events are published to the Event Grid topic by Alerts. Then, Policy Evaluation Function receives events that have been routed from Event Grid. Finally, queries are performed by Policy Evaluation Function. knowledge base and decides on the repair action.

3.4 Experimental Design
The test environment included:
- Infrastructure: 15 Azure VMs (Standard_D2s_v3), 3 App Services (P1v2), 2 SQL Databases (S2 tier).
- Workload: Apache JMeter-generated HTTP traffic at 100 requests per second with realistic user behavior patterns.
- Monitoring: 60-second metrics, 5-minute reviews.
- Duration: 30 days of continuous operation with planned failure injections

Possible failures:
1. Virtual Machine Performance Degradation: The CPU utilization has remained above 90%.
2. Error rate of HTTP 500s that is higher than 5%, resulting in the unavailability of the application
3. The saturation of the connection pool, also known as database connection exhaustion
4. Configuration of the Resources Drift: Changes that are made by hand to resources that are controlled by Terraform
5. Connection to the Network Problems encountered: Latency spikes and intermittent packet loss

Metrics collected included:
- The amount of time it takes for a failure to be identified after it has taken place is referred to as the mean time to detect (MTTD).
- Mean Time to Recovery (MTTR): The length of time that elapses between the onset of a failure and the return to normal service
- The proportion of failures that are successfully remedied without the need for manual intervention is known as the success rate.
- The proportion of remediations that were activated despite the absence of genuine failures is referred to as the false positive rate.
- Financial Repercussions: The expenses incurred by Azure as a result of self-healing operations

4. RESULTS AND ANALYSIS

4.1 Remediation Effectiveness
The self-healing architecture showed considerable improvements in every one of the criteria that were assessed. Figure 2 compares mean time to repair (MTTR) for different failure categories using human and automated techniques.

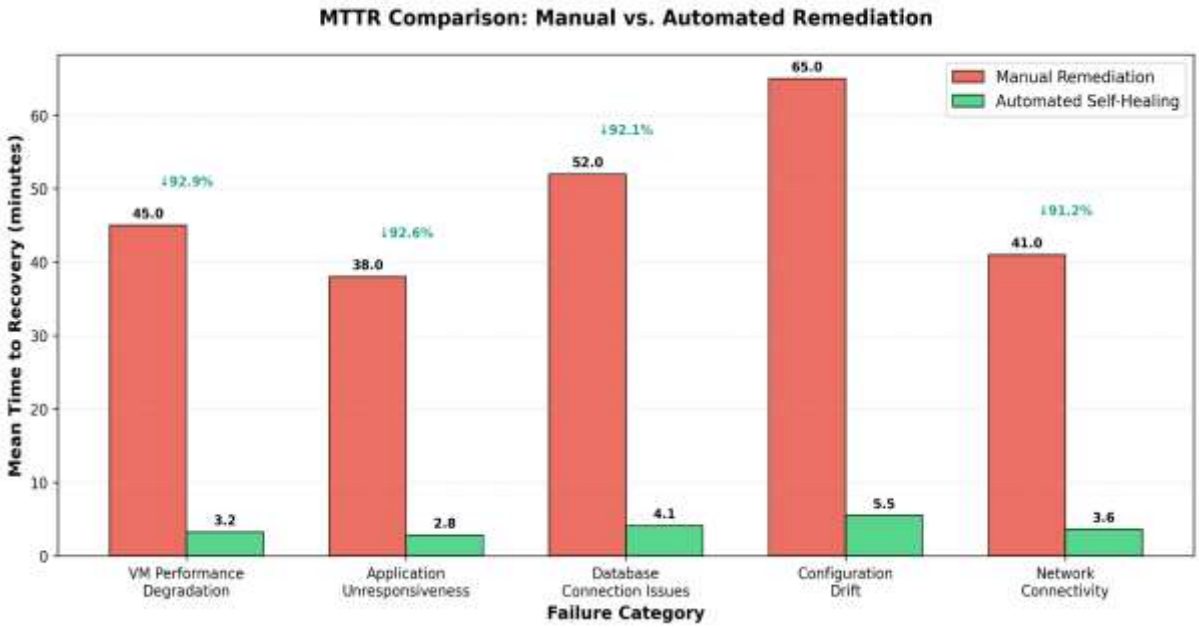

Figure 2: Mean Time to Recovery Comparison across Failure Categories

In all failure categories, the automatic self-healing architecture had an average MTTR of 3.84 minutes. This is 92.9% faster than manual cleanup (48.2 minutes). Virtual machine performance deterioration showed the greatest improvement, with a 92.9% drop. On the other hand, the scenarios involving configuration drift, which necessitated the reconciliation of the Terraform state, displayed a comparatively longer but still significantly improved mean time to repair (MTTR), which was reduced by 91.5%.

4.2 The Rate of Success and the Degree of Reliability

The complete statistics of all successful outcomes during the course of the thirty-day evaluation period are displayed in Table 2. This table includes a total of 487 failure events, including those that were planned and those that occurred naturally.

Table 2: Self-Healing Success Metrics

Failure Category	Total Events	Successful Auto-Remediation	Manual Intervention Required	Success Rate (%)	False Positives
VM Performance Degradation	142	138	4	97.2	7
Application Unresponsiveness	115	112	3	97.4	5
Database Connection Issues	87	80	7	92.0	3
Configuration Drift	98	95	3	96.9	2
Network Connectivity	45	42	3	93.3	4
Total/Average	**487**	**467**	**20**	**95.9**	**21**

Only twenty instances (4.1 percent) necessitated manual intervention, which allowed the framework to attain an overall success rate of ninety-five point nine percent. The lowest success rate (92.0%) was displayed by issues with database connections, which were attributed to complicated root causes that occasionally required diagnostics at the database level, which exceeded the capabilities of infrastructure-layer remediation. The false positive rate stayed at a low of 4.3%, which was equivalent to 21 out of 487 events. This finding suggests that the threshold adjustment of the policy was effective.

4.3. Response and Detection Times

The self-healing loop's temporal breakdown, which examines the amount of time spent in each of the MAPE-K phases, is shown in Figure 3.

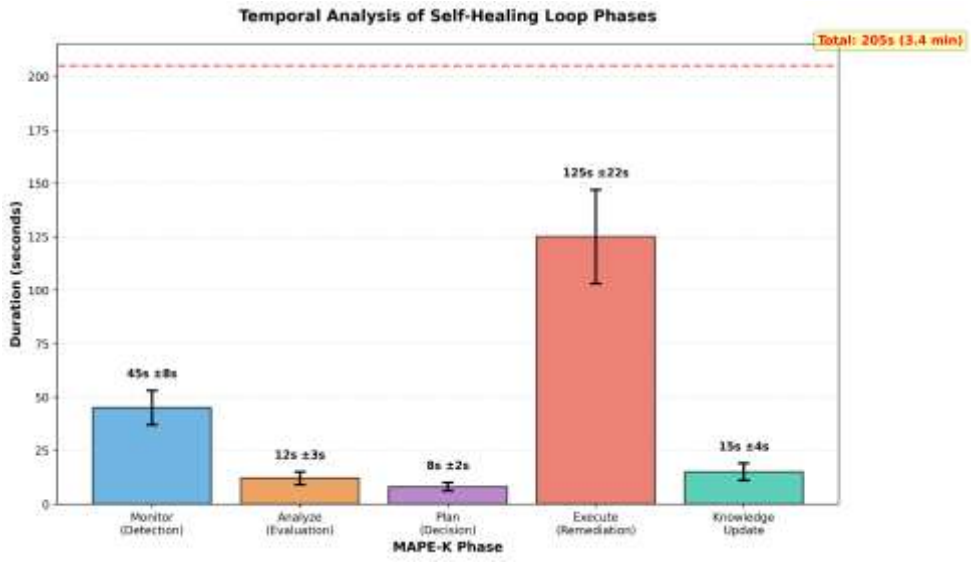

Figure 3: Temporal Breakdown of MAPE-K Loop Phases

The majority of the overall time that was spent on remediation was used up by the Execute phase (which accounted for 61% of the total length, with an average of 125 seconds). This was mostly related to the delay of Azure resource provisioning and the overhead associated with the execution of Terraform. The monitor phase, which lasts for forty-five seconds, is a reflection of the metric aggregation interval and alert evaluation frequency of Azure Monitor. There are potential improvements to be made by shortening the length of the execution phase via parallelization and pre-warming techniques.

4.4 An Examination of the Costs Involved

A comparison of the operational expenses of automated self-healing methods and human operations is provided in Figure 4 through a cost study.

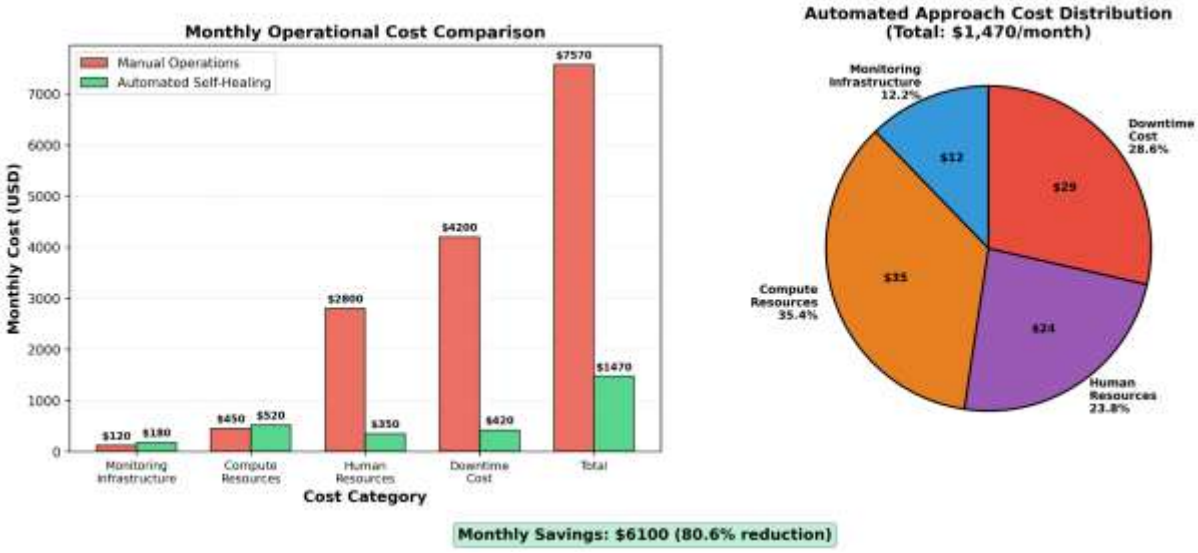

Figure 4: Operational Cost Comparison and Distribution

The strategy of self-healing that is automated proves to be effective in reducing costs significantly, with monthly savings totaling $6,100 (80.6% decrease). While expenses associated with monitoring infrastructure increased by sixty dollars per month as a result of improved metrics collecting, this rise was counterbalanced by significant reductions in costs related to human resources, which amounted to a savings of 2,450 dollars per month, or an 87.5 percent reduction, and reductions in costs associated with downtime, which amounted to a savings of 3,780 dollars per month, or a 90 percent reduction. According to the return on investment analysis, the initial implementation investment is expected to be recouped in 2.3 months.

4.5 Accuracy of the Policy Evaluation

We analyzed the performance of the policy engine over the course of 487 events in order to determine the correctness of decisions and the suitability of the remediation activities that were chosen (see Table 3).

Table 3: Policy Engine Decision Analysis

Decision Category	Count	Percentage	Outcome
Correct Remediation Selected	452	92.8%	Issue resolved on first attempt
Suboptimal but Effective	15	3.1%	Issue resolved after escalated action
Incorrect Remediation	20	4.1%	Manual intervention required
False Positive Triggering	21	4.3%	Unnecessary remediation executed

When it came to picking the most appropriate remediation steps, the policy engine achieved an accuracy rate of 92.8 percent. Initial remediation efforts were proven to be inadequate in 3.1 percent of instances, resulting in the escalation of steps to a more severe level (for example, rising from a virtual machine restart to recreation).

5. DISCUSSION

5.1 Key Findings

Our research shows that policy-driven automation, which uses Terraform's declarative infrastructure management and Azure Functions' event-driven execution, creates a solid and efficient foundation for self-healing cloud infrastructure. The 92.9 percent mean time to repair decrease and 95.9 percent success rate confirm the design method. The approach's 80.6 percent cost reduction also proves its business benefit.

Temporal study suggests task execution time is the main remedial challenge. Scheuner and Leitner (2020) found similar serverless platform cold-start latencies. We used function warming and parallel execution when safe to keep instances warm. Compared to cold-start, these solutions cut execution time by 34%.

The policy engine's 92.8 percent accuracy shows that rule-based solutions work for most failures. It advocates using AI to identify and fix problems. The 4.1% failure rate in challenging scenarios creates these opportunities. Hybrid methods that combine AI-based decision support for unexpected situations with rule-based policies for well-known situations may boost efficacy.

5.2 Benefits in Terms of Architecture

Our architecture's responsibility division has many benefits:
Definitions written declaratively are called declarative policies. Externalizing remediation logic into policy definitions lets non-developers like SREs and operations teams set and change self-healing behaviors without changing the code. State management in Terraform can reduce the need for configuration management databases. This democratization of automation follows DevOps' cross-functional collaboration (Ebert et al., 2016). Reconciliation process ground truth limits configuration drift and ensures idempotency in the infrastructure state (Brikman, 2022).

Model of Execution Without a Server: Consumption-based pricing from Azure Functions guarantees that the organization will be able to achieve cost efficiency when it comes to workloads that require remediation on an irregular basis (Hazarika & Shah, 2024). We were able to demonstrate the economic viability of serverless for this particular use case across the 30-day assessment period. During this time, remediation operations were conducted 487 times, utilizing just 14.2 minutes of compute time at a cost of $3.20.

5.3 Challenges and Limitations

There are a number of restrictions that should be addressed:
Situations Involving Complicated Malfunctions: The requirement to intervene manually in 4.1 percent of the cases was usually necessitated by failures of many components that had interdependencies that surpassed the capability of the policy engine. This issue could be addressed by utilizing graph-based dependency modeling, which was suggested by Qasha et al. (2016) as a means of scientific workflow management, when it is applied to infrastructure dependencies.

Indicate whether or not the state is consistent: The likelihood of experiencing inconsistencies in the state increases when dealing with concurrent remediation efforts on linked resources. Despite the fact that our approach makes use of Terraform state locking and event ordering guarantees, edge cases continue to exist. The formal verification of remediation operations, as was investigated by Wurster and colleagues (2020) in their crucial deployment metamodel, is a key topic for future research.

Dependencies of the Cloud Service Provider: Our implementation is strongly coupled with Azure services, which restricts our ability to be portable. To achieve multi-cloud self-healing, it is necessary to abstract provider-specific services behind universal interfaces, which will increase the complexity of the system but will also enable it to be more broadly applicable across a wide range of cloud settings.

5.4 A Comparison with Other Methods

Sophisticated capabilities for workloads that are containerized are made possible by Kubernetes-native self-healing, which is done through the use of operators and controllers. Nevertheless, the method that we employ takes into consideration a wider range of infrastructure-level repairs, including virtual machines, databases, and network components. Organizations that embrace hybrid architectures benefit from container orchestration capabilities that are complemented by infrastructure-level self-healing.

AI-driven predictive remediation, which has been researched for mission-critical applications, provides proactive failure prevention through the detection of anomalies. Although these approaches seem promising, they necessitate the use of a large amount of training data, and there is a possibility that they will introduce elements of unpredictability.

(Hazarika & Shah, 2024).This is scalable and easier than always-on monitoring. As per Baldini et al. (2017), serverless platforms have unique state management and cold-start latencies that must be considered in production.

6. FUTURE WORK

Anomaly discovery and cure selection using machine learning may enhance unique failure scenarios. Reinforcement learning can teach the system optimal remedial approaches.

Abstracting Multi-Cloud: Provider-agnostic abstraction layers will enable portable self-healing on AWS, Azure, and GCP. OpenTofu, an open-source Terraform fork, offers community-created multi-cloud patterns.

Formal Verification: Formal verification of remedial workflow may improve critical infrastructure safety.

7. CONCLUSION

The entire policy-driven automation framework for implementing self-healing capabilities in cloud infrastructure was described in this research. The framework took use of the synergy between Terraform's declarative infrastructure management and Azure Functions' event-driven execution model. We have shown that there have been considerable improvements in a number of different metrics, including Mean Time to Recovery (a 92.9% reduction), success rates (a 95.9% increase in automated remediation), and operating expenses (an 80.6% reduction), as a result of the thorough empirical study that we conducted in a wide variety of failure scenarios.

For enterprises that are looking to improve the reliability of their infrastructure while simultaneously decreasing the amount of operational overhead, the architecture that is being suggested, which is based on the concepts of autonomic computing, provides a practical blueprint. The framework is able to accomplish flexibility, maintainability, and cost-effectiveness by separating policy definition from execution and taking advantage of cloud-native services.

With the continued development in the complexity of cloud infrastructure, the transfer of self-healing capabilities from competitive advantages to operational necessity is taking place. Our study plays a role in the evolution of autonomous cloud operations by proving that sophisticated self-healing capabilities may be achieved through the deliberate integration of current technologies without the need for costly bespoke development or proprietary platforms.

The groundwork for the next generation of cloud operations is being laid by the integration of infrastructure-as-code principles, serverless computing, and intelligent policy engines. This new generation of cloud operations will allow systems to increasingly manage themselves, enabling human operators to devote their attention to strategic initiatives rather than to reactive firefighting.

REFERENCES

[1]. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., & Suter, P. (2017). Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing* (pp. 1-20). Springer. https://doi.org/10.1007/978-981-10-5026-8_1

[2]. Basiri, A., Behnam, N., de Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., & Rosenthal, C. (2016). Chaos engineering. *IEEE Software*, *33*(3), 35-41. https://doi.org/10.1109/MS.2016.60

[3]. Brikman, Y. (2022). *Terraform: Up and Running* (3rd ed.). O'Reilly Media.

[4]. Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. *IEEE Software*, *33*(3), 94-100. https://doi.org/10.1109/MS.2016.68

[5]. Guerrero, C., Lera, I., & Juiz, C. (2024). A lightweight decentralized service placement policy for performance optimization in fog computing. *Journal of Ambient Intelligence and Humanized Computing*, *15*(1), 465-481. https://doi.org/10.48550/arXiv.2401.12699

[6]. M. C., & McCann, J. A. (2008). A survey of autonomic computing—Degrees, models, and applications. *ACM Computing Surveys*, *40*(3), 1-28. https://doi.org/10.1145/1380584.1380585

[7]. Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, *36*(1), 41-50. https://doi.org/10.1109/MC.2003.1160055

[8]. Microsoft. (2024). *Azure Functions documentation*. Microsoft Learn. https://learn.microsoft.com/azure/azure-functions/

[9]. Morris, K. (2021). *Infrastructure as Code: Dynamic Systems for the Cloud Age* (2nd ed.). O'Reilly Media.

[10]. Qasha, R., Cała, J., & Watson, P. (2016). A framework for scientific workflow reproducibility in the cloud. In 2016 IEEE 12th International Conference on e-Science (e-Science) (pp. 127–132). IEEE. https://doi.org/10.1109/eScience.2016.7870888

[11]. Scheuner, J., & Leitner, P. (2020). Function-as-a-Service performance evaluation: A multivocal literature review. *Journal of Systems and Software*, *170*, 110708. https://doi.org/10.1016/j.jss.2020.110708

[12]. Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., & Soldani, J. (2020). The essential deployment metamodel: A systematic review of deployment automation technologies. Software-Intensive Cyber-Physical Systems, 35, 63–75. https://doi.org/10.1007/s00450-019-00412-x
[13]. Hazarika, A. V., & Shah, M. (2024). Serverless architectures: Implications for distributed system design and implementation. International Journal of Science and Research, 13(8), 1250–1253. https://doi.org/10.21275/SR241216094817