Chaos engineering
Software Testing ETSN20

1st Hannes Lantz
dept. Computer Science
Faculty of Engineering, Lund University
Lund, Sweden
ha7117la-s@student.lu.se

2nd Axel Peterson
dept. Computer Science
Faculty of Engineering, Lund University
Lund, Sweden
mas13ape@student.lu.se

3rd Jesper Grahm
dept. Computer Science
Faculty of Engineering, Lund University
Lund, Sweden
dat15jgr@student.lu.se

***Abstract*—Chaos Engineering is a strategy for testing a system's resilience and it was developed by Netflix in 2011. By deliberately introducing failures in the system while in production, information can be gathered about the behavior of the system and preventive measures can be taken. This strategy relies on the philosophy that failures will occur eventually and that you need to be ready for them when they do. Injecting failures into a system in production entails some risks but if done correctly will ensure that the system is as resilient as possible. New knowledge of the system gathered using this strategy will also greatly help in handling unexpected failures.**

***Index Terms*—Chaos Engineering, Testing, Netflix, Resilience**

I. Introduction

Testing large-scale systems is hard. It is impossible to test every part of a system. Not only does testing take time, it also costs money. However it still needs to be done as system failures can serious injury a product and company.

Chaos Engineering is a fairly new strategy for building system resilience. It has only been around for 8 years and is scarcely used by companies. A big factor to this might be the risks involving conducting Chaos Experimenting which might deter some companies. To fully get the benefits of Chaos Engineering a certain mindset must be embraced while still maintaining carefulness and proper risk analysis.

II. Background of chaos engineering

A. What is Chaos Engineering?

The concept of Chaos Engineering is a software testing strategy that involves deliberately putting strain on an active system to simulate unexpected events. Through this, the resilience of the of system can be assessed and preventive measures can be taken to ensure adequate quality of service. The resilience of the system is a measurement that shows the system's ability to tolerate failures while still being able to deliver the expected services that the system entails. Chaos Engineering can be used to check a system's resilience against for example Infrastructure, Network or Application failures [1].

The way Chaos Engineering works is often described using the Chaos Monkey metaphor. This describes the process of Chaos Engineering as letting out a monkey in a data center and letting it go wild. This monkey can for example rip out network cables, trash servers or rearrange critical hardware. If the system is resilient enough it will still be able to deliver the expected services despite this monkey doing a lot of damage to network and hardware connections. In practice there is of course no actual monkey but instead a system is set up to simulate this. This is done by sabotaging an active system by for example shutting down network connections and servers. Some failures that can be introduced are:

- Network connections interrupted
- Latency increase
- Data center shutdown (extreme)
- Hardware failure
- Database erase

These failures can be automated by having a running program that puts these kind of strains on the system once in a while.

B. Why was Chaos Engineering developed?

Chaos Engineering was developed by Netflix in 2011 during the migration of their service to the cloud. Since Netflix is a streaming service for movies it needs to have as close to 100% up time as possible. Large distributed sytems tend to fail a lot and they are hard to test with traditional testing [2]. Greg Orzell got the idea that instead of assuming no breakdowns would occur it would be better to see breakdowns as inevitable and take preventive measure to strengthen the system against them. This led to the development of Chaos Engineering and introducing resilience as an obligation for developer rather than an option.

C. Netflix's role in Chaos Engineering

As mentioned before, Netflix is the most key actor in Chaos Engineering. Their gigantic market and strict requirement of up time makes their service an ideal system for using Chaos Engineering. During their cloud migration in 2011 they developed "The Simian Army" a category of tools used to test a service using Chaos Engineering. This is further described in section IV.

III. An overview of Chaos Engineering

Chaos Engineering and testing differ slightly between one another. Testing is done with the goal of to see if your application run as expected within controlled boundaries. While

chaos is about stating a hypothesis and then find evidence to accept/reject said hypothesis. There are four base principles of Chaos Engineering [3].

- Build a hypothesis around steady-state behavior.
- Vary real-world events.
- Run experiments in production.
- Automate experiments to run continuously.

A. Build a hypothesis

Before designing a chaos experiment is it important to reflect on the potential outcome of running said experiment. Could it lead to a problem that affect customers or severely injure some other part of the system? These kinds of hypothesis are important to make [3].

B. Vary real-world events

It is easy for a developer to just test what they know and will not work and that will probably cover most common cases. There are however many edge cases that can happened in the real world that might never be seen when testing. Incorrect nonfatal error handling are the source for 92% of catastrophic system failures. This is the base for the second principle of Chaos Engineering. Chaos experiments should be designed input samples from the real world and that are believed to be able to potentially disrupt the steady-state of the system. Some examples of inputs that Netflix have used are latency-injection and set a entire Amazon region offline [3].

C. Run experiments in production

When a service is a distributed system based on many different servers coordinated over network traditional testing is not sufficient to detect all the failures and potential failures of the system. It is also not always possible to fully reproduce the entire system in a test context and real clients in a client-server relationship behave different than synthetic clients which makes it important to run failure tests in production in a relevant and realistic testing environment [3].

D. Automate experiments to run continuously

Since most systems nowadays are continuously being maintained, updated and modified it is important that the testing also is continuously being updated and ran since the trust in the results of past experiments will decrease when the system is modified [3]. Running automated chaos experiments on a live system with the goal to find failures in the system can sound scary and dangerous to someone that has never experienced chaos engineering before there are security measures that can be taken to prevent that the system breaks. Some of these are only running these automated chaos experiments during working hours and aborting the experiments if they detect the users being impacted. A couple other security mechanisms that can be used are explained in IV.

IV. Tools and Automation

There are a fair bit of different chaos-tools many of which are open sourced. Most of the tools are also developed by Netflix.

Chaos Monkey:

This is Netflix's first chaos tool in what later became the Simian Army. It began development in 2011, a couple years after the launch of their streaming platform. It is a tool for testing resilience by randomly shutting down parts of a system network [4].

Simian Army:

This is what Netflix calls their collection of testing tools that followed the success of Chaos Monkey. Two notable tools in this collection is Latency Monkey and Chaos Gorilla. Latency Monkey makes it possible to simulate a large delays to measure how upstream services react. It is particularly useful for testing new services. By simulating a dependencies failure it can test a service without taking those dependencies off from the rest of the system. Chaos Gorilla is in short a up scaled version of Chaos Monkey. It can simulate a complete Amazon availability zone going offline. This is used to test a service ability to automatically re-balance its resources on what is available in the system [4].

FIT:

FIT, Failure Injection Testing, is a system developed by Netflix that makes it easier to inject failures in a controlled manner on a microservice level in production. Before developing FIT Netflix had some problems with their monkeys, especially Latency Monkey, having to big impact on the on the whole system and FIT helped Netflix limit the impact of the testing to applications that did not need to be a part of that particular testing [5] [6].

ChAP:

ChAP, Chaos Automation Platform, is also a system developed by Netflix and the successor to FIT. While using FIT was an improvement on the impact on the whole system compared to before, FIT still required a large test group for the results to be detectable and not disappear in the systems natural noise. ChAP improves on this by using two small test groups that are being compared. One experiment group that experiences the failure test and a second control group to act as a baseline. ChAP also has four major security mechanisms. The first one is that the failure testing only run during business hours when there always are software engineers available to fix any issues if anything would break. The second is that the testing will stop if there is any major customer impact from the testing. The third security mechanism is limiting the total impact the testing have. To do this all currently running ChAP experiments cannot affect more than 5% of the total traffic. The last security mechanism is failover. When a big enough problems is detected Netflix can redirect traffic from the region with problems via the other regions and this is called failover. Since a failover can invalidate some of the assumptions and parameters used in the chaos experiments no ChAP experiments are allowed to run during the time of a

failover [6] [7].

Monocle: Monocle is another service developed and used by Netflix in their chaos testing. Monocle has two major functions. The first one is that Netflix uses Monocle to introspect Netflix's other services and the second one is to generate new experiments. The different types of experiments Monocle can generate are failure, latency just below a timeout and latency causing failure [6].

V. Analysis

We will look deeper into what we consider to be the important bits about Chaos Engineering. Benefits, risk and relation to traditional testing. With the first one being what we think is the most important one.

A. What are the benefits?

As mentioned before, the benefits of using Chaos Engineering are huge. Without the knowledge that it provides the developers are essentially sitting in the dark and waiting for something to go wrong. This provides very limited knowledge about how the system will react when something unexpected occurs. By using Chaos Engineering the developers can get a much greater understanding of the system and how it behaves in certain circumstances. This makes it very hard to make the system fail gracefully. No large system will be perfect and there will always exist failures and potential failures in all large systems. Thus if a small part of the system fail the system should fail gracefully in a way that if possible the end user should not be affected or even notice it. For example in the case of Netflix. If the service in their system that is manage the bookmarks or the rating on the series and movies fail, the end user should not not really be affected by it and should still be able to use the streaming service. Due to Chaos Engineering preventive measures can be taken to ensure that the system is able to handle unexpected failures which will eventually occur. By increasing the system's resilience a much more stable base is created and when a failure occurs there are clear paths and information generated by the experiments which will provide the developers with the knowledge to handle the failure in the best possible way.

Chaos Engineering is a relatively new practice and it can be hard and intimidating to start when an engineer or a team never experienced it before and it will cost time and money to implement and develop it will make your system more resilient and will prevent future costs of downtime and headaches when the system would otherwise eventually fail.

B. The risks of conducting Chaos Engineering

As can be expected there are many risks involved when sabotaging your own system in production but Rosenthal et al [8] describe some perceived risks as poor excuses for not conducting Chaos Engineering. They recognize that some system are not ideal for Chaos Engineering. The software of a self driving car containing passengers is a good example of a system that obviously is exempt from the need and usefulness of Chaos Engineering. Another good example is the software systems of a bank where even small down times can result in enormous loss of money and transactions. However most users do not work on system of this kind. A common excuse is "I'm pretty sure it will break". Rosenthal et al counter this with the argument that if you are reluctant to use Chaos Engineering on your system because you are afraid it will break then your system is not ready for Chaos Experiments and needs to mature further. You should only start conducting Chaos Engineering when you are relatively sure of the resilience of your system and that it can withstand at least some failures.

Another poor excuse according to [8] is "If it does break, we're in big trouble". This is a legitimate concern and a key part of Chaos Engineering. They key is to minimize the harm the experiment can do if it breaks the system by making it easy to abort the experiment and minimizing the blast radius of the experiment. Making it easy to abort can be done by implementing a big red abort button or by automating the process by having the experiment automatically abort when it detects a potentially harmful break in the system. Minimizing the blast radius is harder to implement and since the experiment often needs to be big enough to generate relevant findings but the bigger the test is, the more risk it can involve if it actually breaks something. One way to minimizing the blast radius is to conduct smaller tests on small groups of users and breaking up bigger tests into this smaller category. This way, if it does break, it only affects a small part of the system. Another strategy is to use custom routing to protect the users affected by the test if it breaks. By doing this the users can be redirected to working server och system fast if the system breaks.

As mentioned above there are definitely some risks with conducting Chaos Engineering. But there are accurate strategies to minimize the risks and a big part of the Chaos Engineering philosophy is to accept that failure will happen and when they do happen it benefits the developers and owners greatly if they are prepared for it.

C. How Chaos Engineering relates to Traditional Testing

Chaos Engineering have much in common with traditional testing in that it is used to find flaws in the system. But the strategy is very different from traditional test cases. According to [8] testing is often used to isolate specific cases. An assertion is made and a specific result is sought. This does not generate new information about the system which is an important distinction. All it does is see if the system behaves as predicted.

Using Chaos Engineering, on the other hand, new information about the system is generated. A failure is injected into the system and they way the system reacts produces a new behavior of the system under those circumstances. When conducting Chaos Experimenting no specific result is sought. Instead a failure is injected and the resulting behavior gives the developers new knowledge about the system. In this way, Chaos Experimenting can greatly help in getting a broader view of the system and how it might react to certain often

unexpected scenarios. This is extremely valuable for the health of these kinds of systems.

VI. Conclusion

Chaos Engineering is a fairly new practice that seems to be getting more attention. Most chaos-tools are open source which is a good thing as it makes it possible for a developer to use a already existing tool that is able to be further customized to suit the developers need. Open source is also a great as it makes it widely accessible which in turn boost the awareness of the concept. We think that Chaos Engineering will grow in popularity and usage in the years to come. It is clear that Netflix have had success with running chaos experiments on their services.

By conducting Chaos Engineering you ensure that your system has at least some resilience to unexpected failures. Not conducting it can be seen as very naive since the scenario of countless types of failures is not properly analysed. By getting into the mindset of Chaos Engineering and embracing that failures will happen you can be prepared for them and in many cases prevent before they even happen. This is the key to Chaos Engineering and a resilient system.

Chaos Engineering is important for both small and large systems. Small scale systems often have more to lose compare to the size of the system and often have not developed their backup, restore process and preventive measures as much as larger systems. It is also easier to start early and not have to start when the system already have a large user base. Since there are a lot of tools open-source it is also not huge cost issue for smaller systems.

Contribution Statement

A. Hannes Lantz

- Introduction
- Why was Chaos Engineering developed?
- Tools and Automation
- An overview of Chaos Engineering
- Conclusion

B. Axel Peterson

- An overview of Chaos Engineering
- Tools and Automation
- Analysis
- Conclusion

C. Jesper Grahm

- Abstract
- Introduction
- Background of Chaos Engineering
- Analysis
- Conclusion

References

[1] J. Simonsson, L. Zhang, B. Morin, B. Baudry, and M. Monperrus, "Observability and Chaos Engineering on System Calls for Containerized Applications in Docker," *arXiv:1907.13039 [cs]*, Jul. 2019, arXiv: 1907.13039. [Online]. Available: http://arxiv.org/abs/1907.13039

[2] H. Tucker, L. Hochstein, N. Jones, A. Basiri, and C. Rosenthal, "The Business Case for Chaos Engineering," *IEEE Cloud Computing*, vol. 5, no. 3, pp. 45–54, May 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8383672/

[3] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos Engineering," *IEEE Software*, vol. 33, no. 3, pp. 35–41, May 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7436642/

[4] The netflix simian army. [Online]. Available: https://medium.com/netflix-techblog/the-netflix-simian-army-16e57fbab116

[5] Fit: Failure injection testing. [Online]. Available: https://medium.com/netflix-techblog/fit-failure-injection-testing-35d8e2a9bb2

[6] A. Basiri, L. Hochstein, N. Jones, and H. Tucker, "Automating Chaos Experiments in Production," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Montreal, QC, Canada: IEEE, May 2019, pp. 31–40. [Online]. Available: https://ieeexplore.ieee.org/document/8804433/

[7] Chap: Chaos automation platform. [Online]. Available: https://medium.com/netflix-techblog/chap-chaos-automation-platform-53e6d528371f

[8] B. J. Rosenthal, Hochstein and Basiri, "Chaos engineering, building confidence in system behavior through experiments," *IEEE Software*, Aug. 2017. [Online]. Available: http://channyblog.s3-ap-northeast-2.amazonaws.com/data/channy/2018/01/18023151/chaos-engineering.pdf