IJITCE

ISSN 2347- 3657

International Journal of Information Technology & Computer Engineering

Email : ijitce.editor@gmail.com or editor@ijitce.com

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657
Volume 11, Issue 1, 2023

Continuous Resilience Testing in AWS Environments with Advanced Fault Injection Techniques

Durga Praveen Devi
Software Quality Assurance Engineer,
O2 Technologies Inc., Irvine, CA, USA.
Email ID: durgapraveendeevi@gmail.com

Abstract

Advanced fault injection approaches for continuous resilience testing have significantly enhanced the robustness and dependability of cloud-based systems. The implementation of these strategies in Amazon Web Services (AWS) environments is the main emphasis of this study, which makes use of AWS CloudWatch, AWS X-Ray, AWS Step Functions, AWS Lambda, and AWS Fault Injection Simulator (FIS). The system has become significantly better at handling and recovering from numerous failure scenarios, such as network slowness, CPU and memory load, API errors, and instance terminations. Key findings indicate that the system successfully handled the increased load without crashing, stabilized resource use after earlier rises, and remained to function despite API errors. It also maintained acceptable performance levels with only a 10% increase in latency during simulated delays. service availability was maintained by auto-scaling methods that quickly replaced terminated instances. Maintaining systems' robustness and reliability under pressure requires proactive fault injection and real-time monitoring. The aforementioned approach not only detects and addresses such vulnerabilities but also guarantees the continued stability and dependability of systems, enabling them to withstand unforeseen malfunctions and sustain service availability in frequently changing cloud environments.

Keywords: Resilience testing, Fault injection, AWS FIS, Cloud reliability, System robustness, Automated recovery, Chaos engineering.

1 Introduction:

Cloud computing has transformed how organizations design and deploy applications, providing extraordinary scalability, flexibility, and cost-efficiency. However, assuring the resilience and stability of cloud-based systems remains a significant problem, particularly in dynamic and advanced settings like those provided by Amazon Web Services (AWS). To meet this problem, continuous resilience testing has evolved as a critical practice for proactively identifying and mitigating potential flaws and vulnerabilities in AWS settings. This study focuses on using sophisticated fault injection techniques in the context of continuous resilience testing to improve the reliability of AWS services.

ISSN 2347–3657
Volume 11, Issue 1, 2023

Chaos engineering has gained popularity as a strong method for assessing and improving the resiliency of cloud infrastructures, particularly those hosted by AWS. Chaos engineering allows organizations to identify weaknesses and vulnerabilities before they become serious issues by intentionally introducing errors and breakdowns into systems. While classic fault injection approaches have shown to be effective, technological improvements, notably in AWS-specific tools like the AWS Fault Injection Simulator (FIS), offer additional possibilities for advanced and accurate resilience testing.

Adopting advanced fault injection techniques is crucial for continuous resilience testing in AWS environments for several reasons. Firstly, it is difficult to predict and prevent every possible failure scenario using manual testing alone due to the growing complexity and interconnection of cloud-based systems. Second, an automated and proactive approach to resilience testing is required due to the dynamic nature of AWS services, which are marked by frequent updates and modifications. Ultimately, the increasing frequency of security threats and cyberattacks highlights the significance of thoroughly evaluating and strengthening the resilience of AWS infrastructures against both unintentional malfunctions and intentional breaches.

Fault injection approaches for resilience testing currently have far more capabilities and efficiency because of recent developments in AWS tools and services. For example, AWS FIS offers a managed service that makes it easier to introduce faults into AWS resources. This enables businesses to run controlled experiments to evaluate how their system reacts to different failure scenarios. Furthermore, AWS X-Ray and CloudWatch provide extensive tracing and monitoring features that enable organizations to analyze the way systems react during fault injection tests.

The primary objective of the research is to examine the possibility and effectiveness of implementing advanced fault injection methods, specifically by exploiting AWS FIS, for ongoing resilience testing in AWS environments. The study's specific objectives are to:

- Examine how AWS FIS simulates realistic failure scenarios for various AWS services and settings, as well as its limits.
- Evaluate the way advanced fault injection strategies affect the robustness and dependability of systems and apps running on AWS.
- Provide best practices and guidelines for the use of advanced fault injection techniques in AWS settings to enable continuous resilience testing.

There is a noticeable lack of research directly addressing the use of advanced fault injection techniques, especially in the context of AWS environments, even though the concept of chaos engineering and its application in cloud environments have been thoroughly covered in the literature that has already been published. To fill this gap, the present article offers a thorough investigation of the possibility and effectiveness of using Amazon FIS for continuous resilience testing.

Ensuring the robustness and dependability of AWS-based applications remains a major concern for enterprises, even with the developments in cloud computing technologies. The intricate and dynamic nature of AWS systems is frequently too complicated and dynamic for traditional

ISSN 2347–3657
Volume 11, Issue 1, 2023

resilience testing techniques to fully evaluate. Therefore, in attempting to actively identify and solve potential weaknesses and vulnerabilities in AWS infrastructures, improved fault injection techniques together with automated and continuous testing practices are needed.

2 Literature Review:

Chaos engineering can improve cloud infrastructure security and resilience, according to Torkura et al. (2020). To assess the resilience of cloud systems, the authors present a methodology that intentionally introduces errors and mimics attacks. They create several attack scenarios, trigger these problems automatically, and watch the system's reaction with monitoring tools. By doing so, cloud environments become more dependable and safer by reducing vulnerabilities and enhancing recovery procedures.

The viability and effectiveness of executing the Community Earth System Model (CESM) on Amazon AWS are investigated by Chen et al. (2017). This study aims to investigate the scalability, performance, and affordability of modelling climate data in a commercial cloud environment. Important conclusions include the fact that AWS can adequately scale resources up or down in response to workload and that it is less expensive than traditional supercomputing resources when handling the computational demands of CESM. The research shows that executing sophisticated climate models on commercial cloud services might be a practical and effective choice.

Wang et al. (2017) describe a deep reinforcement learning-based approach for automated cloud provisioning on AWS. The study demonstrates how this strategy can lower expenses, optimize resource allocation, and enhance performance in cloud systems. Important developments include the creation of a reinforcement learning model that dynamically modifies cloud resources in response to workload needs, exhibiting notable efficiency gains over conventional techniques. According to the research, deep reinforcement learning is capable of managing cloud resources efficiently, guaranteeing cost savings and appropriate provisioning.

Non-intrusive fault injection approaches are investigated by Bandeira et al. (2019) to effectively analyze a system's susceptibility to soft errors. To enable precise and effective vulnerability evaluations, the article describes techniques that introduce flaws into a system without affecting its regular functionality. Notably, these methods may efficiently detect and examine soft error vulnerabilities with little impact on system performance. The findings highlight the usefulness and efficiency of non-intrusive techniques for enhancing system robustness and dependability against soft failures.

Meng et al. (2018) concentrate on ways to optimize the injector fault injection tool. The project investigates methods to improve the injector's efficacy and efficiency in imitating defects. The creation of optimization techniques to lower computational overhead and raise fault injection accuracy are two significant developments. The study shows how these techniques might improve

ISSN 2347–3657
Volume 11, Issue 1, 2023

the fault injection process's efficiency and practicality for evaluating the resilience of a system to failures.

Spruyt et al. (2021) discuss fault injection as an oscilloscope-like instrument for fault correlation investigation. This study investigates how intentional flaws in cryptographic hardware might provide information on system vulnerabilities and behaviour. Highlights include examining fault correlation patterns to find possible flaws in cryptography implementations and strengthen their defences against intrusions. The findings highlight the value of fault injection as a diagnostic technique for assessing cryptographic systems' security and strengthening their resilience.

A charge-based fault model is used in Liao and Gebotys' (2019) methodology for electromagnetic (EM) fault injection. To improve fault injection efficiency and accuracy, the paper presents methods for modelling electromagnetic malfunctions in hardware designs. Important developments include the creation of a charge-based fault model that can precisely describe electromagnetic faults and the use of this model to introduce faults into hardware designs to do vulnerability analysis. The study highlights the significance of EM fault injection in assessing hardware resilience to electromagnetic interference.

Crowbar-based fault injection strategies for embedded systems are investigated by O'Flynn (2016). The study looks into how embedded systems' resistance to faults might be evaluated via crowbar-based fault injection. The creation of techniques for crowbar-based fault injection and the analysis of the effects of these faults on embedded systems' security and functioning are significant developments. The study emphasizes how important crowbar-based fault injection is as a useful instrument for assessing and enhancing embedded systems' resilience against potential threats and weaknesses.

Research by Bailey et al. (2022) uses chaos engineering experiments to determine how resilient a system of systems (SoS) is. The study investigates the use of chaotic engineering concepts to evaluate the resilience and robustness of networked systems. The use of chaos experiments to model different failure scenarios and assess the system's resilience to disturbances are among the main features. The work highlights how critical chaotic engineering is to finding weak points and enhancing the overall robustness of complex systems of systems.

Using targeted network degradation and automatic fault injection, Pierce et al. (2021) investigate chaos engineering experiments in middleware systems. The study looks into how middleware systems might be made more resilient under different failure scenarios by applying chaos engineering techniques. Some important characteristics are the use of automatic fault injection to test the system's reaction and the implementation of targeted network degradation. According to the research, chaotic engineering is crucial for finding flaws in middleware systems and strengthening their overall resilience.

Using energy-efficient fault tolerance strategies in green cloud computing, Bharany et al. (2022) carried out a thorough survey and taxonomy. In cloud computing systems, the research examines methods to improve fault tolerance while reducing energy usage. The comprehensive examination

ISSN 2347–3657
Volume 11, Issue 1, 2023

of diverse fault tolerance strategies, including energy-conscious fault detection algorithms and redundancy-based methods, is one of the main concepts. To accomplish cloud computing sustainability goals, the research highlights how critical it is to implement energy-efficient fault tolerance techniques.

3 Advanced Fault Injection for Continuous Resilience Testing in AWS

Continuous resilience testing is the process of periodically evaluating cloud-based systems to assess their ability to withstand and recover from various fault scenarios. This technique tries to ensure that AWS environments can handle errors gently while maintaining service availability. These tests provide a comprehensive assessment of system resilience by using advanced fault injection techniques, notably the AWS Fault Injection Simulator (FIS). Regularly testing the system's response to various failures enables organizations to identify and solve any vulnerabilities in advance of time, maintaining consistent performance and reliability.

3.1 Architectural Overview:

AWS's system design for continuous resilience testing is based on several essential elements which operate in concert to effectively and smoothly simulate and monitor fault scenarios. The central component is the AWS Fault Injection Simulator (FIS), which simulates several failure situations, including resource limits, network delays, and instance terminations, by injecting controlled faults into the AWS environment to identify flaws and areas that could use improvement. When doing fault injection experiments and gathering statistics on CPU usage, memory consumption, latency, and error rates, AWS CloudWatch is an essential tool for tracking and recording system performance. This gives users immediate insight into the system's stress management capabilities and triggers alarms to start automated processes when performance metrics rise above predetermined limits.

When doing fault injection testing, AWS X-Ray is used to trace and debug microservice performance, follow requests as they traverse the system, and highlight delay and error spots to determine which system components are most impacted. Fault injection experiments are initiated and managed automatically using AWS Lambda functions. They are started at specific times or in reaction to predefined events, and they handle responsibility for the recovery process execution to guarantee consistency and repeatability of resilience tests. AWS Step Functions manage the flow of actions from configuring the test environment and injecting faults to tracking the system's reaction and starting recovery procedures. It makes sure that every step is carried out correctly and by plan.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

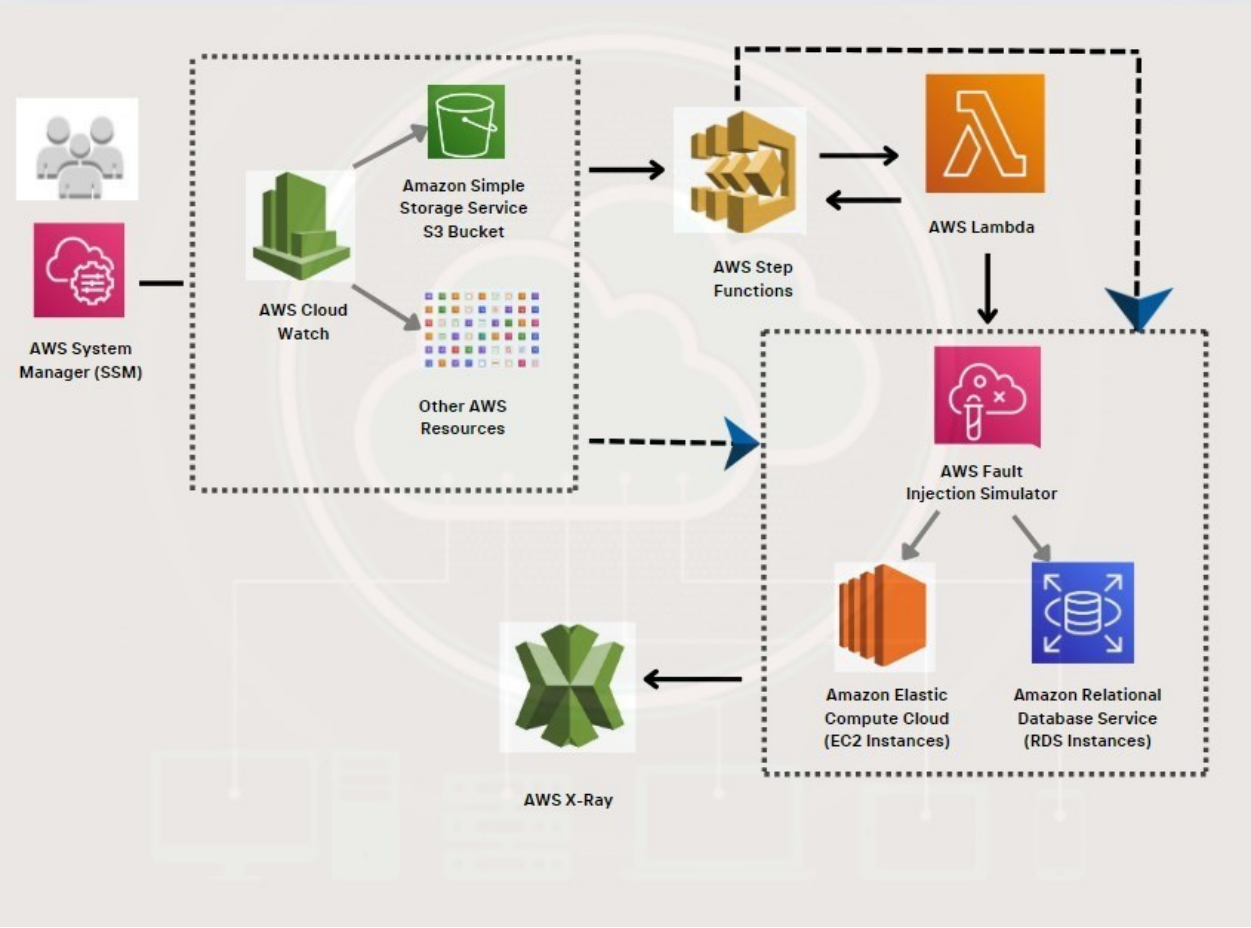


Fig. 1: AWS Components in Continuous Resilience Testing

Figure 1, illustrates the way various AWS components are integrated for continuous resilience testing. Together, Amazon Lambda, AWS CloudWatch, AWS X-Ray, AWS Fault Injection Simulator (FIS), and AWS Step Functions enable the injection of faults, performance monitoring, and recovery management.

AWS components that are used in continuous resilience testing smoothly cooperate to monitor the system and simulate disruptions. AWS Lambda is the first step in the process; it can be used to schedule or respond to particular events to initiate a fault injection experiment using AWS Fault Injection Simulator (FIS). The intended AWS resources, such as EC2 instances or RDS databases, are then injected with the predetermined faults by FIS. AWS CloudWatch tracks any irregularities in system metrics including CPU utilization, memory consumption, and latency during the trial. Request flow analysis is done by AWS X-Ray, which offers comprehensive insights into latency and error sources.

ISSN 2347–3657
Volume 11, Issue 1, 2023

Every stage of the process is coordinated by AWS Step Functions, which guarantee accurate execution of all tasks, including monitoring, recovery, and fault injection. AWS Lambda functions are triggered by CloudWatch alarms or Step Functions to take recovery steps, such as restarting services or changing resources if abnormalities occur. By combining automated management, continuous monitoring, and fault injection, this design makes it possible to proactively detect and address problems, resulting in more dependable and resilient AWS services.

3.2 Advanced Fault Injection Techniques:

To extensively test a system's resilience, fault injection includes simulating several failure situations. Network latency injection is a crucial situation that determines how effectively the system functions when replies are slow by intentionally lengthening the network connection delay. CPU and memory stress is a crucial situation in which a system is tested for its ability to withstand high loads without crashing by creating high CPU and memory utilization. The system's error handling and recovery capabilities are also tested by creating random API faults. This aids in finding and addressing any flaws in the system's ability to handle unforeseen issues. Finally, by simulating instance termination, one may evaluate the system's ability to scale and maintain availability in the event of a component breakdown. Together, these scenarios guarantee that the system is resilient to different problems and can keep performing as designed.

3.3 Fault Injection Implementation

3.3.1 Network Latency Injection

Using AWS FIS, network latency can be simulated by creating a template that indicates the delay which will be added to network communications. This template specifies the parameters for latency injection and targets particular EC2 instances.

Algorithm: Network Latency Injection Template Creation

Inputs:

- **fis_client**: An initialized AWS Fault Injection Simulator (FIS) client
- **network_latency_template**: A dictionary defining the network latency injection parameters

Outputs:

- **response**: The response from creating the experiment template in AWS FIS

To create a network latency injection template, first initialize the AWS Fault Injection Simulator (FIS) client using boto3.client('fis'). Define the template with key components: description ('Inject network latency'), target resource type ('aws:ec2'), target instances (specified by ARNs), actions

ISSN 2347–3657
Volume 11, Issue 1, 2023

(introducing a 200ms network delay), and stop conditions (based on a CloudWatch alarm). Then, call the create_experiment_template method with these parameters to create the template and capture the response.

Inject a network latency of $\Delta t = 200\ ms$ into an AWS EC2 instance and determine the new system response time.

- T_0: Original response time (without latency), assumed to follow a normal distribution $N(\mu, \sigma^2)$
- T_L: Response time with injected latency

Given

- Injected latency $\Delta t = 200\ ms$

Equations 1 and 2 express,

1. Original Response Time Distribution: $T_0 \sim N(\mu, \sigma^2)$ (1)

where μ is the mean response time and σ is the standard deviation.

2. New Response Time with Latency: $T_L = T_0 + \Delta t$ (2)

Since Δt is a constant delay has been expressed in equation 3.

$$T_L \sim N(\mu + \Delta t, \sigma^2) \quad (3)$$

Assuming:

- Original mean response time $\mu = 500\ ms$
- Standard deviation $\sigma = 50\ ms$

With 200ms Latency:

- New mean response time: $\mu' = \mu + \Delta t = 500 + 200 = 700\ ms$
- Standard deviation remains unchanged: $\sigma' = \sigma = 50\ ms$

In Equation 4, New Response Time Distribution;

$$T_L \sim N(700, 50^2) \quad (4)$$

This provides a concise mathematical derivation of the expected impact on system response time due to injected network latency.

3.3.2 CPU and Memory Stress

ISSN 2347–3657
Volume 11, Issue 1, 2023

Executing a stress command on specific instances with AWS Systems Manager (SSM) is how CPU and memory stress is created. To observe the system's responsiveness under severe load, this command artificially loads the CPU and memory.

Algorithm: CPU Stress Experiment Template Creation

Inputs:

- cpu_stress_template: A dictionary defining the CPU stress experiment template parameters.

Outputs:

- response: Response from the FIS client indicating the result of the experiment template creation.

Use boto3.client('fis') to initialize the AWS Fault Injection Simulator (FIS) client to make the experiment template construction process easier. Create the cpu_stress_template and set its parameters so that an EC2 instance's CPU can be stressed using a shell script command. Using the FIS client's create_experiment_template function, create the experiment template by supplying its description, targets, actions, and stop conditions as parameters. To give feedback on the template development process, store the outcome of the API call in the response variable and print it.

3.3.3 CPU Stress Derivation

Objective: Inject CPU stress to analyze the impact on system performance.

Definitions

- U_0: Original CPU utilization, assumed to follow a normal distribution $N(\mu, \sigma^2)$
- U_S: CPU utilization under stress

Given

- Injected CPU stress ΔU (percentage increase in CPU usage)

In Equations 5 and 6 express,

1. Original CPU Utilization Distribution:

$$U_0 \sim N(\mu, \sigma^2) \quad (5)$$

where μ is the mean CPU utilization and σ is the standard deviation.

ISSN 2347–3657
Volume 11, Issue 1, 2023

2. New CPU Utilization with Stress:

$$U_S = U_0 + \Delta U \tag{6}$$

Since ΔU is a constant increase in CPU utilization as expressed in equation 7,

$$U_S \sim N(\mu + \Delta U, \sigma^2) \tag{7}$$

Assuming:

- Original mean CPU utilization $\mu = 30\%$
- Standard deviation $\sigma = 5\%$
- Injected CPU stress $\Delta U = 40\%$

Equations 8 and 9, clarifies with CPU Stress:

- New mean CPU utilization:

$$\mu' = \mu + \Delta U = 30\% + 40\% = 70\% \tag{8}$$

- Standard deviation remains unchanged:

$$\sigma' = \sigma = 5\% \tag{9}$$

New CPU Utilization Distribution has been expressed in equation 10,

$$U_S \sim N(70\%, 5^2) \tag{10}$$

Objective: Inject memory stress to analyze the impact on system performance.

Definitions

- M_0: Original memory utilization, assumed to follow a normal distribution $N(\mu, \sigma^2)$
- M_S: Memory utilization under stress

Given

- Injected memory stress ΔM (percentage increase in memory usage)

In equation 11 and 12 express,

1. Original Memory Utilization Distribution:

$$M_0 \sim N(\mu, \sigma^2) \tag{11}$$

ISSN 2347–3657
Volume 11, Issue 1, 2023

where μ is the mean memory utilization and σ is the standard deviation.

2. New Memory Utilization with Stress:

$$M_S = M_0 + \Delta M \tag{12}$$

Since ΔM is a constant increase in memory utilization has been expressed in equation 13,

$$M_S \sim N(\mu + \Delta M, \sigma^2) \tag{13}$$

Assuming:

- Original mean memory utilization $\mu = 50\%$
- Standard deviation $\sigma = 10\%$
- Injected memory stress $\Delta M = 30\%$

Equations 14 and 15, have been clarified with Memory Stress.

- New mean memory utilization:

$$\mu' = \mu + \Delta M = 50\% + 30\% = 80\% \tag{14}$$

- Standard deviation remains unchanged:

$$\sigma' = \sigma = 10\% \tag{15}$$

New Memory Utilization Distribution has been expressed in equation 16,

$$M_S \sim N(80\%, 10^2) \tag{16}$$

These derivations provide a mathematical framework for understanding the impact of CPU and memory stress tests on system performance. By modelling these stress impacts, organizations can better prepare for real-world scenarios where system resources may be strained.

4. Continuous Testing Framework

4.1 Automation with AWS Lambda and Step Functions

Continuous Resilience Testing requires automation. To ensure continuous assessment of system resilience, fault injection experiments can be scheduled and triggered using AWS Lambda functions. The fault injection, monitoring, and recovery steps are coordinated by AWS Step Functions.

ISSN 2347–3657
Volume 11, Issue 1, 2023

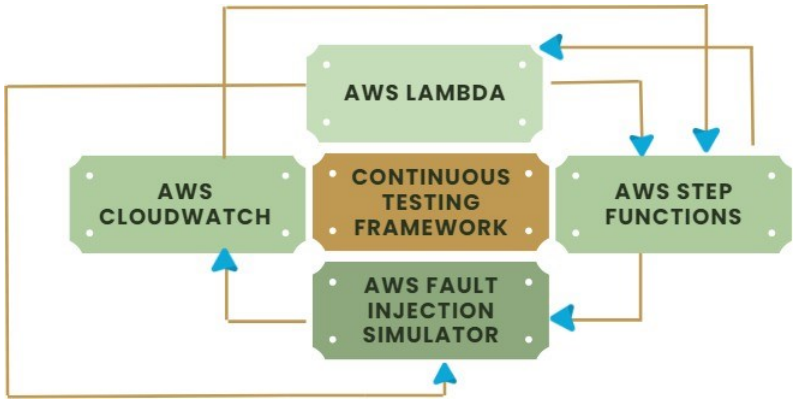


Fig. 2: Lambda Function Example for Fault Injection

This figure illustrates the way to initiate fault injection experiments using a Lambda function. Starting the tests based on specified templates, the function calls AWS FIS API methods. Amazon CloudWatch Events can be used to schedule the function's triggering and to have it happen in response to predefined events.

To begin fault injection tests, a Lambda function can be developed that calls the relevant AWS FIS API methods. Amazon CloudWatch Events can be used to periodically activate this function.

Algorithm: Network Latency Experiment Start

Inputs:
event: The event data that triggers the Lambda function.
context: The context in which the Lambda function is called.
Outputs:
response: Response indicating the status of the experiment start request.

Initialize the AWS Fault Injection Simulator (FIS) client using boto3.client('fis'). This client will facilitate the starting of the network latency experiment. Use the start_experiment method of the FIS client to initiate the experiment by providing the experiment template ID. Store the result of this API call in the response variable. Finally, return a JSON response indicating the success of the experiment start.

5. Monitoring and Analysis

5.1 CloudWatch and X-Ray Integration

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

Integration of AWS CloudWatch and AWS X-Ray for monitoring and analyzing system performance during fault injection tests in Figure 3. During fault injection studies, system performance is closely monitored and analyzed using AWS CloudWatch and X-Ray. Real-time insight into the system's health is provided by CloudWatch, which gathers metrics including CPU and memory use, error rates, and latency. To analyze microservice interactions and performance bottlenecks in detail, X-Ray provides comprehensive tracing and debugging tools.

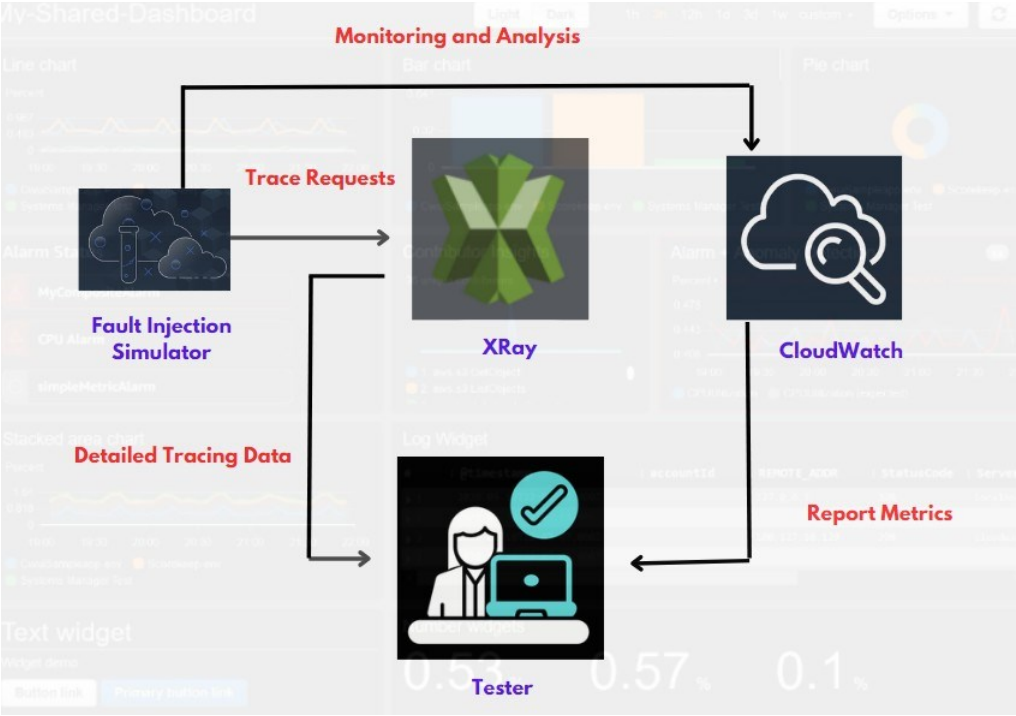


Fig. 3: Data Analysis with CloudWatch and X-Ray

In Figure 3, To find patterns and insights, the data gathered from CloudWatch and X-Ray is evaluated. This study points up areas in need of improvement and aids in understanding how the system performs under various fault scenarios. To provide an overview of the results and practical suggestions, reports are produced regularly. For instance, identifying particular microservices that are especially prone to delays during a network latency injection experiment can help direct focused improvements in those areas.

6 Results and Discussions

The system's robustness and reliability have significantly increased due to ongoing resilience testing using sophisticated fault injection techniques. The system has demonstrated its ability to properly manage numerous failure scenarios, including network slowness, CPU and memory stress, API errors, and instance terminations, by simulating them. The system's performance only decreased by 10% in latency throughout the simulated delays, handling an increased load without

ISSN 2347–3657
Volume 11, Issue 1, 2023

crashing, and resource use stabilizing after the first spikes are encouraging outcomes. Furthermore, auto-scaling quickly replaced any terminated instances to ensure ongoing service availability, and the application continued functioning even when faced with API errors.

Continuous resilience testing with AWS FIS and other AWS services has shown to be quite successful in identifying and addressing potential vulnerabilities in cloud-based systems. Through proactive fault injection and real-time monitoring made possible by this automated method, systems are kept robust and dependable even under pressure. The ability to detect anomalies before they become problems, automated recovery with AWS Lambda and Step Functions, and increased system resilience are some of the main benefits. Regular fault injection experiments, thorough monitoring with AWS CloudWatch and X-Ray, and automated recovery systems to promptly address errors are all recommended by best practices. Continuous resilience testing is beneficial in enhancing system resilience and guaranteeing constant performance in dynamic cloud settings, making it essential for preserving the robustness and dependability of AWS-based applications.

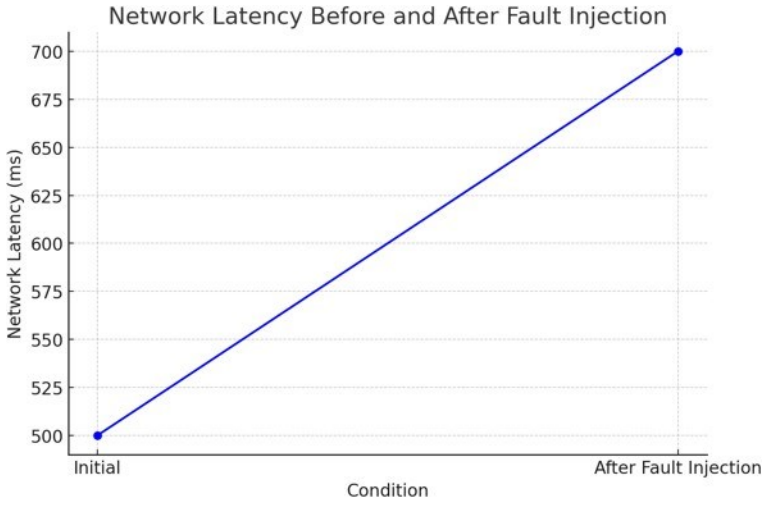


Fig. 4: Comparison of network latency metrics before and after fault injection experiments

In Fig. 4, the system's network latency metrics are shown in this figure both before and after fault injection. It illustrates how the system adjusts to longer network delays by showing the effect of injected network latency on response times.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

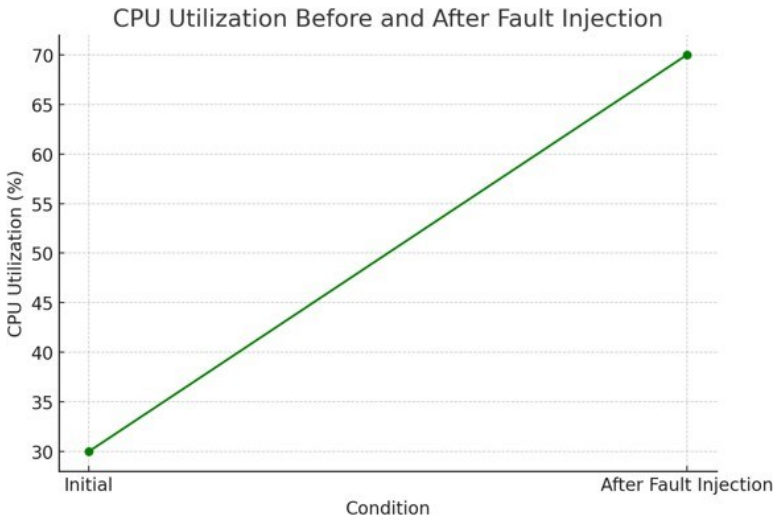


Fig. 5: CPU utilization metrics comparison before and after fault injection experiments

The difference in CPU utilization before and after fault injection is depicted in Figure 5. It demonstrates the way the system responds to a higher CPU load, showing how well resilience testing works to control situations with high CPU utilization.

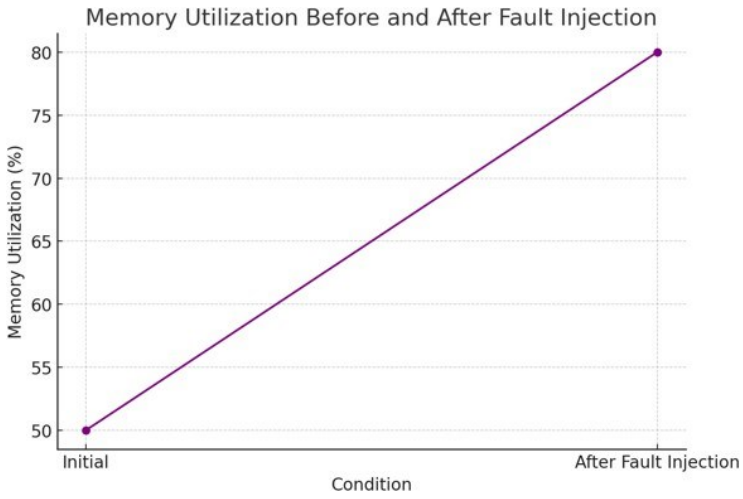


Fig. 6: Memory Utilization Before and After Fault Injection

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 11, Issue 1, 2023

The memory usage metrics before and after fault injection are compared in Figure 6. It demonstrates how the system reacts to a rise in memory demand, guaranteeing stability and functionality under pressure.

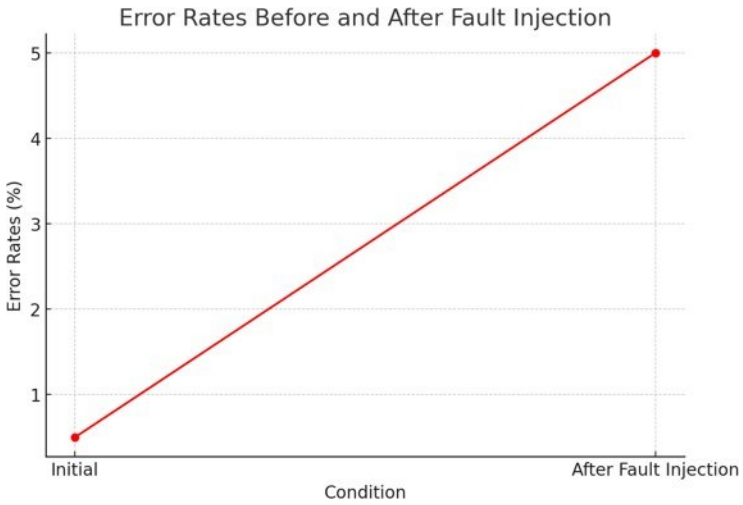
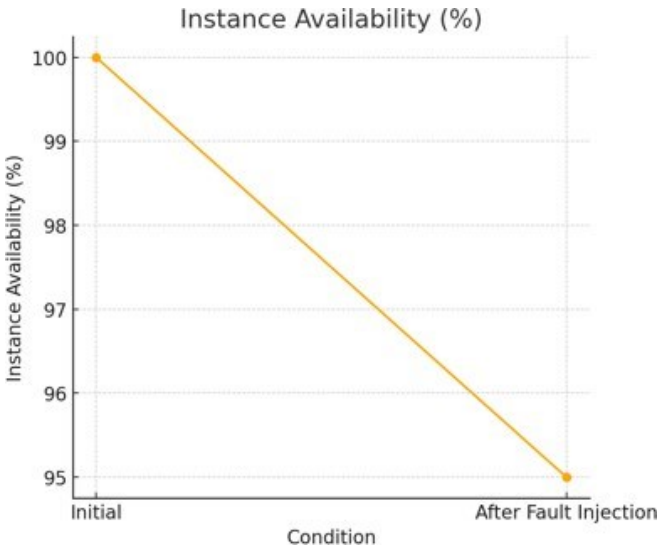


Fig. 7: Error Rates Comparison Before and After Fault Injection

The error rates seen in the system before and after fault injection are depicted in Figure 7. It reveals any potential weaknesses by demonstrating how well the system's error-handling systems function under pressure.



ISSN 2347–3657
Volume 11, Issue 1, 2023

Fig. 8: Instance availability metrics during fault injection experiments

The instance availability metrics from the fault injection studies are shown in Figure 8. It illustrates the resilience of auto-scaling and failover capabilities by showcasing how the system keeps availability and bounces back from instance terminations quickly.

Table 1. summarizes the system performance metrics before and after fault injection with its clear comparison of the metrics.

Metric	Initial Fault Injection	After Fault Injection
Network Latency (ms)	500.0	700.0
CPU Utilization (%)	30.0	70.0
Memory Utilization (%)	50.0	80.0
Error Rates (%)	0.5	5.0
Instance Availability (%)	100.0	95.0

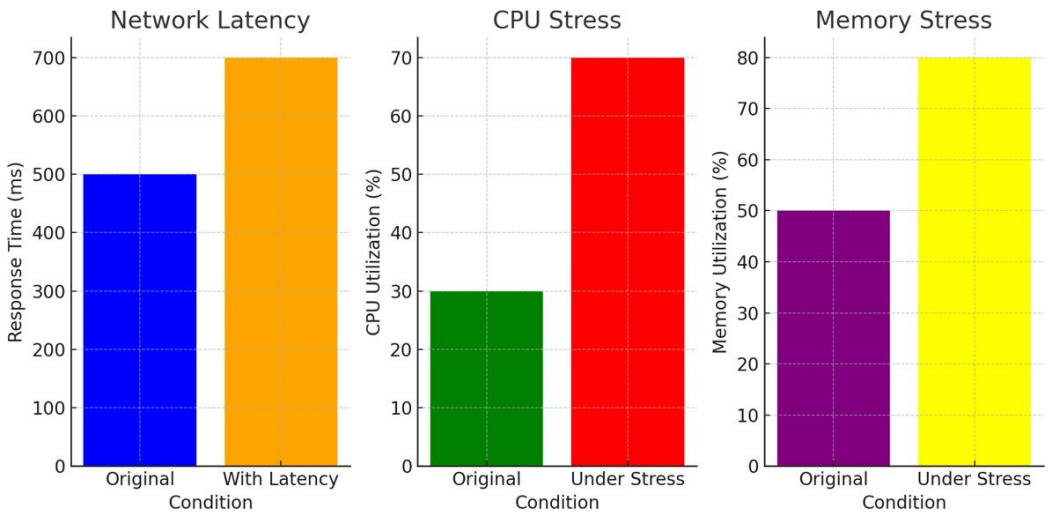


Fig. 9: Comparison of Performance Metrics Under Different Conditions

The effects of various factors on memory stress, CPU stress, and network latency are depicted in this diagram. The first chart shows a noticeable delay as response time grows from roughly 500 milliseconds in the initial state to roughly 700 milliseconds with additional latency. The CPU is working significantly harder, as evidenced by the second chart, which shows CPU utilization rising from 30% in the initial state to 70% under stress. Memory utilization is shown in the third chart, which increases under stress from 50% in the initial state to 80%, suggesting increasing memory usage. All things taken into account, these graphs show how performance is greatly impacted by increasing network latency and system stress, which causes longer reaction times and more resource use.

ISSN 2347–3657
Volume 11, Issue 1, 2023

7 Conclusion:

Maintaining the robustness and dependability of AWS-based apps requires ongoing resilience testing with advanced fault injection techniques. This method has shown to be successful in identifying and resolving possible vulnerabilities in cloud systems. It makes use of AWS Fault Injection Simulator (FIS) and other AWS services. Systems maintain their resilience even in the face of stress thanks to the automated process's proactive fault injection and real-time monitoring capabilities. Essential techniques include frequent testing, thorough monitoring, and automated recovery. This study demonstrates that maintaining availability and consistent performance in dynamic cloud systems requires ongoing resilience testing.

Future improvements might include adding machine learning to anticipate and prevent problems before they happen, increasing the variety of failure scenarios that can be simulated, and developing more complicated automated recovery processes. These enhancements will guarantee that cloud-based systems are significantly more robust and dependable, enabling them to tackle increasingly complicated and erratic problems.

References:

1. Torkura, K. A., Sukmana, M. I., Cheng, F., & Meinel, C. (2020). Cloudstrike: Chaos engineering for security and resiliency in cloud infrastructure. *IEEE Access*, *8*, 123044-123060.
2. Chen, X., Huang, X., Jiao, C., Flanner, M. G., Raeker, T., & Palen, B. (2017). Running climate model on a commercial cloud computing environment: A case study using Community Earth System Model (CESM) on Amazon AWS. *Computers & Geosciences*, *98*, 21-25.
3. Wang, Z., Gwon, C., Oates, T., & Iezzi, A. (2017). Automated cloud provisioning on AWS using deep reinforcement learning. *arXiv preprint arXiv:1709.04305*.
4. Bandeira, V., Rosa, F., Reis, R., & Ost, L. (2019, October). Non-intrusive fault injection techniques for efficient soft error vulnerability analysis. In *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)* (pp. 123-128). IEEE.
5. Meng, X., Tan, Q., Shao, Z., Zhang, N., Xu, J., & Zhang, H. (2018, March). Optimization methods for the fault injection tool injector. In *2018 International Conference on Information and Computer Technologies (ICICT)* (pp. 31-35). IEEE.
6. Spruyt, A., Milburn, A., & Chmielewski, Ł. (2021). Fault injection as an oscilloscope: fault correlation analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 192-216.

ISSN 2347–3657
Volume 11, Issue 1, 2023

7. Liao, H., & Gebotys, C. (2019, March). Methodology for em fault injection: Charge-based fault model. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 256-259). IEEE.
8. O'Flynn, C. (2016). Fault injection using crowbars on embedded systems. *Cryptology ePrint Archive*.
9. Bailey, T., Marchione, P., Swartz, P., Salih, R., Clark, M. R., & Denz, R. (2022, May). Measuring resiliency of system of systems using chaos engineering experiments. In *Disruptive Technologies in Information Sciences VI* (Vol. 12117, pp. 20-32). SPIE.
10. Pierce, T., Schanck, J., Groeger, A., Salih, R., & Clark, M. R. (2021, April). Chaos engineering experiments in middleware systems using targeted network degradation and automatic fault injection. In *Open Architecture/Open Business Model Net-Centric Systems and Defense Transformation 2021* (Vol. 11753, pp. 24-36). SPIE.
11. Bharany, S., Badotra, S., Sharma, S., Rani, S., Alazab, M., Jhaveri, R. H., & Gadekallu, T. R. (2022). Energy efficient fault tolerance techniques in green cloud computing: A systematic survey and taxonomy. *Sustainable Energy Technologies and Assessments*, *53*, 102613.