Principles of chaos engineering

en ar bs cz es fr hi hu it ja ko pl pt ro ru sr tr ua uz zh

Fork me on GitHub

PRINCIPLES OF CHAOS ENGINEERING

Last Update: 2019 March (changes)

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.

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. 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

To specifically address the uncertainty of distributed systems at scale, Chaos Engineering can be thought of as the facilitation of experiments to uncover systemic weaknesses. These experiments follow four steps:

1. Start by defining 'steady state' as some measurable output of a system that indicates normal behavior.
2. Hypothesize that this steady state will continue in both the control group and the experimental group.
3. Introduce variables that reflect real world events like servers that crash, hard drives that malfunction, network connections that are severed, etc.

4. Try to disprove the hypothesis by looking for a difference in steady state between the control group and the experimental group.

The harder it is to disrupt the steady state, the more confidence we have in the behavior of the system. If a weakness is uncovered, we now have a target for improvement before that behavior manifests in the system at large.

ADVANCED PRINCIPLES

The following principles describe an ideal application of Chaos Engineering, applied to the processes of experimentation described above. The degree to which these principles are pursued strongly correlates to the confidence we can have in a distributed system at scale.

Build a Hypothesis around Steady State Behavior

Focus on the measurable output of a system, rather than internal attributes of the system. Measurements of that output over a short period of time constitute a proxy for the system's steady state. The overall system's throughput, error rates, latency percentiles, etc. could all be metrics of interest representing steady state behavior. By focusing on systemic behavior patterns during experiments, Chaos verifies that the system does work, rather than trying to validate how it works.

Vary Real-world Events

Chaos variables reflect real-world events. Prioritize events either by potential impact or estimated frequency. Consider events that correspond to hardware failures like servers dying, software failures like malformed responses, and non-failure events like a spike in traffic or a scaling event. Any event capable of disrupting steady state is a potential variable in a Chaos experiment.

Run Experiments in Production

Systems behave differently depending on environment and traffic patterns. Since the behavior of utilization can change at any time, sampling real traffic is the only way to reliably capture the request path. To guarantee both authenticity of the way in which the system is exercised and relevance to the current deployed system, Chaos strongly prefers to experiment directly on production traffic.

Automate Experiments to Run Continuously

Running experiments manually is labor-intensive and ultimately unsustainable. Automate experiments and run them continuously. Chaos Engineering builds automation into the system to drive both orchestration and analysis.

Minimize Blast Radius

Experimenting in production has the potential to cause unnecessary customer pain. While there must be an allowance for some short-term negative impact, it is the responsibility and obligation of the Chaos Engineer to ensure the fallout from experiments are minimized and contained.

Chaos Engineering is a powerful practice that is already changing how software is designed and engineered at some of the largest-scale operations in the world. Where other practices address velocity and flexibility, Chaos specifically tackles systemic uncertainty in these distributed systems. The Principles of Chaos provide confidence to innovate quickly at massive scales and give customers the high quality experiences they deserve.

Join the ongoing discussion of the Principles of Chaos and their application in the Chaos Community.

Powered by Hugo & Pixyll