"Chaos Engineering is the discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production."
---https//principlesofchaos.org---

WHY DO CHAOS ENGINEERING?
Advances in large-scale, distributed software systems are changing the game for software engineering. As an industry, we are quick to adopt practices that increase flexibility of development and velocity of deployment. As an illustration, our systems re scaling fast and our newest technology like microservice architecture is tricky also. An urgent question follows on the heels of these benefits: How much confidence we can have in the complex systems that we put into production?

Even when all of the individual services in a distributed system are functioning properly, the interactions between those services can cause unpredictable outcomes. Unpredictable outcomes, compounded by rare but disruptive real-world events that affect production environments, make these distributed systems inherently chaotic.

We need to identify weaknesses before they manifest in system-wide, aberrant behaviors. Systemic weaknesses could take the form of: improper fallback settings when a service is unavailable; retry storms from improperly tuned timeouts; outages when a downstream dependency receives too much traffic; cascading failures when a single point of failure crashes; etc. We must address the most significant weaknesses proactively, before they affect our customers in production. We need a way to manage the chaos inherent in these systems, take advantage of increasing flexibility and velocity, and have confidence in our production deployments despite the complexity that they represent.

An empirical, systems-based approach addresses the chaos in distributed systems at scale and builds confidence in the ability of those systems to withstand realistic conditions. We learn about the behavior of a distributed system by observing it during a controlled experiment. We call this Chaos Engineering.

CHAOS IN PRACTICE
These experiments follow these steps:
Steady state -> Define Hypothesis -> Design and execute -> Learn -> Fix -> Embed

- Start by defining 'steady state' as some measurable output of a system that indicates normal behavior.
  For examples:
  Amazon : 100 ms of extra load time caused a 1% drop in sales (Greg Linden).
  Google : 500 ms of extra load time caused 20% fewer searches (Marissa Mayer).
  Yahoo!: 400 ms of extra load time caused a 5–9% increase in the number of people who clicked "back" before the page even loaded (Nicole Sullivan).
- Define Hypothesis: brainstorm what can go wrong. Hypothesize that this steady state will continue in both the control group and the experimental group. Introduce variables that reflect real world events like servers that crash, hard drives that

malfunction, network connections that are severed, etc. Don't make an hypothesis that you know will break you!
- Design and execute: start small, notify people involved, as close as possible to production, minimize the blast redius.
- Verify and learn: Quantifying the result of the experiment: how fast did we detect, how fast did we recover and do not blame.
- Fix: implement fix and rerun experiment
- Embed: embed in culture onboarding continuous chaos

What tools we can use:
Chaos Monkey: Chaos Monkey is a resiliency tool that helps applications tolerate random instance failures.
Simian Army: Tools for keeping your cloud operating in top form.
Litmus: It is chaos engineering for stateful workloads on Kubernetes, hopefully without learning curves.
Powerful Seal: A powerful testing tool for Kubernetes clusters.
etc

COMPANIES DOING CHAOS ENGINEERING:
They are Netflix, Twilio, Dropbox, Uber, Amazon, Gremlin, Expedia and etc.