International Journal on Advanced Science Engineering Information Technology

Vol.15 (2025) No. 3
ISSN: 2088-5334

AgnosticChaos: A Tool to Assess Software Applications' Reliability

Odai Hussein Ahmed Al-sayaghi [a], Noraini Che Pa [a,*], Hafeez Osman [a], Ainita Ban [a]

[a] *Department of Software Engineering and Information System, Faculty of Computer Science and Information Technology, University Putra Malaysia, Serdang, Selangor, Malaysia*

Corresponding author: [*]norainip@upm.edu.my

Abstract*—Data-intensive software applications that process millions of events per second from IoT sensors need to maintain high availability to deliver continuous data to consumers. Downtime can have significant impacts on service quality, so many cloud vendors, including Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP), provide geo-redundant infrastructure to enhance service reliability. However, despite these provisions, it remains crucial to assess the resilience and reliability of software applications under varied outage and fault scenarios to ensure that they can handle unexpected disruptions. Chaos engineering offers a systematic approach to enhancing this reliability by deliberately introducing controlled failures into systems. This practice enables developers to gain insights into an application's response under stress, ultimately fostering a better understanding of its robustness and identifying areas for improvement. This research introduces AgnosticChaos, a novel tool designed to integrate with Azure's continuous delivery pipelines, enabling the seamless use of multiple cloud vendors and third-party chaos engineering tools. AgnosticChaos provides a streamlined environment for testing applications' resilience and reliability prior to deployment in a production environment. To evaluate its effectiveness, AgnosticChaos was tested on three open-source microservices: an event producer, event receiver, and event retainer. Our findings reveal that AgnosticChaos is not only more efficient and developer-friendly but also offers comparable effectiveness to direct use of third-party chaos engineering tools. This study highlights the value of AgnosticChaos as a vital component in pre-production workflows, offering a comprehensive and adaptable solution for resilience testing across diverse cloud environments.

***Keywords*— IoT; chaos engineering; continuous delivery; reliability.**

Manuscript received 24 Dec. 2024; revised 13 Jan. 2025; accepted 17 Feb. 2025. Date of publication 30 Jun. 2025.

I. INTRODUCTION

The reliability and resilience of data-intensive software applications are crucial for the successful operation of the Internet of Things (IoT), as IoT devices generate partitioned data, encrypting only sensitive portions [1]. IoT applications must efficiently manage and transmit large amounts of data from interconnected devices and software that exchange information with each other and the cloud [2], [3], [4]. Any disruptions in their operation could result in the loss of vital data and cause disturbances in the operation of interconnected devices and systems.

Cloud computing and serverless computing have emerged as powerful tools for building and deploying scalable applications. Cloud computing is an online platform that provides access to applications, hardware, and system software as services, enabling access to software, data, and resources from anywhere. Deployment models are based on resource availability and accessibility [4], [5]. Numerous cloud providers offer geo-redundancy, duplicating data and applications across multiple data centers. Manufacturers can also adopt a multi-cloud strategy, leveraging services from various providers for redundancy, vendor lock-in mitigation, and access to best-in-class services, thereby enabling tailored solutions, cost optimization, and flexibility [7]. Investing in cloud services entails risks, including security and cost issues, as well as concerns about advanced persistent threats targeting cloud systems [8]. Additionally, some providers offer serverless computing, which has effectively addressed numerous concerns and obstacles, including load balancing, manageability, and scalability. Users and customers are relieved of the burden of dealing with these complexities.

However, as mentioned in [9], serverless computing comes with various challenges and issues. A significant issue revolves around the server start-up time for infrequently used applications, as it can have a substantial impact on the application's performance and Quality of Service (QoS). When servers are shut down during idle periods, this can be crucial for ensuring the quality of security services [10]. A study by [11] highlighted that Cloud-based services, like any

other technology, can experience failures or crashes. One example of this is the distributed denial-of-service (DDoS) attacks that affected Amazon's access to information for an extended period in 2009. Additionally, in 2013, there were reports of cloud outages for Amazon, Microsoft, and Google.

Hence, there is a crucial need to continuously assess the reliability and resilience of these software applications to ensure their proper functioning in such failures [12], [13]. Developers use chaos engineering to test a system's resilience by intentionally causing failures directly on a production system, building confidence in its ability to withstand unexpected chaotic conditions that start with known problems and aims to uncover unknown ones [14], [15], [16]. Another study by [17] explained that evaluating a distributed system is necessary to gain confidence in its ability to tolerate chaotic production issues. These conditions may range from hardware failure to an unexpected surge in client requests to an erroneous value in a runtime configuration parameter. Through simulating various failure scenarios and observing the system's response, organizations can improve their understanding of the system's resilience and reliability [18].

This paper introduces AgnosticChaos, a prototype tool that aims to address this limitation by seamlessly integrating Azure Chaos Studio and Chaos Mesh into Azure continuous delivery pipelines. This integration streamlines and automates the evaluation of software applications in a pre-production environment before their deployment to the production stage. By proactively introducing controlled failures into the system, developers can identify and address potential vulnerabilities before they impact production environments. This approach helps to build more robust and resilient applications that can withstand unexpected disruptions, such as hardware failures, network outages, or software bugs. AgnosticChaos strives to make chaos engineering more accessible and practical, enabling organizations to continuously improve the reliability and resilience of their critical software applications.

The subsequent sections of this paper are structured as follows: Section 11 provides an overview of previous research studies and methods. Section III outlines our tool and discusses resilience and reliability in various aspects by evaluating the prototype tool, while Section IV serves as the concluding section of the paper.

II. Materials and Method

Many researchers have proposed various tools and frameworks utilizing chaos engineering to evaluate the reliability and resilience of software applications [19]. A study by [20] proposed ChaosOrca, a system designed to inject chaos engineering failures into a system to evaluate its resilience. ChaosOrca controls the scope of chaotic experiments by utilizing Linux kernel features such as cgroups and namespaces, and interfaces with various monitoring tools to enhance observability. The tool introduces perturbations by introducing faults or delays in system call execution as defined in a resilience experiment.

Other researchers [21] proposed a chaos engineering framework designed based on a review of existing literature and a survey of available tools on the market, with part of it applied in a Swedish grocery store chain. This helped identify opportunities for improvement in the existing systems. In [22], a Risk-driven Fault Injection (RDFI) technique is proposed that utilizes chaos engineering principles to evaluate the security of cloud systems by injecting security faults to identify vulnerabilities and improve resilience.

Another study by [23] noted that chaos engineering can be costly and time-consuming to set up, and it often focuses on technical rather than business-level evaluation. To address these challenges, the authors propose ChaosTwin, a technique that leverages chaos engineering to create virtual representations of physical systems, known as digital twins. ChaosTwin assists service providers in finding cost-effective configurations that mitigate the negative consequences of unexpected occurrences by injecting faults into digital twins and evaluating alternative service configurations and fault management strategies from a business perspective. In [24] and [25], it is described that Netflix built a system called Chaos Automation Platform (CHAP) that conducts chaos engineering experiments within its microservice architecture. The goal of these experiments is to evaluate the overall system's resilience in the event of a service interruption or degradation (e.g., increased response latency or errors). CHAP utilizes the FIT fault injection approach to inject faults at the application level by annotating incoming requests with metadata that indicates the call should fail or be delayed.

Furthermore, regularly performing continuous chaos engineering experiments ensures the ongoing relevance of the evaluation process. Despite the increased difficulty and quality risks involved with dividing development tasks, many companies implement continuous delivery (CD), indicating that the business balances the drawbacks [26]. Numerous research studies emphasize the significance of continuous delivery pipelines, as described by [27], a software engineering process that enables teams to continuously produce valuable software in shorter cycles while ensuring it remains releasable at all times. Adapting continuous integration allows problems to be identified and fixed as they occur, rather than waiting until later stages of the development process. As noted by [28] and [29], Continuous Integration (CI) and Continuous Delivery (CD) have emerged as blessings for conventional application development and release management practices. These practices enable the continuous provision of high-quality artifacts to customers along with ongoing integrated feedback.

The domain of our research study is to assess data-intensive applications that require higher resilience and reliability, which becomes even more critical when operational consumers use the data to make real-time decisions. There are often many challenges and technical issues due to the complex nature of such architecture, the software, physical components, network configuration, operational methods, and data formats [30]. Network computing environments are vulnerable to failures and outages due to the unpredictable nature of network infrastructure [31]. As mentioned by [32], a bottleneck in network bandwidth and communication latency is expected to occur during the processing of IoT data through distant data centers in the cloud.

Another challenge mentioned by [33] is that the transmission and processing of massive volumes of data in cloud computing will increase the burden on the core network and limit the pace of data transmission and processing. Data-intensive applications deployed in the cloud require more scalability and redundancy capabilities as compared to

internal on-prem setups. While cloud vendors offer scalability and high availability because of their ability to replicate resources such as edge cloud computing into multiple geographical locations, according to [34], one issue with replicating resources like edge cloud computing in various locations is that it can lead to strong consistency issues due to the data nodes being in different places, which can cause delays and insufficient bandwidth.

To highlight the difference between fault injection and chaos engineering. In [17], it is mentioned that fault injections are more of a testing technique in which the system is tested against expected behavior or condition, while chaos engineering generates new information or knowledge each time it is conducted. When it comes to faults in software, a fault, resulting from an abnormal state of a system component, can cause an error, leading to the partial or complete failure of a system [34]. As mentioned in [35], the Fault Tolerant Elastic Resource Management (FT-ERM) framework addresses cloud outages by incorporating high availability (HA) awareness into servers and virtual machines (VMs). It utilizes online monitoring to assess server health, creates dedicated High Availability Virtual Networks (HAVNs) for users, and employs a Multi-Input and Multi-Output Evolutionary Neural Network (MIMO-ENN) based predictor to forecast VM failures proactively. According to [36], faults in software can be categorized into six categories, which include: omission, hardware, software, network, response, and miscellaneous.

The research employed a mixed-methods approach, combining theoretical analysis and practical implementation. Initially, a comprehensive literature review was conducted to gain a deep understanding of the challenges and potential solutions related to the reliability and resilience of data-intensive software applications. This review explored the challenges caused by IoT-based applications, such as data volume, latency, and security risks. It also explored how cloud-based solutions, including geo-redundancy, can mitigate these challenges. Furthermore, the review examined the principles and benefits of chaos engineering as a proactive approach to testing system resilience. Finally, the research explored how continuous integration and delivery practices can be integrated with chaos engineering to automate testing and improve deployment frequency.

Subsequently, a tool, AgnosticChaos, was developed and implemented to automate chaos engineering experiments within continuous integration and delivery pipelines. This tool was designed to integrate with cloud vendor APIs, such as Azure Chaos Studio and AWS Fault Injection Simulator, to trigger fault injection experiments. The tool interacted with continuous integration and delivery pipelines, analyzed system responses, and provided insights into the system's resilience. A primary application was developed to serve as a central hub. This application communicates with various fault injection APIs provided by different cloud vendors, enabling the injection of faults and simulation of diverse scenarios within pre-production environments. Complementing the core application, four microservices were developed to simulate and process events streamed from an IoT simulator. These microservices create a realistic testing environment, facilitating the evaluation of IoT applications under various fault conditions. A case study was conducted to evaluate the tool's effectiveness in real-world scenarios, involving developers from multiple industries who assessed the reliability and resilience of their applications using the tool.

The proposed approach centers on DevOps, incorporating supplementary stages depicted in green in Fig. 1. These additional steps aim to ensure that the application is initially deployed in a digital twin environment (pre-production), where we can deliberately introduce faults. This ensures that the applications deployed in the production environment remain undisturbed. As suggested by [23], the utilization of digital twins for infrastructure helps mitigate the risks associated with conducting chaos engineering experiments in a production environment. Following this recommendation, we have integrated digital twins into our proposed approach.

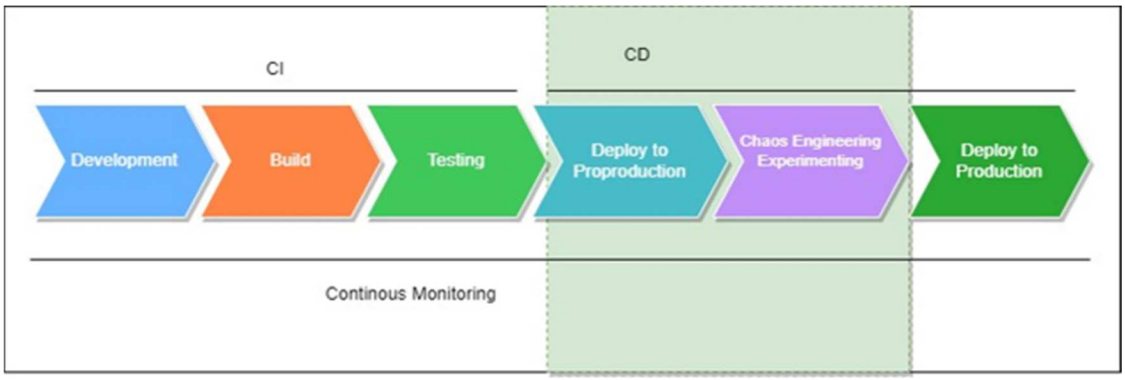

Fig. 1 DevOps with chaos engineering and digital twin environment

The activity diagram in Fig. 2 describes the DevOps process. The process starts when a developer introduces new enhancements or bug fixes to an existing system. Following the code change, a pull request (PR) is generated to undergo code review and simultaneous quality assurance (QA) testing. Concurrently, a build validation is conducted to deploy the application in a digital twin environment (pre-production). This step enables the code reviewer and QA personnel to verify whether the newly added code meets the resilience and reliability expectations.

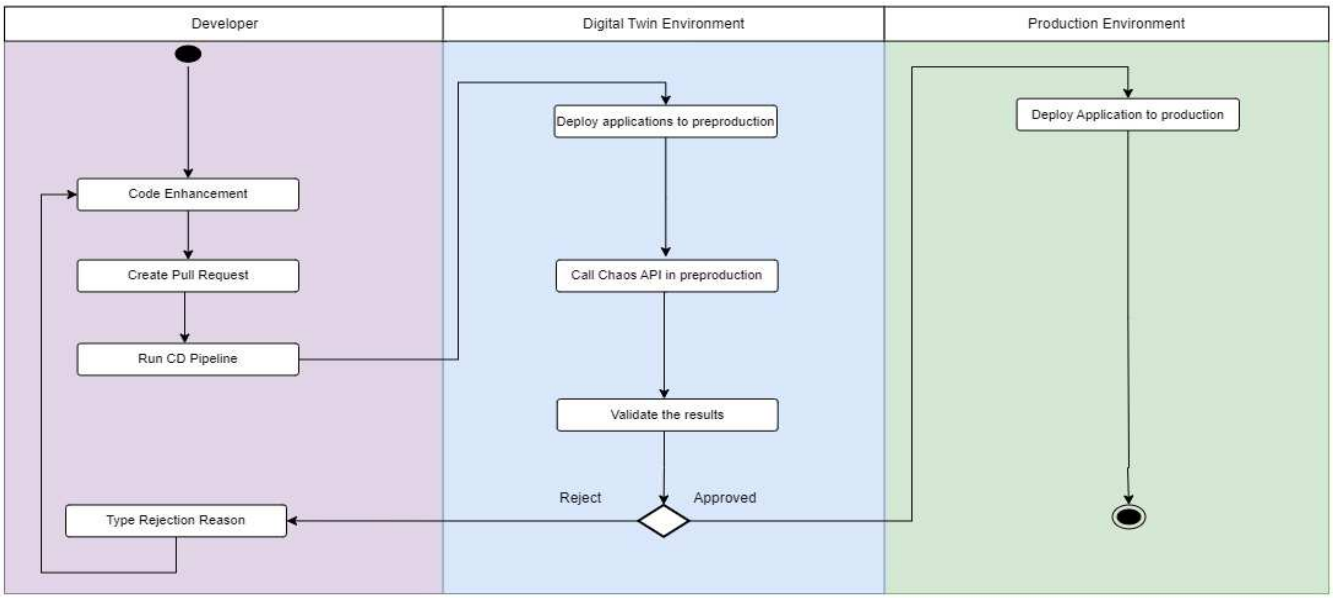

Fig. 2 AgnosticChaos tool architecture

During this phase, the applications are running, and our tool initiates the process by contacting cloud vendors and third-party chaos engineering tools to inject faults. Subsequently, the logs from the application are gathered and presented to the reviewer and QA, ensuring that they are informed about the current state of the application's resilience and reliability. Based on the system's acceptance criteria, it is then their responsibility to approve and deploy the change or reject it.

A. Design and Implementation

We have created a tool called AgnosticChaos that integrates with Azure Chaos Studio and can be extended to integrate with other third-party fault injection APIs. AgnosticChaos will be used in Azure continuous pipelines to inject faults into microservices deployed in Kubernetes. Our tool interacts with the Azure Chaos Studio API using an Azure Service Principal for authentication.

Subsequently, fault injection is initiated through communication between Azure Chaos Studio and separate managed identities associated with each Chaos experiment. These managed identities possess the necessary permissions to inject faults against AKS and Azure Cosmos DB. Our tool was developed using .NET 6, and it is integrated with Application Insights. It can be invoked by either developers or Azure DevOps agents, ensuring seamless integration with other DevOps tools. Fig. 3 illustrates the architecture of the tool, including its dependencies and communication.

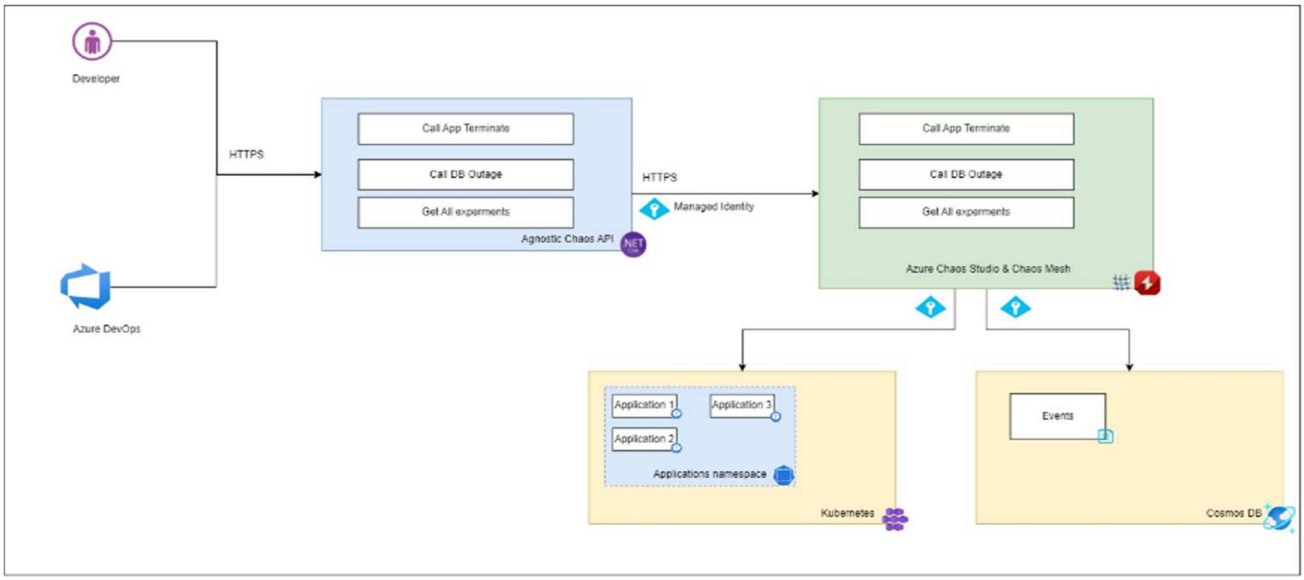

Fig. 3 AgnosticChaos tool architecture

Additionally, the pipeline we have built as a sample will collect errors by communicating with the log store API. Another task will display these error logs and a link to a dashboard that we have built, which shows the CPU and memory usage of the pods deployed in Kubernetes, along with Infrastructure dependencies such as Event Hub and CosmosDB. The environment we have created contains three applications: Event Producer, Event Receiver, and Event

Retainer. These components are derived from an open-source C# solution (https://github.com/denniszielke/resilient-cloud-apps). We have added YAML pipelines to be able to deploy them through Azure DevOps pipelines and we have added another application to trigger the event producer which acts as an IoT simulator.

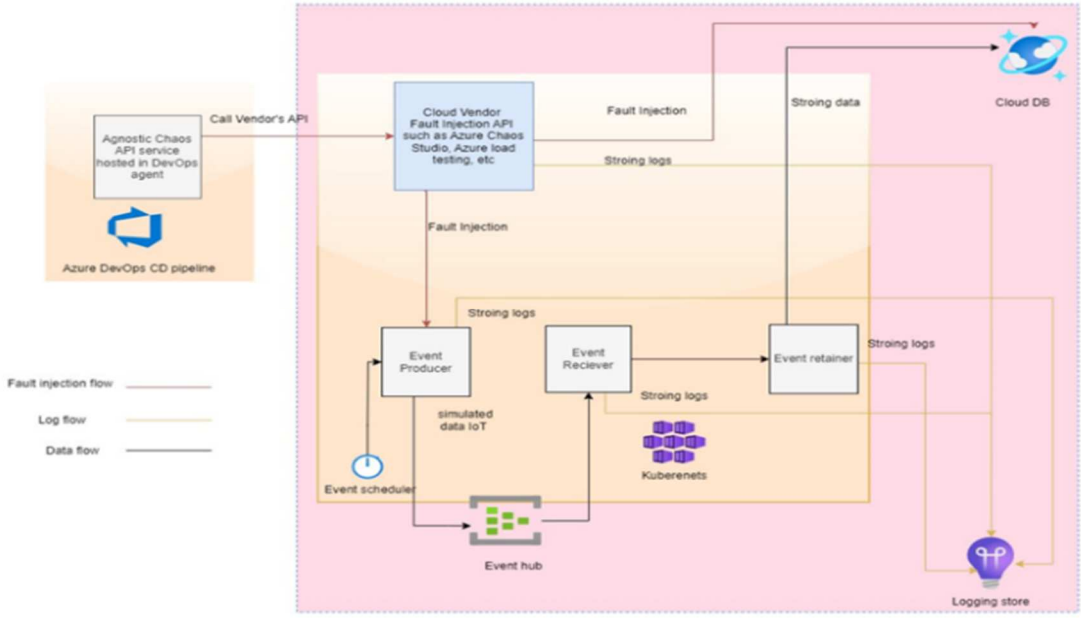

Fig. 4 AgnosticChaos development environment architecture in IoT domain

Fig. 4 provides a clear view of the architecture of the development environment in IoT. The following faults will be injected into the Azure DevOps pipeline using our AgnosticChaos tool in the continuous delivery pipeline after the application is deployed to pre-production. This step will always be part of the process of releasing new bug fixes or some features to production. It is worth noting that Azure continuous delivery pipelines can be triggered at any time or run according to a specific schedule, which further enhances the system by ensuring its resilience and reliability consistently meet the criteria.

The following details explain the different faults that will be injected into the system and the execution order.

1) Application Termination: The fault will simulate the termination of an application using pod kill, which is provided by a library called Chaos Mesh, and Azure Chaos Studio already has integration with it.

2) CPU/Memory Stress: The scope of this fault is to increase the utilization of the CPU and memory to a specific threshold. We will use both Azure Chaos Studio and Chaos Mesh to inject them via our API.

3) Cloud Data Center Outage: The fault will simulate a data center outage for a database (Cosmos DB). This fault originates from Azure Chaos Studio and will be invoked by our tool.

III. Results and Discussion

A case study was carried out to evaluate our tool and approach focusing on three aspects: tool efficiency as compared to using chaos engineering tools with integration to DevOps; usability, to measure how usable and developer friendly is the tool; and tool effectiveness, involving the use of various metrics to evaluate resilience and reliability. Table 1 presents the five developers who evaluated the tool and their corresponding levels of experience. It is essential to note that the evaluation was conducted separately for each developer, utilizing two distinct approaches: the manual approach, where the developer manually executed the experiments through the Azure portal after deploying the applications in the pre-production environment, and our proposed approach, which is automated within the continuous delivery pipelines.

TABLE I
EVALUATORS' DEVELOPMENT EXPERIENCE

Participant	Years of Experience	Experience in Azure DevOps
Developer 1	5	1
Developer 2	8	4
Developer 3	12	3
Developer 4	14	3
Developer 5	9	1

A. Efficiency

To evaluate the efficiency of the AgnosticChaos Tool, two experiments were conducted to test the mentioned open-source application, which comprises three microservices applications. The first experiment utilized our tool, which is integrated into Azure DevOps pipelines. In the second experiment, a manual process was employed, utilizing Azure Chaos Studio and Chaos Mesh directly.

The hypothesis is as follows:

- H_0: *μ AgnosticChaos =μ manual*
- H_a: *μ AgnosticChaos <μ manual*
- *μ AgnosticChaos: AgnosticChaos tool.*
- *μ manual: The compared Approach (Manual).*

Table 2 presents the data collected from participants to measure the time spent using both approaches.

TABLE II
DATA INPUT

Group A (Manual) in minutes	Group B (AgnosticChaos) in minutes
44	24
48	23.12
43	22
50	22
88	23

The critical significance level was set at 0.05. After analyzing and testing the data, a significant difference was observed in the time taken to complete the task using the AgnosticChaos tool (Mean = 22.824) compared to the time taken using the manual approach with Azure Chaos Studio and Chaos Mesh tools (Mean = 54.6). In conclusion, the study presented substantial evidence to reject the null hypotheses and accept the alternative hypotheses. There is strong proof that employing the AgnosticChoas tool enhances efficiency as the time spent is less than utilizing the manual approach with azure chaos studio and chaos mesh. This analysis was conducted to fulfil the requirement of evaluating the proposed tool and provide compelling evidence to support the acceptance of the alternative hypothesis.

B. Usability

To evaluate usability, we used the System Usability Scale (SUS), a measure designed to assess the usability of websites, tools, or interactive systems. For each question, participants are required to select one option from the form. A SUS score equal to or greater than 68 is considered above average, while scores below this threshold are considered below standard. The scores for each question are converted to a new numerical value, summed up, and then multiplied by 2.5 (www.usability.gov). Table 3 shows the results of this evaluation for the two groups.

TABLE III
SUS DATA RESULTS

No.	Group A (Manual)	Group B (Agnostic Chaos)
1	10	85
2	15	85
3	20	92.5
4	52.5	90
5	45	55
Percentage (%)	28.50	81.50

C. Effectiveness

For our assessment, we employed resilience and reliability metrics to gauge the effectiveness of the tool. As a metric, we utilized the Mean Time Between Failures (MTBF), which represents the expected duration between successive failures of a system during its regular operation. MTBF is typically determined by calculating the average time between failures of a system [37]. On the other hand, we will also use Mean Time to Recovery (MTTR), which indicates the expected time until a system recovers. The MTBF and MTTR can be calculated by applying the equations below, respectively. These equations state MTTF and MTTR as the averages of uptime and downtime, respectively. Where nf is the total number of failures, and the variables DT and UT denote service downtime and uptime, respectively.

$$MTBF = \frac{\sum_{i=1}^{M} UT_i}{nf} \quad [38] \tag{1}$$

$$MTTR = \frac{\sum_{i=1}^{M} DT_i}{nf} \quad [39] \tag{2}$$

Upon analyzing the gathered data illustrated in Table 4, we observed that both approaches yielded identical results. In essence, our tool demonstrated the same level of effectiveness as the manual approach, as both methods injected the same number of faults using the vendor API, resulting in comparable values.

TABLE IV
MTTR AND MTBF RESULTS

No of faults nf	Estimated Downtime in minutes DT	Estimated Uptime in Minutes UT	MTBF in minutes	MTTR in minutes
2	5	24	12	2.5
2	5	23.12	11.56	2.5
2	5	22	11	2.5
2	5	22	11	2.5
2	5	23	11.5	2.5

The above results demonstrate the efficiency, usability, and effectiveness of our proposed tool and approach in the continuous delivery process as compared to using the manual approach, in which developers directly use chaos engineering tools without integrating them into continuous delivery pipelines. These results were analyzed quantitatively through experimental analysis. We downloaded and evaluated three IoT applications (Message creator, message receiver, and message retainer) from "GitHub" using our tool, AgnosticChaos. These applications were deployed in Kubernetes through Azure DevOps pipelines. Our tool played a vital role in facilitating communication between Azure DevOps and Azure Chaos Studio, functioning as a centralized solution that abstracts the developer from using cloud-specific fault injection.

This unification simplifies the process of injecting faults and employing chaos engineering in the DevOps workflow, making it more efficient and developer-friendly. Although our current tool supports only Azure Chaos Studio, it can be extended to communicate with other cloud vendors and third-party chaos engineering tools, offering increased efficiency and usability. This will help developers save time and continuously gain insights into the resilience and reliability of the software applications. The current tool is valuable for evaluating the resilience and reliability of software applications. However, it is limited to assessing applications deployed exclusively in Kubernetes environments, thereby excluding the evaluation of locally running applications. While Kubernetes is commonly used for production environments, developers may also need to assess application behavior on local development machines without a local Kubernetes instance. This limitation hinders its use in early development stages or non-Kubernetes setups. It is essential to highlight that our evaluation process involves only five developers and three applications. To gain more insights into the efficiency, effectiveness, and usability of this approach, it would be beneficial to extend the evaluation to include

developers from different backgrounds and applications from various domains. This would enhance the robustness of the findings and provide a more comprehensive understanding of the approach's capabilities.

IV. Conclusion

This research has developed a tool and proposed an innovative approach that facilitates the comprehensive evaluation of the resilience and reliability of data-intensive software applications. By seamlessly integrating chaos engineering techniques with multiple cloud vendors and third-party fault injection APIs, this tool is directly incorporated into continuous delivery pipelines, enabling continuous and automated testing of application resilience. This integration provides developers with an invaluable resource to assess how their applications respond to simulated faults, disruptions, and outage scenarios before deployment in a production environment. The tool not only aids in identifying vulnerabilities within the software architecture but also enables developers to proactively address potential weaknesses, thereby enhancing the system's overall robustness.

Furthermore, this research study lays the groundwork for developing an adaptable tool that empowers developers to assess the reliability of distributed and geo-redundant applications operating across multi-cloud environments. This contribution is significant for applications that process high volumes of data in real-time, where downtime or failure can have significant implications. By ensuring that applications are rigorously tested for resilience, this research promotes a higher standard of software quality, reliability, and robustness, ultimately benefiting both developers and end-users. The methodologies provided by this study pave the way for further advancements in automated resilience testing, supporting the continuous improvement of cloud-native and data-intensive software applications in an increasingly interconnected digital landscape.

To achieve this, a mixed-methods approach was employed, combining theoretical analysis and practical implementation. Initially, a comprehensive literature review was conducted to gain a deep understanding of the challenges and potential solutions related to the reliability and resilience of data-intensive software applications. Subsequently, a tool, AgnosticChaos, was developed and implemented to automate chaos engineering experiments within continuous integration and delivery pipelines. This tool was designed to integrate with cloud vendor APIs, such as Azure Chaos Studio and AWS Fault Injection Simulator, to trigger fault injection experiments. The tool's architecture was designed to interact with continuous integration and delivery pipelines, analyze system responses, and provide insights into the system's resilience. A case study was conducted to evaluate the tool's effectiveness in real-world scenarios, involving developers from various industries who assessed the reliability and resilience of their applications using the tool. By combining these theoretical and practical aspects, the research aimed to advance chaos engineering practices and improve the reliability and resilience of data-intensive software applications.

Acknowledgment

This research was supported by the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia.

References

[1] R. Gupta et al., "An IoT-centric data protection method for preserving security and privacy in cloud," *IEEE Syst. J.*, vol. 17, no. 2, pp. 2445-2454, Jun. 2023, doi: 10.1109/jsyst.2022.3218894.\

[2] H. A. A. Hassan and M. Zolfy, "Exploring lightweight deep learning techniques for intrusion detection systems in IoT networks: A survey," *J. Electr. Syst.*, pp. 1944-1958, 2024, doi: 10.52783/jes.2292.

[3] M. Kokila and K. Srinivasa Reddy, "Authentication, access control and scalability models in Internet of Things security-A review," *Cybersecur. Appl.*, vol. 3, pp. 1-18, 2025, doi:10.1016/j.csa.2024.100057.

[4] S. Pal, A. Dorri, and R. Jurdak, "Blockchain for IoT access control: Recent trends and future research directions," *J. Netw. Comput. Appl.*, vol. 203, pp. 1-19, 2022, doi: 10.1016/j.jnca.2022.103371.

[5] C. M. Mohammed and S. R. M. Zeebaree, "Sufficient comparison among cloud computing services: IaaS, PaaS, and SaaS: A review," *Int. J. Sci. Bus.*, vol. 5, no. 2, pp. 17-30, 2021.

[6] G. Saini and N. Kaur, "Information leakage techniques in cloud computing: A review," in *Proc. ICTACS*, Tashkent, Uzbekistan, 2022, pp. 327-334, doi: 10.1109/ictacs56270.2022.9988405.

[7] U. S. Umar and M. E. Rana, "Cloud revolution in manufacturing: Exploring benefits, applications, and challenges in the era of digital transformation," in *Proc. ICETSIS*, Manama, Bahrain, 2024, pp. 1890-1897, doi: 10.1109/icetsis61505.2024.10459473.

[8] R. Islam et al., "The future of cloud computing: Benefits and challenges," *Int. J. Commun. Netw. Syst. Sci.*, vol. 16, no. 4, pp. 53-65, Apr. 2023, doi: 10.4236/ijcns.2023.164004.

[9] S. S. Gill et al., "Transformative effects of IoT, Blockchain and Artificial Intelligence on cloud computing: Evolution, vision, trends and open challenges," *Internet Things*, vol. 8, pp. 1-26, 2019, doi:10.1016/j.iot.2019.100118.

[10] L. Zhang, J. Ron, B. Baudry, and M. Monperrus, "Chaos engineering of ethereum blockchain clients," *Distrib. Ledger Technol. Res. Pract.*, vol. 2, no. 3, pp. 1-18, 2023, doi: 10.1145/3611649.

[11] A. Tchernykh et al., "Towards understanding uncertainty in cloud computing with risks of confidentiality, integrity, and availability," *J. Comput. Sci.*, vol. 36, no. 4, pp. 1-9, 2019, doi:10.1016/j.jocs.2016.11.011.

[12] M. Akour, M. Alenezi, and O. Alqasem, "Enhancing software fault detection with deep reinforcement learning: A Q-learning approach," in *Proc. ACM Int. Conf. Proc. Ser.*, 2024, pp. 97-101, doi:10.1145/3651781.3651796.

[13] C. Jiang et al., "A hybrid computing framework for risk-oriented reliability analysis in dynamic PSA context: A case study," *Qual. Reliab. Eng.*, vol. 39, no. 8, pp. 3445-3471, 2023, doi:10.1002/qre.3196.

[14] M. Jaival, K. Markaym, and A. Kaplan, "Serverless cloud functions - Opportunity in chaos," in *Proc. CSCI*, Las Vegas, NV, USA, 2022, pp. 1330-1335, doi: 10.1109/csci58124.2022.00239.

[15] G. Chen et al., "Big data system testing method based on chaos engineering," in *Proc. ICEIEC*, Beijing, China, 2022, pp. 210-215, doi: 10.1109/iceiec54567.2022.9835072.

[16] S. Sharieh and A. Ferworn, "Securing APIs and chaos engineering," in *Proc. ESSCA*, 2021, pp. 290-294, doi:10.1109/cns53000.2021.9705049.

[17] A. Basiri et al., "Chaos engineering," *IEEE Softw.*, vol. 33, no. 3, pp. 35-41, May-Jun. 2016, doi: 10.1109/ms.2016.60.

[18] A. A. Z. Ibrahim et al., "Reliability-aware swarm based multi-objective optimization for controller placement in distributed SDN architecture," *Digit. Commun. Netw.*, vol. 10, no. 5, pp. 1245-1257, 2024, doi: 10.1016/j.dcan.2023.11.007.

[19] M. Verma et al., "A chaos recommendation tool for reliability testing in large-scale cloud-native systems," in *Proc. COMSNETS*, 2024, pp. 270-272, doi: 10.1109/comsnets59351.2024.10427311.

[20] J. Simonsson et al., "Observability and chaos engineering on system calls for containerized applications in Docker," *Future Gener. Comput. Syst.*, vol. 122, pp. 117-129, 2021, doi:10.1016/j.future.2021.04.001.

[21] H. Jernberg, P. Runeson, and E. Engström, "Getting started with chaos engineering - Design of an implementation framework in practice," in *Proc. ESEM*, Bari, Italy, 2020, pp. 1-10, doi:10.1145/3382494.3421464.

[22] K. A. Torkura et al., "CloudStrike: Chaos engineering for security and resiliency in cloud infrastructure," *IEEE Access*, vol. 8, pp. 123044-123060, 2020, doi: 10.1109/access.2020.3007338.

[23] F. Poltronieri, M. Tortonesi, and C. Stefanelli, "ChaosTwin: A chaos engineering and digital twin approach for the design of resilient IT services," in *Proc. CNSM*, 2021, pp. 234-238, doi:10.23919/cnsm52442.2021.9615519.

[24] A. Basiri et al., "Automating chaos experiments in production," in *Proc. ICSE-SEIP*, Montreal, QC, Canada, 2019, pp. 31-40, doi:10.1109/icse-seip.2019.00012.

[25] A. Blohowiak et al., "A platform for automating chaos experiments," in *Proc. ISSREW*, 2016, pp. 5-8, doi: 10.1109/issrew.2016.52.

[26] M. Ohshima and N. Uchihira, "Mechanisms for improving investment efficiency through continuous delivery in internet services," in *Proc. PICMET*, 2023, pp. 1-6, doi:10.23919/picmet59654.2023.10216798.

[27] L. Chen, "Continuous delivery: Huge benefits, but challenges too," *IEEE Softw.*, vol. 32, no. 2, pp. 50-54, 2015, doi:10.1109/MS.2015.27.

[28] M. Soni, "End to end automation on cloud with build pipeline: The case for DevOps in insurance industry, continuous integration, continuous testing, and continuous delivery," in *Proc. CCEM*, 2015, pp. 85-89, doi: 10.1109/ccem.2015.29.

[29] S. Afaneh et al., "Security challenges review in agile and DevOps practices," in *Proc. ICIT*, 2023, pp. 102-107, doi:10.1109/icit58056.2023.10226018.

[30] S. Shawki et al., "Healthcare monitoring system for automatic database management using mobile application in IoT environment," *Bull. Electr. Eng. Inform.*, vol. 12, no. 2, pp. 1055-1068, Apr. 2023, doi: 10.11591/eei.v12i2.4282.

[31] M. S. Almhanna et al., "Customizing the minimum number of replicas for achieving fault tolerance in a cloud/grid environment," *Bull. Electr. Eng. Inform.*, vol. 13, no. 1, pp. 396-404, 2024, doi:10.11591/eei.v13i1.5413.

[32] G. Premsankar, M. Di Francesco, and T. Taleb, "Edge computing for the Internet of Things: A case study," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1275-1284, 2018, doi: 10.1109/jiot.2018.2805263.

[33] L. M. Song, M. Zhang, and Y. Luo, "Effective replica management for improving reliability and availability in edge-cloud computing environment," *J. Parallel Distrib. Comput.*, vol. 143, pp. 107-128, 2020, doi: 10.1016/j.jpdc.2020.04.012.

[34] S. Isukapalli and S. N. Srirama, "A systematic survey on fault-tolerant solutions for distributed data analytics: Taxonomy, comparison, and future directions," *Comput. Sci. Rev.*, vol. 53, pp. 1-25, Aug. 2024, doi:10.1016/j.cosrev.2024.100660.

[35] D. Saxena et al., "A fault tolerant elastic resource management framework toward high availability of cloud services," *IEEE Trans. Netw. Serv. Manag.*, vol. 19, no. 3, pp. 3048-3061, Sep. 2022, doi:10.1109/tnsm.2022.3170379.

[36] P. Kumari and P. Kaur, "A survey of fault tolerance in cloud computing," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 33, no. 10, pp. 1159-1176, 2021, doi: 10.1016/j.jksuci.2018.09.021.

[37] A. Gupta, V. Chandra, and A. Dixit, "Reliability analysis of a fault-tolerant full-duplex optical wireless communication transceiver," *IEEE Access*, vol. 11, pp. 61298-61312, 2023, doi:10.1109/access.2023.3287335.

[38] H. Adamu et al., "An approach to failure prediction in a cloud based environment," in *Proc. FiCloud*, 2017, pp. 191-197, doi:10.1109/ficloud.2017.56.

[39] Q. Lin et al., "Predicting node failure in cloud service systems," in *Proc. ESEC/FSE*, 2018, pp. 480-490, doi:10.1145/3236024.3236060.