International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 2, Mar-Apr 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Measuring Self-Healing Efficiency in Docker Containers Using Recovery-Complete Response Delay on AWS EC2

Aditya Deo*, Kalpesh Gaikwad**
*(Department of Information Technology, B.K. Birla College, and Kalyan
adeo86547@gmail.com)
** (Department of IT/CS, B.K. Birla College, and Kalyan
kalpesh.gaikwad@bkbck.edu.in)

--************************----------------------------------

Abstract

Because of the scalability, portability and rapid deployment, Containerized applications are widely adopted in cloud environments. With Docker's Restart policies and other automated self-healing mechanisms, the modern container platforms provide recovery from failures. However the real recovery experience from the user's perspective is not interpreted by traditional evaluation metrics like uptime and restart frequency. This paper presents a new metric named as Recovery-Complete Response Delay (RCRD), which evaluates the time period that passes when the fault occurs till the successful responses of restoration of the application. Using AWS EC2 instance, a Dockerized NGINX container was set up with automatic restart policies in order to analyze the efficiency of RCRD. Faults like out-of-memory (OOM), CPU stress and forced process termination were introduced intentionally. Automated monitoring scripts were used to solidify temporal accuracy. The findings showcases the limitations of traditional metrics and points up the importance of time-based evaluation. RCRD provides user-centric and thorough measure of system resilience in cloud-native applications.

Keywords — **Container Resilience, Fault Injection, Docker, Cloud Reliability, Self-Healing, RCRD**

--************************----------------------------------

I. INTRODUCTION

Containerization has become a foundational technology in modern cloud computing due to its lightweight virtualization, portability, and rapid deployment capabilities. Platforms such as Docker and Kubernetes provide automated recovery mechanisms that restart failed services with minimal manual intervention. However, commonly used resilience indicators such as container restart counts or uptime percentages provide limited insight into how quickly an application becomes responsive again from an end-user perspective.

In practical deployments, a container may restart successfully while the application remains temporarily unavailable due to initialization delays, resource contention, or system instability. This discrepancy motivates the need for metrics that capture recovery behaviour beyond binary availability. In this work, we focus on the Recovery-Complete Response Delay (RCRD), which measures the elapsed time between fault injection and the restoration of successful application level responses.

II. OBJECTIVES

The primary objective of this study is to evaluate the effectiveness of automated self-healing mechanisms in containerized applications using a time-aware resilience metric. Unlike traditional availability metrics that rely on restart counts or uptime, this work focuses on quantifying user perceived recovery behaviour.

The specific objectives of this research are as follows:

- To formally define Recovery-Complete Response Delay (RCRD) as a practical metric for measuring container recovery latency.
- To experimentally evaluate the recovery behaviour of a Docker-based NGINX container under different fault injection models.
- To compare recovery characteristics across CPU stress, process termination, and out-of-memory (OOM) fault scenarios.
- To analyse whether restart counts alone are sufficient to represent resilience in containerized systems.
- To demonstrate the relevance of time-based resilience metrics for cloud-native application reliability assessment.

Through these objectives, the study aims to provide empirical evidence supporting the need for more expressive resilience metrics in modern cloud environments.

III. MATERIALS AND METHODS

Fault tolerance in distributed systems has traditionally relied on redundancy, replication, and failover mechanisms. In cloud native environments, failures often arise from transient conditions such as resource exhaustion rather than complete system crashes. Metrics such as Mean Time to Recovery (MTTR) are often coarse-grained and manually estimated, making them unsuitable for fine-grained resilience analysis.
RCRD addresses this limitation by measuring recovery latency directly at the application response level. This makes it particularly relevant for containerized workloads, where automated orchestration masks failures while users remain sensitive to service responsiveness.

A. Recovery-Complete Response Delay (RCRD)

Recovery-Complete Response Delay (RCRD) is defined as the time interval between the moment a fault is injected into the system and the moment the application resumes successful responses to client requests.

$$RCRD = T_{\text{recover}} - T_{\text{inject}}, \quad RCRD \geq 0 \tag{1}$$

where T_{inject} represents the timestamp at which the fault is introduced, and $T_{recover}$ denotes the timestamp corresponding to the first successful HTTP response observed after fault injection.
Unlike restart-based metrics, RCRD captures recovery delays caused by container re-initialization, resource reallocation, and transient instability that may not be reflected in restart counts alone.

B. Infrastructure

The experiments were conducted on an AWS EC2 Linux instance running Ubuntu 22.04 with Docker Engine installed. An NGINX container was deployed with Docker's restart=always policy enabled to provide automatic recovery after failure.

C. Application

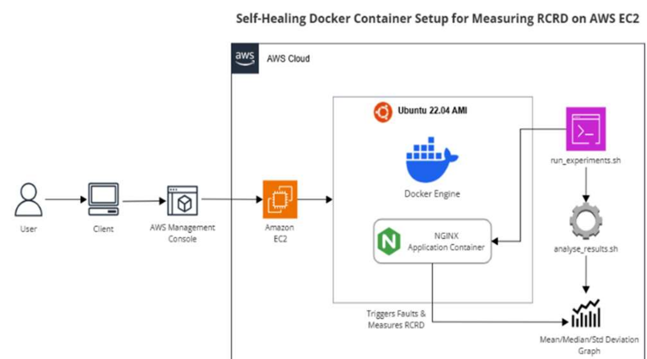

Fig 1: Experimental architecture for fault injection and RCRD measurement

NGINX was selected due to its widespread use as a web server and reverse proxy. Application health was continuously monitored using periodic HTTP requests issued via curl.

D. Automation

Shell scripts were developed to automate fault injection, timestamp recording with nanosecond precision, recovery detection, and CSV-based logging. Each experiment was repeated ten times to ensure statistical consistency.

E. Fault Injection Models

Three fault types were evaluated:

 ©IJSRED: All Rights Reserved

1) CPU Stress Fault:

A CPU-intensive workload was injected using stress-ng to simulate computational overload without terminating the container.

2) Process Kill Fault:

The NGINX process was forcefully terminated to trigger Docker's restart mechanism and observe recovery behaviour.

3) Out-of-Memory (OOM) Fault:

Memory exhaustion was induced using stress-ng, resulting in kernel-level OOM conditions and delayed stabilization.

IV. RESULTS AND DISCUSSION

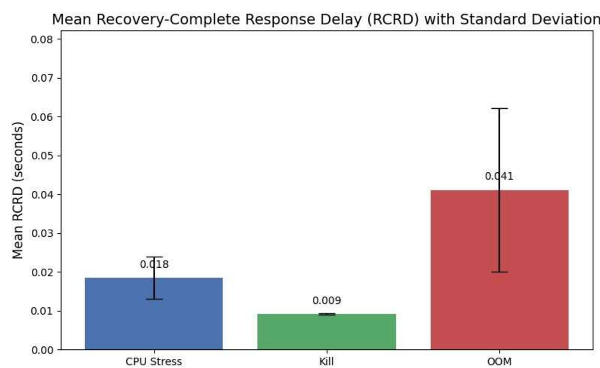

Fig 2: Distribution of Recovery-Complete Response Delay (RCRD) under stress, kill, and out-of-memory (OOM) fault injections

A. Results

Figure 2 illustrates the distribution of RCRD across the evaluated fault models. Process kill faults exhibit the lowest and most consistent recovery times, indicating deterministic restart behaviour. CPU stress faults show moderate variability due to transient performance degradation. OOM faults demonstrate the highest variance and extreme recovery delays, reflecting the non-deterministic nature of kernel-level memory reclamation.

TABLE I
SUMMARY OF RCRD STATISTICS

Fault Type	***Mean (s)***	***Median (s)***	***Std. Dev.(s)***
CPU Stress	0.0185	0.0177	0.0054
Kill	0.0092	0.0092	0.0002
OOM	0.0411	0.0343	0.0210

B. Discussion

The experimental results highlight clear differences in recovery behaviour across the evaluated fault models, even when traditional indicators such as restart count remain unchanged. This observation underscores the limitations of binary availability metrics in capturing the full impact of failures on user perceived service quality.
Kill-based faults exhibited the lowest and most stable RCRD values. This can be attributed to Docker's deterministic restart mechanism, which rapidly replaces the failed process with minimal initialization overhead. Such behaviour suggests that crash-only failures are efficiently handled by container runtimes.
In contrast, CPU stress faults resulted in moderate recovery delays with higher variance. Since the container remains running during stress injection, recovery depends on gradual resource contention resolution rather than explicit restart actions. This leads to transient instability that is not reflected in restart statistics.
OOM faults produced the highest RCRD values and the widest distribution. Memory exhaustion triggers kernel-level intervention, which may terminate processes or delay memory reclamation unpredictably. This behaviour explains the increased variability and longer recovery times observed.
These findings demonstrate that RCRD provides a more nuanced understanding of self-healing behaviour by capturing recovery latency directly at the application response level. As cloud-native systems increasingly rely on automated recovery, such time-aware metrics are essential for accurate resilience evaluation.

C. Limitations

Despite demonstrating the applicability of RCRD for evaluating container self-healing behaviour, this

 ©IJSRED: All Rights are Reserved

study has several limitations that should be considered when interpreting the results.
First, all experiments were conducted on a single AWS EC2 instance using a standalone Docker setup. The absence of orchestration layers such as Kubernetes limits the generalizability of the findings to large-scale, multi-node environments.
Second, the evaluation focuses on a single application container (NGINX) with minimal external dependencies. While this simplifies analysis, real-world cloud applications often consist of multiple interdependent services whose recovery behaviour may differ significantly.
Third, only three fault types—CPU stress, process termination, and memory exhaustion—were considered. Other failure scenarios such as network partitions, disk I/O saturation, and configuration errors were not explored.
Finally, RCRD measurements were obtained in a controlled experimental environment and do not account for background workload variability or external traffic fluctuations commonly observed in production systems.

D. Future Work

While this study demonstrates the effectiveness of RCRD as a resilience metric in a single-container Docker environment, several extensions can further enhance the scope and applicability of this work.

- Kubernetes-based Evaluation: Extend RCRD measurement to Kubernetes-managed environments to analyse recovery behaviour under pod restarts, rescheduling, and replica-based self-healing mechanisms.
- Multi-Container and Microservice Architectures: Apply RCRD to interconnected services to capture cascading recovery delays and inter-service dependency effects.
- Network and I/O Fault Injection: Incorporate network latency, packet loss, and disk I/O saturation faults to evaluate resilience beyond CPU and memory failures.
- SLO and User-Perceived Metrics: Correlate RCRD values with Service Level Objectives (SLOs) and user experience indicators to assess practical service reliability.
- Automated Observability Integration: Integrate RCRD measurement with monitoring tools such as Prometheus and Grafana for continuous resilience assessment in production systems.

V. CONCLUSION

This paper presented an experimental evaluation of container self-healing behaviour using the Recovery-Complete Response Delay (RCRD) metric. By conducting controlled fault injections on a Docker-based NGINX container deployed on AWS EC2, the study demonstrated that different fault types exhibit distinct recovery characteristics that are not captured by conventional availability metrics.
The results show that while restart counts often remain unchanged, recovery latency varies significantly across CPU stress, process kill, and out-of-memory faults. In particular, OOM conditions introduce higher variability and longer recovery times, highlighting the importance of measuring recovery completeness rather than restart events alone.
RCRD offers a lightweight, implementation-independent metric that directly reflects user-perceived service recovery. As cloud-native architectures continue to evolve toward increased automation and self-healing, metrics such as RCRD can play a crucial role in evaluating and improving system resilience.
Overall, this work provides a practical foundation for time aware resilience analysis in containerized environments and motivates further research into fine-grained recovery for modern cloud systems.

REFERENCES

[1] D. Merkel, “Docker: Lightweight Linux containers for consistent development and deployment,” *Linux Journal*, vol. 2014, no. 239, 2014.

 ©IJSRED: All Rights are Reserved

[2] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, Omega, and Kubernetes,” *Communications of the ACM*, vol. 59, no. 5, pp. 50–57, 2016.
[3] C. King, “stress-ng: System stress testing,” 2020. [Online]. Available: https://manpages.ubuntu.com
[4] N. Basiri et al., “Chaos engineering,” *IEEE Software*, vol. 33, no. 3, pp. 35–41, 2016.
[5] Z. Chen, M. Goudarzi, and A. N. Toosi, “Resilience evaluation of containerized cloud applications,” *Future Generation Computer Systems*, vol. 128, pp. 1–12, 2022.