ISSN: 2583-9888
DOI: doi.org/10.51219/JAIMLD/anila-gogineni/477

URF PUBLISHERS
connect with research world

Journal of Artificial Intelligence, Machine Learning and Data Science

https://urfpublishers.com/journal/artificial-intelligence

Vol: 3 & Iss: 1

Research Article

Chaos Engineering in the Cloud-Native Era: Evaluating Distributed AI Model Resilience on Kubernetes

Anila Gogineni*

Citation: Gogineni A. Chaos Engineering in the Cloud-Native Era: Evaluating Distributed AI Model Resilience on Kubernetes. *J Artif Intell Mach Learn & Data Sci 2025*, 3(1), 2182-2187. DOI: doi.org/10.51219/JAIMLD/anila-gogineni/477

Received: 02 January, 2025; **Accepted:** 18 January, 2025; **Published:** 20 January, 2025

***Corresponding author:** Anila Gogineni, Independent Researcher, USA, E-mail: anila.ssn@gmail.com

ABSTRACT

In this study, participants assigned the role of a cloud-native system-operator will apply chaos engineering practices on Kubernetes clusters to assess the operation of distributed AI models' resilience. Chaos engineering is the most vital technique when it comes to testing the failure conditions of AI systems and their vulnerabilities. The study aims at analyzing how the concept of chaos engineering can be applied, specifically in Kubernetes environments and mainly in terms of fault tolerance and performance recovery. This study will explore how Kubernetes can enable the daily operations of AI applications at scale by performing multiple failure simulations and utilizing automated tools. It also outlines implements, techniques and approaches to use when performing chaos experiments in the context of this discussion.

Keywords: Chaos engineering, Kubernetes, Cloud-native, Distributed AI models, Resilience, Fault tolerance, Kubernetes clusters, AI Reliability and Automation

1. Introduction

Cloud-native era has significantly transformed the development, deployment and management of applications and services. Container orchestration has quickly evolved as a mainstream part of cloud computing, with Kubernetes as the leading platform. Many of the modern AI workloads, which operate under distributed models, execute in Kubernetes-based environments to leverage the flexibility and orchestration offered by cloud-native solutions. Nonetheless, making these AI models robust under failure conditions is going to be a challenging task. This paper is centered on harnessing chaos engineering as a means of testing the robustness of distributed AI models running on Kubernetes.

Chaos engineering is an emerging technique for testing reliability and robustness of cloud-native systems where controlled experiments are performed to elucidate system behavior under failure. The idea behind purposefully creating failures is to look at how the system responds and what improvements can be made to it by engineers. This paper examines the process and effects of using chaos engineering in order to assess and improve the robustness of AI models deployed on Kubernetes platforms.

The paper discusses the following core concepts:

- Understanding Kubernetes and its relevance to AI models.
- The fundamentals of chaos engineering and its application in cloud-native environments.
- Techniques for evaluating the resilience of distributed AI models.
- Tools and frameworks available for chaos engineering in Kubernetes.

2. Key Concepts

2.1. Kubernetes and distributed AI models

Kubernetes can be called the basics of Cloud Native environments, in particular, for managing distributed systems,

such as AI models. It has highly efficient orchestration capabilities, including automatic scaling, load balancing and container management that are crucial to AI applications as they can quickly change the needed resource consumption. For the large datasets, the resource management for AI model deployment plays an important role in ensuring that the utilized system remains performant[5]. Kubernetes do this by providing a way of managing containers that allows AI systems to be deployed, monitored and scaled easily.

A distributed AI model is aimed to split dataset or computational work-whether it is the training or the inference-across different nodes of the Kubernetes cluster. This parallelization increases performance because various computational elements can process subsets of the data at the same time[15]. As AI systems grow more complex these days, from shallow machine learning to deep learning networks, distributed computing offers optimal solutions to the volume of data and computation needed. Kubernetes helps to manage this parallel processing environment by maintaining resource isolation through containers and state between them.

For instance, Kubernetes permits the distribution of models that can incorporate several services or facets, such as data cleaning, model building, model prediction and so more, where all of them will be in the form of pods. These pods are deployed across different worker nodes of the cluster[2]. In case a pod crashes, Kubernetes swiftly reschedules its execution in a different node, to enhance availability. This capability means that the system does not shut down often, which is important for AI systems that may be used for mission-critical applications or for client 24/7 applications.

Pod and Node Resilience in the Kubernetes architecture, containers can be grouped together into a pod, with one or more containers allowed per pod. These pods are spread across several worker nodes in a Kubernetes cluster, nodes which may be physical or virtual hosts. Due to the node level resilience in Kubernetes, the system provides mechanisms for handling failure of individual nodes by rebalancing the pod workloads on healthy nodes in the cluster. Pod-level resiliency is an important feature that allows the containers residing in a pod to be automatically restarted or migrated if they get stuck **(Figure 1)**.

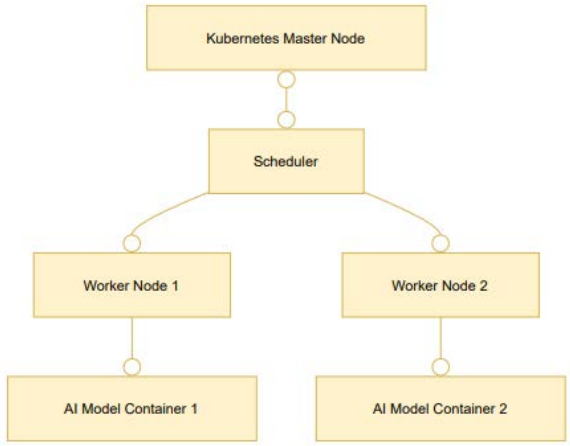

Figure 1: A Kubernetes cluster setup with AI models distributed across pods and nodes.

Kubernetes' flexible features can help AI models to scale up or down and recover effectively from failures, which is crucial when deploying large-scale, distributed systems in production[1].

1.2. Chaos engineering in cloud-native systems

Chaos Engineering is designed to introduce failures into a system to identify its robustness and determine how well it can respond to different drastically conditions. In particular, through the implementation of chaos engineering organizations can discover the vulnerability of their infrastructure and guarantee that their systems are stable in failure conditions. The cloud-native environment poses some problems, most especially when working with distributed systems, for instance, Kubernetes[14]. In this context, chaos engineering can be used to perform a variety of failure tests and to identify how the system is affected by them. This helps in guaranteeing that functions such as Cloud Native Applications, AI models running in Kubernetes clusters are well constructed to include available options for failure, flexibility for expansion, as well as recovery **(Figure 2)**.

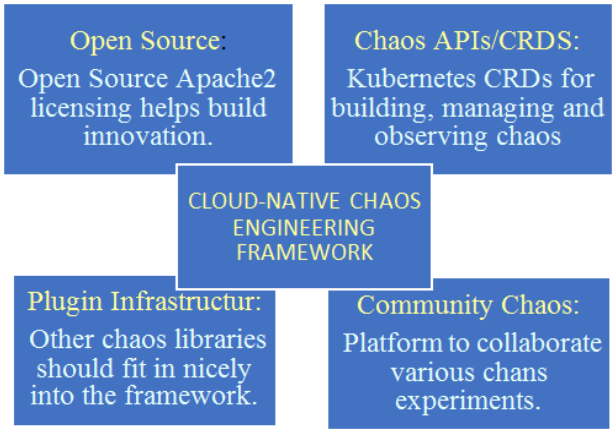

Figure 2: Chaos Engineering in Cloud-Native System Framework.

Chaos engineering experiments can occur at the pod level, the node level and even the networking level in the Kubernetes environment. The purpose is to recreate realistic situations where failures are bound to occur and determine if the framework can detect the failure and continue to deliver the expected level of service in the absence of human intervention. The primary scenarios tested include:

1.2.1. Pod failures: Pods are the smallest objects of deployment in the Kubernetes environment and can consist of one or multiple containers. In actual chaos engineering, pod failures can be emulated through the killing or removal of the pods that host the AI model containers. This will prove that an attempt is made to identify the failure and place the pod on another node while the continued service is not affected. This experiment is especially beneficial for distributed AI systems that need to be always ON and self-healing at the same time. In the case when a pod fails, Kubernetes will need to reschedule this pod without interrupting model inference and training tasks and with minimal impact on the performance.

1.2.2. Node failures: A node failure is a bigger failure where an entire Kubernetes worker node is down and it may take down several pods. Through various emulation scenarios such as node failure, engineers are able to determine if the system is capable of recovering from failure by shifting workload to other more healthy nodes in the cluster[16]. Cooperative AI is designed to apply to the failure of important sub-tasks in deep learning like data pre-processing, model training and model inference for

the distributed AI models. While Kubernetes redistributes pods automatically, chaos engineering enables the teams to determine whether this process is fast enough, given the AI models' need to process requests in real time.

1.2.3. Network latency: Another potential failure mode is network latency, which is closely related to microservices or distributed architecture of AI models. Adding pauses between nodes or pods mimics a network overload or bad conditions of the network, which can dramatically impact the speed of communication between the components of the model. This experiment also allows testing how the system performs with slower communication, for example, whether the load is balanced more efficiently now or the system tries to minimize the usage of pod-to-pod communication which is not optimal in this case.

To manage and control these chaos experiments tools like Chaos mesh and Gremlin are used quite often in Kubernetes environments[17]. These tools offer strong functioning features for introducing fault, failure infusion and observation of the system's reaction in actual times.

Chaos Mesh is an open-source software for chaos engineering that works and coordinates well with the Kubernetes platform. It enables people to perform some failure scenarios like pod failures, network failures, CPU overload etc all of which are weak points in cloud-native applications[13]. Chaos Mesh is something where, for example, users can control failure cases using YAML configuration files and perform various chaos tests inside a Kubernetes infrastructure without additional operations. It also offers metrics, logs and real-time statistics for capturing the results of the experiment and verifying the efficiency of the recovery processes.

Nevertheless, Gremlin is an enterprise-grade, chaos engineering tool that can also work with Kubernetes to perform more complex tests. Following the control plane details, Gremlin has the possibility of testing latency, CPU spikes, container failures and memory leaks[12]. Gremlin is perfect for organizations that want to practice chaos engineering at scale, integrating extensive features for focusing on particular workloads, software application monitoring during tests and studying the effects of failures on distributed structures, such as AI models.

Both of these tools help to ensure that the Kubernetes managed AI systems can perform and remain stable even under stress or failure scenarios which is crucial for keeping AI models running in production.

Pseudocode Example:

```
# Pseudocode: Simulating network latency in Kubernetes using Chaos Mesh
Function simulate_network_latency():
 Initialize chaos_mesh as new ChaosMesh()
 chaos_mesh.create_experiment('network-latency',
 parameters:
 target = 'pod'
 latency = '500ms'
 duration = '60s'
 )
Call simulate_network_latency()
```

2. Evaluating AI Model Resilience

The quality of the distributed AI models is an outstanding attribute that defines the performance and availability of the system in case some attacks happen in the future. In Kubernetes-managed AI systems, resilience is not restricted to recovering from failures but is equally the ability to provide conclusive results in and around structures[3]. AI model robustness requires checking the AI system's ability to be relatively robust to failure modes and maintain the ability to quickly recover from failures while still generating accurate predictions with low latency.

The following key metrics are commonly used to measure the resilience of AI models:

2.1. Availability

Availability is defined as the time that AI models can remain active after a failure has happened. A robust AI solution should be able to handle faults including node failure or pod shut down while continuing to operate without many disruptions to service. Ideally, the system should be self-healing so that it can easily switch to healthy nodes or pods without manual manipulation. For instance, if a pod that is running an AI inference container has crashed, Kubernetes should transfer that pod to another node in the cluster and make sure that the model is available to be on demand to users.

2.2. Consistency

Consistency implies that after the system has come back from the failure, the AI model should be able to provide the right predictions or results[4]. It is crucial to be consistent across containers and nodes when the internal state of an AI model must remain intact (e.g., model weights or training progress). To measure consistency, checkpointing or rolling update can be performed on AI models during the chaos experiment wherein state of the model is captured before and after a failure, respectively, to ensure that the predictions are still accurate at that point.

2.3. Performance

Performance involves monitoring the **(Figure 3),**

Response time and throughput of the AI model during failure events like congestion or intractable algorithms[11]. It should not be affected by latencies in the network or failures of nodes or conflicts in resources in terms of response time and number of successful transactions it can handle concurrently. For instance, during chaos experiments when simulating network latency or CPU spikes, the AI model should not considerably slow down. A performance test uses actual data in normal environmental conditions to establish the efficiency baseline; then it introduces disturbances to the functional model to compare the results.

A resilience test should be done using chaos experiments, along with the use of the monitoring tools that would display the latency, throughput, errors and system integrity in real-time[18]. These tools enable the engineers to determine the influence of faults on the model and the ability of the model to self-correct without further assistance.

The following flowchart outlines the general process of conducting a resilience test for distributed AI models in Kubernetes:

During the Chaos Experiment phase, different failure

scenarios like pod eviction, node failure, network latency etc are injected into the Kubernetes environment.

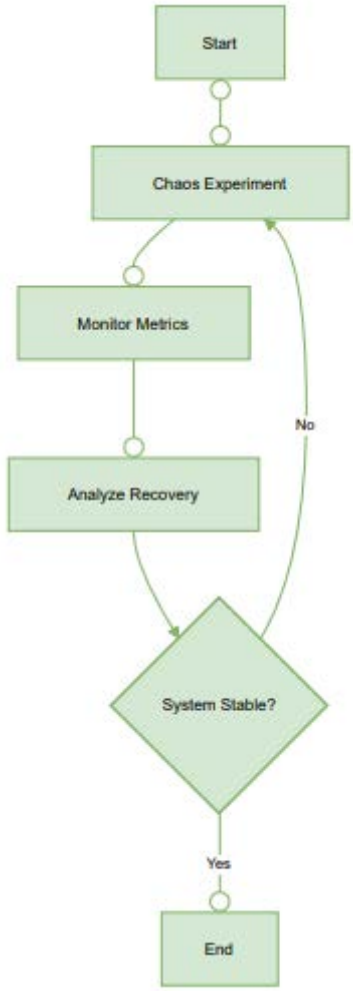

Figure 3: Flowchart of Resilience Testing for Distributed AI Models.

Monitoring Metrics involve gathering of data like response time, errors and load during chaos experiments.

Analyze Recovery investigates if the AI model can return to normal operation after the failure and its efficiency[10].

A System Stable is a checkpoint wherein the stability and robustness of the created model is assessed. In the case that the model comes across all the recovery checks, then it is considered to be stable.

3. Tools and Frameworks

Some of the tools and frameworks with which chaos engineering practices can be performed specifically in the Kubernetes environment includes[7]. These tools facilitate the automation of fault injection, monitoring and analysis, ensuring that the system's resilience can be tested at scale:

3.1. Chaos mesh

Chaos Mesh is an open-source chaos engineering platform particularly designed for integration with the Kubernetes environment. Chaos Mesh is able to perform a number of experiments, including network unavailability, CPU storms or out of memory processes. it lets users describe chaos scenarios in YAML files and apply it on the Kubernetes environments. Chaos Mesh includes metric collection and visualization on the dashboard so that it is possible to monitor the AI model's performance during disruption.

3.2. Gremlin

A commercial chaos engineering platform that offers some sophisticated features on top of the ones explained above for clouds. In this way, through integration with Kubernetes, Gremlin lets engineers cause latencies, container crashes, exhausted resources and more[6]. Gremlin has extreme flexibility in terms of chaos experiments in that users can manipulate individual nodes or pods and inject various fault modes.

3.3. Kube-monkey

Kube-monkey is an application that terminates Kubernetes pods randomly to check the state of the system after a failure. Kube-monkey, further, shuts pods randomly within a Kubernetes cluster and thereby checks the resilience of the cluster's self-healing process of enlightening the pods to healthy nodes, thus ensuring the availability of the AI model in case of random pod failures.

4. Case Study: Chaos Engineering in AI Model Deployment

For contextual purposes and to emphasize chaos engineering as an effective approach to the validation and testing of AI models, we will describe an AI-based recommendation system that was deployed on Kubernetes. The highly variable AI model that makes recommendations based on a user's data was containerized and deployed on to multiple nodes in a Kubernetes cluster. The following chaos experiments were conducted:

4.1. Network interruptions

To further mimic conditions of excessive network connectivity between pods of containers, the recommendation model was subsequently exposed to network latency. The system proved responsive in that it transferred the load from unhealthy pods to healthy pods and proved capable of adapting to shortened latency.

4.2. Node failures

Some nodes were intentionally failed to measure the performance of the system's recovery and continuity in the case of infrastructure outages. When a node was not reachable any longer, Kubernetes alerts the failure and started the management of the rescheduling of the AI model containers in healthy nodes within the cluster. This self-healing mechanism would make sure the system could recover on its own, meaning that the availability of the service is high. When dealing with the recovery process, the AI model worked efficiently as expected without a significant decline in its effectiveness, plus, it possessed fault tolerance **(Figure 4)**.

4.3. High CPU Usage

To simulate resource contention, stress was specifically applied to the CPU components of the Kubernetes cluster on which the AI model was deployed. This mimicked a situation where one or more processes or tasks required a large number of CPU cycles which may have had an effect on the AI model built. However, the results show that the incorporation of new data parameters does not hinder the performance of the AI model, as it showed flexibility to turn down processing power while persistently providing predictions. The relatively low latency

values and the model's stability during the simulation during CPU load confirmed the idea of the system optimizing resource-constrained conditions. Such resource management features in Kubernetes like CPU limits and requests were able to control these stress cases without affecting the availability of services and keeping the AI model running under pressure.

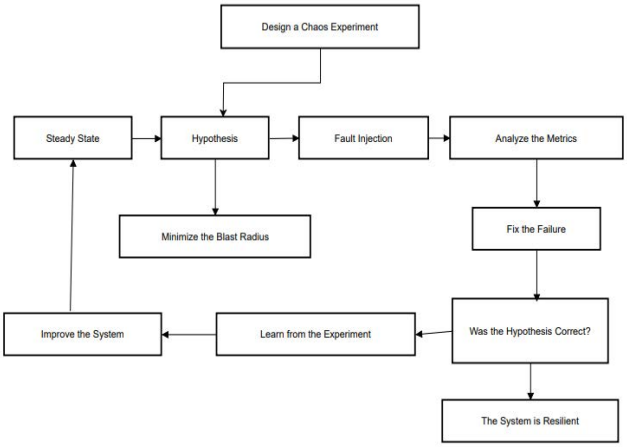

Figure 4: Chaos Engineering Works.

5. Observations

The AI-based recommendation system worked robustly during chaos experiments, ensuring it remained available and did not degrade in terms of functionality even when multiple failure scenarios were encountered. In this particular simulation, the various conditions such as network interference, node failure and high CPU loading did not drastically affect the overall standard of service provision. Each time the model produced invalid values, it returned to abnormal state rapidly, thus guaranteeing that it was coming up with values as expected by the system.

The load balance that is inherent in the Kubernetes platform was particularly impactful in its ability to re-balance workloads across healthy pods, in order to mitigate any service disruptions. The pod rescheduling improves the tolerance of the system for failure by enabling the replacement of the failed containers through automation at a fast rate. Real-time adjustments of the system also helped to maintain the continuity of the uninterrupted work of the end-users during stress conditions. This shows the usefulness of Kubernetes in managing distributed AI models in a reliable and efficient manner.

6. Conclusion

The purpose of this paper is to put emphasis on the importance of chaos engineering for assessment and enhancement of fault tolerance of distributed AI models in the Kubernetes application environment. The primary advantage of incorporating chaos experiments into the continuous delivery pipeline is in revealing areas of vulnerability and regression in such applications as those using artificial intelligence. By actively introducing faults like node failures, network latency and accidents like container crashes, it is possible to test fault tolerance, availability, as well as performance while in a disrupted state. It makes it possible to detect areas in the infrastructure that would not be seen if an actual failure were to happen and not before.

In Chaos Engineering, AI models can be exercised how they hold state across multiple scenarios and how they respond to infrastructure failure. This increases the reliability of AI systems by avoiding issues such as system halts, data disparities and degradation of performance in the production environment, which is a major issue to companies that depend on AI-based services.

Future work should explore the enhancement of automated recovery mechanisms and how chaos engineering practices can be expanded from infrastructure reliability to other areas related to AI systems in cloud native platforms.

7. References

1. Almaraz-Rivera JG. An Anomaly-based Detection System for Monitoring Kubernetes Infrastructures. IEEE Latin America Transactions, 2023;21: 457-465.
2. Cai H, Wang C and Zhou X. Deployment and verification of machine learning tool-chain based on Kubernetes distributed clusters: This paper is submitted for possible publication in the special issue on high performance distributed computing. CCF Transactions on High Performance Computing, 2021;3: 157-170.
3. Carnero A, Martín C, Torres DR, Garrido D, Díaz M and Rubio B. Managing and deploying distributed and deep neural models through Kafka-ML in the cloud-to-things continuum. IEEE Access, 2021;9: 125478-125495.
4. Czyzewski A, Stepień K and Poniszewska-Maranda A. Dynamic Development of Artificial Intelligence Models with CI/CD Environment-a Case Study. In 2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2024: 451-458.
5. Dedousis P, Stergiopoulos G, Arampatzis G and Gritzalis D. Enhancing Operational Resilience of Critical Infrastructure Processes Through Chaos Engineering. IEEE Access, 2023.
6. Deng S, Zhao H, Huang B, Zhang C, Chen F, Deng Y, Yin J, Dustdar S and Zomaya AY. Cloud-native computing: A survey from the perspective of services. Proceedings of the IEEE, 2024.
7. Hettiarachchi LS, Jayadeva SV, Bandara RAV, Palliyaguruge D, Arachchillage USSS and Kasthurirathna D. Artificial Intelligence-Based Centralized Resource Management Application for Distributed Systems. In 2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT), 2022: 1-6.
8. Hosseini MM and Parvania M. Artificial intelligence for resilience enhancement of power distribution systems. The Electricity Journal, 2021;34: 106880.
9. Hu F, Mehta K, Mishra S and AlMutawa M. Distributed Edge AI Systems. In Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing, 2023;1-6.
10. Jorge-Martinez D, Butt SA, Onyema EM, Chakraborty C, Shaheen Q, De-La-Hoz-Franco E and Ariza-Colpas P. Artificial intelligence-based Kubernetes container for scheduling nodes of energy composition. International Journal of System Assurance Engineering and Management, 2021: 1-9.
11. Kosińska J, Baliś B, Konieczny M, Malawski M and Zieliński S. Toward the observability of cloud-native applications: The overview of the state-of-the-art. IEEE Access, 2023;11: 73036-73052.
12. Mitchell BS. Cloud Native Software Engineering, 2023.
13. Mittal U and Panchal D. AI-based evaluation system for supply chain vulnerabilities and resilience amidst external shocks: An empirical approach. Reports in Mechanical Engineering, 2023;4(1): 276-289.

14. Palacios Chavarro S, Nespoli P, Díaz-López D and Niño Roa Y. On the way to automatic exploitation of vulnerabilities and validation of systems security through security chaos engineering. Big Data and Cognitive Computing, 2022;7(1): 1.

15. Ruospo A and Sanchez E. On the reliability assessment of artificial neural networks running on ai-oriented mpsocs. Applied Sciences, 2021;11(14): 6455.

16. Serbout S, El Malki A, Pautasso C and Zdun U. API Rate Limit Adoption--A pattern collection. In Proceedings of the 28th European Conference on Pattern Languages of Programs, 2023;1-20.

17. Spatharakis D, Dimolitsas I, Vlahakis E, Dechouniotis D, Athanasopoulos N and Papavassiliou S. Distributed resource autoscaling in kubernetes edge clusters. In 2022 18th International Conference on Network and Service Management (CNSM), 2022: 163-169.

18. Zhou J, Zhang K, Zhu F, Shi Q, Fang W, Wang L and Wang Y. Elastic Dl: A kubernetes-native deep learning framework with fault-tolerance and elastic scheduling. In Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, 2023: 1148-1151.