From Chaos Monkey to Controlled Chaos - The History of Chaos Engineering

Jennifer Larsson

jennilar@kth.se

April 5, 2023

1 Introduction

For a long time, we have known that testing is a crucial part of software development. By preparing for the worst, we can make our systems better. But what if we don't know what *the worst* is? Chaos Engineering is a DevOps principle, that tests systems for attacks that we might not be able to predict and cover with traditional testing methods. By deliberately inducing various types of failures into a system, we can expose weaknesses and identify the root causes of the issues, resulting in improved resilience and robustness.

Chaos Engineering is an important topic in the DevOps community, with an exciting history, present, and future. In this essay, we will go through the journey that Chaos Engineering has gone through, from its beginnings more than ten years ago to how it is used today and how it may develop in the future.

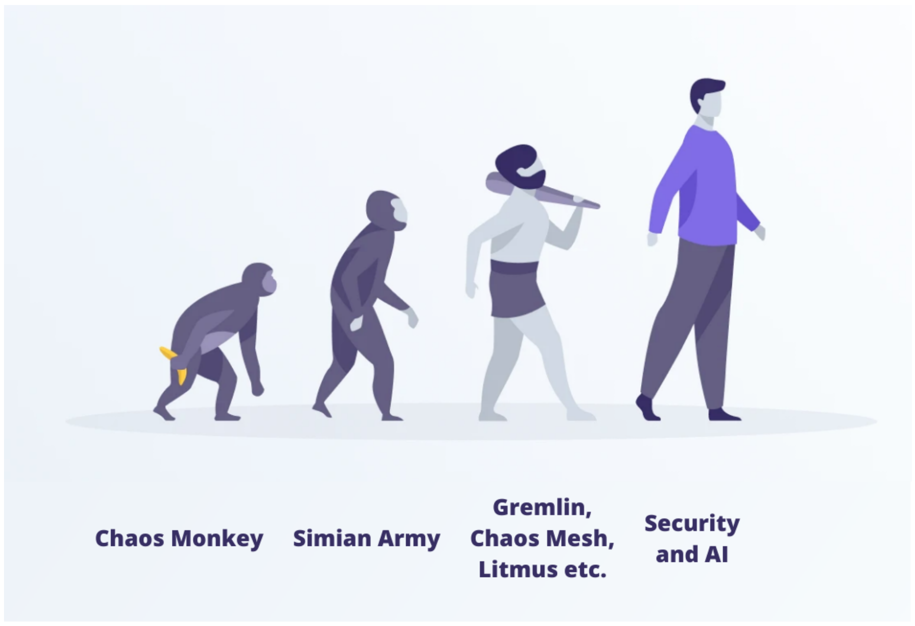

Figure 1: The history, present and future of Chaos Engineering

2 History of Chaos Engineering

The history of Chaos Engineering began in 2008 at Netflix. Up until this point, Netflix stored its data in a monolithic architecture with a single point of failure. In 2008, a major database corruption hit Netflix hard and left them unable to perform their services for three days. To avoid a similar disaster in the future, Netflix began a migration to the cloud. Their database shifted from a single point of truth to a complex distributed cloud architecture, and the increased complexity created a need for improved resiliency to random failures [1]. And thus, the idea of Chaos Monkey was born.

2.1 Chaos Monkey

In 2010, Netflix released Chaos Monkey which introduced a new engineering principle that would later come to be embraced by all types of software development organizations [2]. According to Netflix, the catchy name comes from "the idea of unleashing a wild monkey with a weapon in your data center (or cloud region) to randomly shoot down instances and chew through cables — all the while we continue serving our customers without interruption" [3].

Chaos Monkey was built to induce random failures in the system to expose vulnerabilities. By analyzing these vulnerabilities, developers could identify weak spots within the system and tackle the root cause of the problem. This enabled developers to gain deeper knowledge about the system and to understand how to make it more robust and resilient [2].

2.2 Simian Army

The next phase in the history of Chaos Engineering was the Simian Army. After the success of Chaos Monkey, Netflix introduced a suite of failure injection tools, each inducing a different type of failure to expose a wide range of weaknesses within the system [4]. Together, they make a "virtual Simian Army" [3].

The Simian Army has had many different members over the years. However, only three are publicly available for use today, not including Chaos Monkey. In this section, each of these will be described.

- **Janitor Monkey**
  Janitor Monkey identifies and disposes of unused resources in the cloud using a set of rules [5].
- **Conformity Monkey**
  Conformity Monkey identifies cloud instances that do not conform to the predefined set of rules for the best practice [6].
- **Security Monkey**
  Security Monkey monitors and analyzes the security of cloud configurations [7].

3 Modern Chaos Engineering

Chaos Engineering has come a long way since its beginnings as Chaos Monkey at Netflix. Today, it is a widely embraced set of engineering practices with a specific set of rules and principles to create controlled and optimized chaos. In this section, modern Chaos Engineering will be described and analyzed.

Modern Chaos Engineering has enabled a shift to a *system perspective*. Instead of viewing a set of services and components running in production as separate instances, Chaos Engineering views it as a single integrated system. With this mindset, developers can increase their understanding of a system by injecting real-world inputs instead of using fabricated test cases. The main interest in Chaos Engineering is analyzing a system's typical behavior over time, i.e the steady-state behavior, and asking the corresponding questions. This creates a broader perspective than traditional testing which is mainly concerned with whether the implementation matches the specification [8].

3.1 Principles

According to experimental disciplines, designing an experiment requires specifying the hypothesis, independent variables, dependent variables, and context. In Chaos Engineering, four key principles embody these steps [8], illustrated in figure 1.

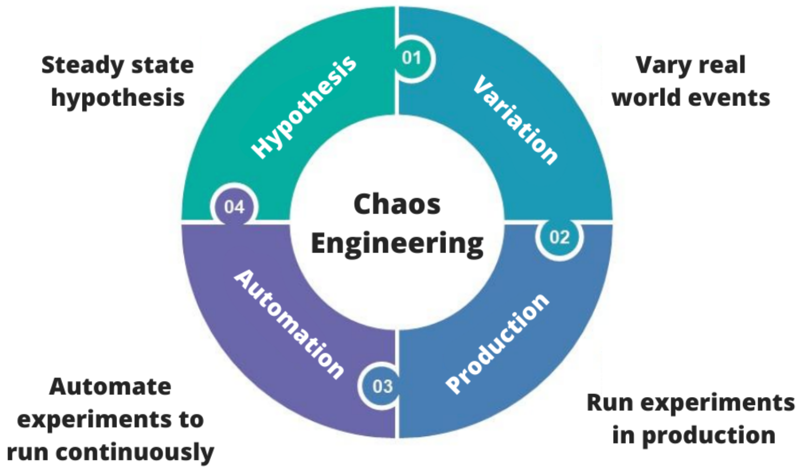

Figure 2: The four principles of Chaos Engineering

1. **Steady state hypothesis**
   The first principle of Chaos Engineering is building a hypothesis about the expected steady-state behavior of the system [9]. The steady state is the state where all components and services of the system are working properly, and in alignment with business standards and goals. The

hypothesis defines how the system should behave in the steady state, for example, outputs, responses, and performance corresponding to different events [10].

2. **Vary real-world events**
   To test the hypothesis and what will break the steady state, the second principle is varying real-world events. This means that failures are injected or simulated in a testing environment [8]. The aim is to simulate realistic failures and stressors that can alter the behavior of the system and break the steady-state hypothesis. Different hardware or software stressors can be injected, such as server crashes, cloud instance failures, and traffic spikes [10].

3. **Run experiments in production**
   It may not be possible to replicate an entire system in a test context. Therefore, tests are also run in a production environment to fully understand the impact of the test, i.e the blast radius [11]. In distributed cloud environments, it is crucial to test the interactions between different services and components. Since it is not possible to fully reproduce all aspects of a system in a testing environment, it is necessary to run experiments in production to achieve full integration testing [8].

4. **Automate experiments to run continuously**
   The final principle is to automate experiments to run continuously to maintain confidence in results over time [9]. By validating the steady-state hypothesis at scale and over time, developers get a consistent source of data and can gain deeper insight into the system [10].

The principles of Chaos Engineering require a deep understanding of the system and its expected behavior [11]. The developers must be able to specify an informed hypothesis, analyze the results and construct a suitable solution.

3.2 Benefits

There are many positive sides to Chaos Engineering, and in this section, some of the most important benefits will be described.

- **Increased resiliency and reliability**
  Chaos Engineering gives deep insight into the system and how it behaves under heavy load or stress, which allows developers to make systems more resilient and reliable [11].

- **Improved design decisions**
  Chaos Engineering practices have shifted the mindset from "*if* component X fails..." to "*when* component X fails..." [12]. This subtle change enables developers to make design decisions with failure in mind and to make sure there is a solution in place for when a component inevitably fails.

- **Faster incident response**

By learning what possible incident scenarios exist, developers can respond fast with troubleshooting and repairs [11].

- **Understanding service criticality**
  In a cloud environment that is susceptible to partial failures, it is crucial to identify which services are critical within a system. By identifying the blast radius, Chaos Engineering helps developers to understand which services are critical and should be prioritized [12].

3.3 Challenges and Drawbacks

Although Chaos Engineering has many benefits, it comes with a set of challenges and drawbacks, which is why it should be practiced with caution in a controlled and systematic way.

- **Lack of visibility**
  To fully understand critical dependencies and to identify root causes of issues, it is important to have end-to-end monitoring and visibility of a system [11]. This is a common challenge within Chaos Engineering that can create big problems.
- **Unnecessary damage**
  One big concern with Chaos Engineering is the risk of causing unnecessary real-world damage. To avoid this, experiments should not exceed the designated blast radius, and the scope of the blast radius should be minimized [11].
- **Lack of clarity**
  Chaos Engineering requires a clear picture of the steady state of the system [11]. However, it can be difficult to clearly define the expected behavior, which compromises the tests and makes the results ambiguous.

3.4 Tools and Platforms

Today, there exist many Chaos Engineering tools and platforms. This section will go through three of the biggest tools and platforms - Gremlin, Chaos Mesh, and Harness, and their benefits and drawbacks.

3.4.1 Gremlin

Gremlin is one of the top Chaos Engineering platforms and the first designed to improve web-based reliability. Its features include multi-level attacks, controlling failures, and testing various types of attacks, including memory leaks, latency injections, and more [13]. Furthermore, Gremlin can autodetect infrastructure components and recommend experiments to identify common causes of failure [14]. However, Gremlin does not support custom faults and have lacking reporting capabilities [13].

3.4.2 Chaos Mesh

Chaos Mesh is another popular Chaos Engineering platform. It is an open-source Kubernetes tool that supports testing for 17 different types of attacks, including resource consumption, network latency, and more [15]. Experiments can be run immediately or be scheduled and may be paused if needed [13]. One negative aspect of Chaos Mesh is that experiments run indefinitely, as the only way to set the duration is to schedule or manually terminate the experiment [15].

3.4.3 Litmus / Harness Chaos Engineering

Litmus is an open-source Chaos Engineering tool, but the project is owned by Harness, which provides Harness Chaos Engineering - a fully managed SaaS version of Litmus. Litmus includes a health-checking feature and supports various types of experiments. One drawback to Litmus is that it is quite complicated and requires an experienced tester [15].

4 Future of Chaos Engineering

Since its start at Netflix, Chaos Engineering has gone through tremendous development, and more is to come. In this section, the key trends in Chaos Engineering will be described, as well as their potential use cases in the real world.

4.1 Security Chaos Engineering

The first trend in Chaos Engineering is the focus on cyber security. The digitization of the world has created a crucial need for strong cyber security measures. In traditional testing, testers test a system's resiliency against given input. However, this does not necessarily cover all possible threats. Chaos Engineering on the other hand can test the resiliency of a system against unexpected attacks [16]. By leveraging the principles of Chaos Engineering and injecting failures that impact integrity and confidentiality, it is possible to gain insight into potential security threats [17].

In Security Chaos Engineering, the principles described in section 3.1 can be applied in the following way.

1. **Steady state hypothesis**
   The steady-state hypothesis should describe the *secure* state of the system. For example, this can include which users should have read/write access to certain components [17].

2. **Vary real-world events**
   To test the hypothesis, real-world events that impact the security of a system should be varied. This could mean creating, modifying, or deleting resources, as well as changing access policies [17].

3. **Run experiments in production**
   As in traditional Chaos Engineering, after running tests in a development environment, they should be run in production to identify the blast radius. In Security Chaos Engineering, it is important to minimize the potential damage as it affects the security of the system, so safety measures for recovering the system to a secure state are essential [17].

4. **Automate experiments to run continuously**
   As opposed to traditional security testing, Security Chaos Engineering leverages automation, which is important to see make the system secure over time [17].

4.2 AI and Chaos Engineering

Another trend in Chaos Engineering is incorporating it into artificial intelligence systems. AI and specifically machine learning models are trained on data, and their behaviors are extremely dependent on the composition of the dataset. A faulty dataset results in a faulty model, which makes ML models very susceptible and vulnerable to attacks [16]. By applying Chaos Engineering practices to the development of an ML model, the model's behavior under stress can be observed, monitored, and improved [18].

5 Reflection and Conclusions

Chaos Engineering has gone through massive development since its humble beginnings in 2010. From the truly chaotic Chaos Monkey to the controlled and optimized Chaos Engineering principles widely embraced today, Chaos Engineering has made a long journey. And the development will continue in the future, as Chaos Engineering takes its place in cyber security and artificial intelligence. It is clear that Chaos Engineering has tremendous potential and can be applied in a variety of scenarios to make systems stronger, more secure, and more resilient. Throughout its history, present, and future, Chaos Engineering revolutionizes the way we think about and conduct testing of software systems. As Netflix's John Ciancutti said:

"The best way to avoid failure is to fail constantly" [19]

References

[1] C. Rosenthal and N. Jones. *Chaos Engineering: System Resiliency in Practice.* O'Reilly Media, 2020. ISBN: 9781492043812. URL: https://books.google.se/books?id=jVjbDwAAQBAJ.

[2] S Carey. *What is Chaos Monkey? Chaos engineering explained.* 2020. URL: https://www.infoworld.com/article/3543233/what-is-chaos-monkey-chaos-engineering-explained.html. (accessed: 01.04.2023).

[3] Y. Izrailevsky and A. Tseitlin. *The Netflix Simian Army.* 2011. URL: https://www.infoworld.com/article/3543233/what-is-chaos-monkey-chaos-engineering-explained.html. (accessed: 01.04.2023).

[4] Gremlin. *The Simian Army.* 2018. URL: https://www.gremlin.com/chaos-monkey/the-simian-army/. (accessed: 01.04.2023).

[5] M. Fu and C. Bennett. *Janitor Monkey — Keeping the Cloud Tidy and Clean.* 2013. URL: https://netflixtechblog.com/janitor-monkey-keeping-the-cloud-tidy-and-clean-d517ad74d648. (accessed: 01.04.2023).

[6] M. Fu and C. Bennett. *Conformity Monkey — Keeping your cloud instances following best practices.* 2013. URL: https://netflixtechblog.com/conformity-monkey-keeping-your-cloud-instances-following-best-practices-2aaff3479adc. (accessed: 01.04.2023).

[7] P. Kelley, K. Glisson, and J. Chan. *Announcing Security Monkey — AWS Security Configuration Monitoring and Analysis.* 2014. URL: https://netflixtechblog.com/announcing-security-monkey-aws-security-configuration-monitoring-and-analysis-1f2bfb001708. (accessed: 01.04.2023).

[8] A. Basiri et al. "Chaos Engineering". In: *IEEE Software* 33.3 (2016), pp. 35–41. DOI: 10.1109/MS.2016.60.

[9] K. Torkura et al. "CloudStrike: Chaos Engineering for Security and Resiliency in Cloud Infrastructure". In: *IEEE Access* 8 (2020), pp. 123044–123060. DOI: 10.1109/ACCESS.2020.3007338.

[10] PagerDuty. *What is Chaos Engineering?* URL: https://www.pagerduty.com/resources/learn/what-is-chaos-engineering/#toc-4. (accessed: 05.04.2023).

[11] S. Gunja. *What is chaos engineering?* 2021. URL: https://www.dynatrace.com/news/blog/what-is-chaos-engineering/#:~:text=Chaos%20engineering%20generally%20originates%20from,and%20stakeholders%20across%20the%20organization. (accessed: 02.04.2023).

[12] H. Tucker et al. "The Business Case for Chaos Engineering". In: *IEEE Cloud Computing* 5.3 (2018), pp. 45–54. DOI: 10.1109/MCC.2018.032591616.

[13] M. Schillerstrom. *The Top Chaos Engineering Tools.* 2022. URL: https://www.harness.io/blog/chaos-engineering-tools. (accessed: 02.04.2023).

[14] G. Lawton. *The Top Chaos Engineering Tools.* 2021. URL: https://www.techtarget.com/searchsoftwarequality/tip/Choosing-the-right-chaos-engineering-tools. (accessed: 02.04.2023).

[15] Gremlin. *Comparing Chaos Engineering tools*. 2020. URL: https://www.gremlin.com/community/tutorials/chaos-engineering-tools-comparison/. (accessed: 02.04.2023).

[16] Consulteer. *Chaos Engineering: Key Future Trends*. 2022. URL: https://www.consulteer.com/chaos-engineering-key-future-trends/. (accessed: 02.04.2023).

[17] Kennedy A. Torkura et al. "Security Chaos Engineering for Cloud Services: Work In Progress". In: *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*. 2019, pp. 1–3. DOI: 10.1109/NCA.2019.8935046.

[18] V. Raja. *Introducing chaos engineering to Machine Learning deployments*. URL: https://www.microsoft.com/en-in/industry/blog/machine-learning/2022/03/29/introducing-chaos-engineering-to-machine-learning-deployments/. (accessed: 02.04.2023).

[19] J. Ciancutti. *5 Lessons We've Learned Using AWS*. 2010. URL: https://netflixtechblog.com/5-lessons-weve-learned-using-aws-1f2a28588e4c. (accessed: 02.04.2023).