BUILDING RELIABLE CLOUD SYSTEMS THROUGH CHAOS ENGINEERING

Yolam Zimba

University of Lusaka, School of Post Graduate Studies, Lusaka, Zambia

ABSTRACT

Cloud computing systems need to be reliable so that they can be accessed and used for computing at any given point in time. The complex nature of cloud systems is the motivation to conduct research in novel ways of ensuring that cloud systems are built with reliability in mind. In building cloud systems, it is expected that the cloud system will be able to deal with high demands and unexpected events that affect the reliability and performance of the system.

In this paper, chaos engineering is considered a heuristic method that can be used to build reliable cloud systems. Chaos engineering is aimed at exposing weaknesses in systems that are in production. Chaos engineering will help identify system weaknesses and strengths when a system is exposed to unexpected knocks and shocks while it is in production.

Chaos engineering allows system developers and administrators to get insights into how the cloud system will behave when it is exposed to unexpected occurrences.

KEYWORDS

Reliability, Cloud Computing, Chaos Engineering, Distributed Systems

1. INTRODUCTION

Cloud reliability is a measure of the probability that a cloud system delivers the services it was designed for at any given point in time [2]. This definition implies that the cloud service should be available at any given point in time and function as expected even if unexpected events do occur. Cloud systems just like any other computer systems are exposed to various shocks and knocks while they are in production or live environments. These shocks and knocks could be a result of an unexpected increase in user requests, an unexpected error, or system misuse by the user. The system should be able to scale up or down to cater for the sudden rise in demand or system usage. [7] states that reliability means that the system should not fail during its operating period.

When we build cloud systems, the goal is to ensure that the system is always available and accessible to users at any given point in time. Through chaos engineering, experiments are conducted on a system in order to build confidence in the system's ability to withstand turbulent conditions while the system is in production [1].

[4] defines reliability as

"the ability of a system or component to perform its required functions under stated conditions for a specified period of time."

DOI: 10.5121/ijmit.2022.14201

This definition coincides with [2] but [4] further breaks down the definition by stating that the definition of reliability has the following components:

- ***Ability:*** This can be measured or expressed quantitatively with the aid of probability.
- ***Required Function:*** This relates to the expected performance of a system or component.
- ***Conditions:*** This relates to the environmental conditions in which the system or component operates.
- ***Specified time:*** This refers to the mission time which provides the expected duration of time in which the component or system is expected to operate.

[4] further provides a mathematical definition of reliability by positing that, reliability is defined as:

"the probability that the random variable time to failure (T) is greater or equal to mission time (t) "

This is illustrated by the formula shown below

$$R(t) = P(T \geq t) \tag{4}$$

[7] simplifies the definition of reliability by stating that:

"Reliability is the probability of no failures in an interval"

This paper does not delve deeper into the mathematical aspect of reliability, but rather dwells much on the principles of chaos engineering and how these principles can be used to build reliable cloud systems. Researchers or readers who are interested in exploring the reliability phenomenon in a mathematical sense can refer to [4] and [7] as these texts have propounded further on the mathematical aspect of reliability engineering.

2. CHAOS ENGINEERING

The above section has laid a foundation by defining reliability and stating that cloud systems need to be designed with reliability in mind. In this section, chaos engineering is explored in detail.

[5] states that Chaos Engineering

"*Involves experimenting on a distributed system to build confidence in its capability to withstand turbulent conditions in production*"

[6] further posits that chaos engineering is,

"*The discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production."*

This is a very important aspect if we are to build reliable cloud systems. Cloud system users need to have confidence in the cloud system and be able to depend on the cloud system for their computing activities. Chaos engineering is one technique that can be used in this regard. Chaos engineering is more of a back-end technique that is used by system engineers to inject failure into

a distributed system and observe how the system performs based on hypotheses that are conjured by system engineers and developers.

[6] propounds that Chaos Engineering is,

"*An empirical, systems-based approach that addresses the chaos in distributed systems at scale and builds confidence in the ability of those systems to withstand realistic conditions."*

With this approach, we learn about the behavior of the distributed system by observing how the system behaves during controlled experiments in which the system is purposefully exposed to aberrant conditions while it is in production [5].

2.1. Principles of Chaos Engineering

Section 3.0 has introduced chaos engineering and its aim. In this section, the principles of chaos engineering are introduced.

The chaos engineering phenomenon revolves around four principles, these are:

1. **Build a hypothesis around steady-state behavior***:* In the first principle of chaos engineering, we define what our steady-state should be. Steady-state here refers to attributes or factors that we consider favorable or in a nutshell what defines the system when it is working properly [5]. These could be attributes like availability, response time, or scalability. The main aim of chaos engineering is to ensure that the cloud system is working properly and the first principle is aimed at ensuring that working properly is defined and the metrics used to measure a system that is working properly are defined [5]. With this principle, it is implied that complex systems will exhibit certain regular behaviors that enable system engineers to predict how the system will perform. In chaos engineering, we come up with hypotheses around how the injection of system shocks and knocks will affect the system while it is in production [5]. The hypotheses formed during chaos engineering should target a specific metric. This metric could be availability or scalability [5].
   Chaos engineering focuses on the measurable outputs of a system rather than the internal attributes of the system [6].

2. **Vary real-world events***:* This principle suggests that we pick the stimulus for the chaos experiment from all possibilities that might occur in the real world or in a given realistic scenario. This enables system engineers to test the system on real world events that have probably occurred or might occur and this ensures that the experiments are controlled and that they are realistic. In a nutshell any input that can affect the steady state is a worthy candidate for input or injection into the system undergoing chaos experiments [5][6]. Some examples of such inputs could be failing a critical service, flooding the cloud system with requests to determine if the system is scalable and performing other aberrant operations that give very good insights into the behavior of the system when it is exposed to harsh or unfamiliar operations [5]. Although we want to get real insights of what will happen when the cloud system is exposed to shocks and knocks,[5] provides a cautionary statement by stating that, engineers should be cautious at this stage, the aim is not to harm the system but to get an understanding of what really happens when the system is exposed to harsh conditions. Any event that is likely to disturb the steady-state in considered as a potential variable in a chaos experiment [6] but cloud system engineers should keep in mind that the aim is not to harm or break the cloud system but to get realistic insights into how the system will function when exposed to aberrant situations or conditions in production.

3. **Run experiments in production***:* This principle advocates for running chaos experiments when the cloud system is in production [6]. A number of tools like cloud sim allow cloud systems to be simulated. Chaos experiments must be run on live systems as it is not possible to simulate all aspects of a live system. The only sure way of taking all aspects of a system into consideration is when the system is in production (live system) and it is being used for computing by cloud users. According to [6] it is critical that the experiments are run in such a way that cloud users are not severely affected during the experiments. [6] describes this as limiting the blast radius. Cloud engineers should be able to row back quickly if the experiment seems to get out of control or affects the cloud users.

4. **Automate experiments to run continuously***:* The final principle of chaos engineering advocates for the use of automation to ensure consistency and to build confidence in the implemented solutions. Scripts can be used here to ensure that aberrant conditions are constantly injected in the live system so as to ensure that the solutions implemented are end to end and that the steady-state is maintained. This results in system engineers gaining confidence in the solutions implemented. This further leads to reliable cloud systems that are perceived as dependable by the users.
   According to [5] the idea of using automation is very intimidating because failure is being automated and injected into the system, this type of purposeful stimulation of system failures is quite daunting for the system engineer.
   Automating the chaos experiments is better than running the experiments manually [6]. Manual experiments are tedious and difficult to track. It is also very difficult to drive system orchestration and analysis when the experiments are manual [6].

2.2. Chaos Engineering Life Cycle

Based on the principles explained in section 2.1, the figure shown below is the proposed life cycle that can followed to implement successful chaos experiments

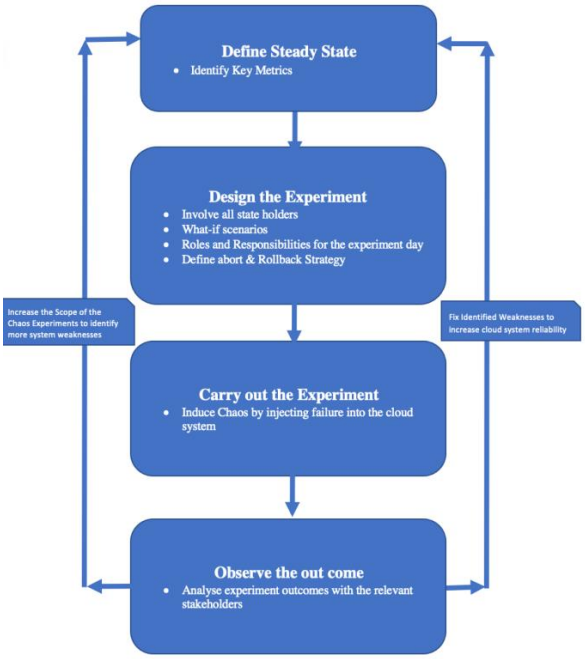

Figure 1. Chaos Engineering Life Cycle adapted from[7]

From figure 1, we can conclude that once a weakness is identified, it has to be fixed and this increases the reliability of the cloud system. On the other hand, if the system is able to overcome the induced failure, the scope of the experiment must be increased to identify more weaknesses. The goal of the chaos experiments is to identify the cloud system weaknesses and develop solutions or mechanisms that increase the reliability of the cloud system. Chaos engineers should have this in mind. The aim is not to break the system but to identify weaknesses through experimentation to gain empirical insights into how the system performs when it is exposed to aberrant conditions.

3. RELIABLE CLOUD SYSTEMS DESIGN CONSIDERATIONS

Section 2.0 introduced the concept of reliability while section 2.1 introduced the concepts and principles of chaos engineering. In this section, some aspects that should be considered by cloud system engineers when designing reliable cloud systems are discussed.

Designing reliable cloud systems is not easy and cloud engineers are aware of the complex nature of cloud systems. It is therefore imperative that different design options are considered and appreciating the pros and cons of each design method chosen is really critical to the design of reliable cloud systems [3].

Cloud system engineers should base their solutions on facts rather than conjuncture when designing reliable cloud systems. These facts can be gathered through the use of chaos engineering. Chaos engineering is one of the empirical methods that can be employed by cloud system designers to really formulate solutions that are tried and tested when building reliable cloud systems.

Cloud systems are complex, but it is always best to simplify the design so as to reduce on the complexity of cloud system [3]. [3] advocates for leaving out marginal complex features when designing systems.

[3] further posits that some complex system features should be discarded and system engineers should aim at simplifying systems but meet the key functional requirements of the system.

[3] affirms the trade off during system design by stating that:

"*Sometimes it will be necessary to discard a few design objectives to achieve a good design. The system engineer should always keep in mind that the design objectives generally contain a list of key features and a list of desirable features. The design must satisfy the key features, but if one or two of the desirable features must be eliminated to achieve a superior design, the trade-off is generally a good one."*

[12] proposes that in designing cloud systems, cloud system engineers should consider splitting the functions of the application and clustering the cloud application [12]. These designs help with isolating system components that develop faults. Though clustering may introduce another level of complexity. In a cluster, a group of systems or components function as one unit, if a unit develops a fault, it might be very difficult to get the system up and running.

4. EXAMPLES OF TOOLS USED IN CHAOS ENGINEERING

The concept of chaos engineering might seem speculative to researchers or readers who have not come across this phenomenon. In this section, examples of real-world tools used to automate chaos experiments are given.

Using tools conforms to the fourth principle of chaos engineering. Automating chaos experiments can efficiently be implemented by using tools.

- ***Chaos Monkey:*** Chaos Monkey was developed by the system engineers at Netflix [1][5]. It is a tool used to test the resilience of the cloud infrastructure [7]. This tool randomly terminates instances in production in order to ensure that the services implemented by system engineers are resilient [8].
- ***Litmus:*** This tool is mostly used to orchestrate chaos in the Kubernetes environment. This tool is an open-source tool and it is used to inject unexpected failures into a system in order to improve the systems resilience [9]. This tool is used to detect bugs in the production environment so that system engineers can improve the reliability of the system [7].
- ***Gremlin:*** This tool is considered a "Failure as a service" tool and it is aimed at making the internet more reliable. It offers a platform that can be used by system engineers to design and safely conduct experiments on complex systems in order to identify system weaknesses and improve the resilience of the cloud system before cloud users are affected [7][10].
- ***Toxiproxy:*** Is a tool designed to test network connections. It is tool that can used to ensure that a system has no single point of failure [11]. It has features that enable randomised chaos and customisation. It supports deterministic tampering with connections [7][11].

The examples given above are very useful if Chaos engineering is to be used to design reliable cloud systems. The tools can be very complex to implement [11] and require a high level of specialization.

5. CONCLUSION

Chaos engineering can be used to design reliable cloud systems. Reliable cloud systems are always available and function as expected by the users at any given point in time. The complex nature of these systems must be appreciated and this should be the motivation for employing design techniques and approaches that make cloud systems reliable.

Users need to gain the confidence in cloud systems and cloud system engineers need to be confident that their solutions work. This can only be achieved by injecting failures in live systems and then observing how the cloud system performs when exposed to aberrant conditions while the system is in production and users are using the system. This is the key theme of chaos engineering, injecting failure in live systems and observing how the systems performs. This leads to system engineers gaining useful insights into the cloud systems. Techniques like chaos engineering result in the production of reliable and dependable cloud systems that are fault tolerant.

REFERENCES

[1] Russ Miles, 2019, Learning Chaos Engineering, First Edition, O'Reilly Media, United States of America.

[2] Mazin yousif, 2018, Cloud Computing Reliability—Failure is an Option IEEE Cloud Computing.

[3] Martin L. Shooman, 2002, Reliability of Computer Systems and Networks: Fault Tolerance, Analysis and Design, John Wiley & Sons Inc.

[4] Ajit Kumar Verma. Srividya Ajit, Durga Rao Karanki, 2016, Reliability and Safety Engineering, Second Edition, Springer Series in Reliability Engineering.

[5] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal, 2016, Chaos Engineering, IEEE software, www.computer.org/software

[6] Principles of Chaos Engineering, 2019, https://principlesofchaos.org/. Last accessed on 23rd April 2022

[7] Chaos Engineering: Approaches, Best Practices and Case Studies, Infostretch, White Paper, https://www.infostretch.com/resources/white-papers/chaos-engineering/ last accessed on 3rd May 2022.

[8] Chaos Monkey, https://netflix.github.io/chaosmonkey/, last accessed on the 3rd of May 2022.

[9] Saiyan Pathak, 2020, Chaos Engineering for Kubernetes with Litmus, https://www.civo.com/learn/chaos-engineering-kubernetes-litmus last accessed on 3rd May 2022

[10] IBM, Chaos Engineering with Gremlin,https://www.ibm.com/cloud/architecture/architecture/practices/chaos-engineering-with-gremlin-on-cloud/ last accesses on 3rd May 2022.

[11] Timothy Agustian, 2021, Simulating Customised Chaos in Golang using Toxiproxy, https://medium.com/tokopedia-engineering/simulating-customized-chaos-in-golang-using-toxiproxy-b913584d88a7 last accessed on 3rd May 2022.

[12] Kola Ayanlowo, O. Shoewu, Segun O. Olatinwo, Tobi Samuel Fadiji, Segun Adeyanju ,2012, Conceptual Design and Implementation of a Cloud Computing Platform Paradigm, Computer Engineering and Intelligent Systems, ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online), Vol 3, No.12, 2012.