HOW DOES CHAOS TESTING WORKS?

In chaos engineering, each experiment begins with the introduction of a particular flaw into the system. The administrators compare what actually occurred to their predictions later on.

Here's the step-by-step flow of chaos engineering experiments in practice:

1. The steady state of the system is defined as a measured output that represents typical behavior.
2. Suggest on the output of the system and how it will differ between the experimental and control groups.
3. Introduce variables that represent actual occurrences, such as traffic spikes, hardware and software failures, and non-failure events.
4. By contrasting the steady state of the system in the control and experimental groups, work on refuting the theory.

Two groups of engineers typically participate in chaos engineering experiments. The first group typically manages the failed injection, while the second group takes care of the outcomes.

An example of testing tool : In order to address the demand for continuous and consistent testing, Netflix started chaos testing their system throughout their migration to Amazon web services. To this end, they built various "chaos monkeys." These chaotic monkeys were introduced into a system to imitate various real-world conditions and introduce unique faults, such as network latency, instances, missing data segments, etc.

Each chaos monkey had its own name and job, including:

- Latency Monkey: Induces artificial delays
- Conformity and Security Monkeys: Hunt and kill instances that don't adhere to best practices
- Janitor Monkey: Cleans up and removes unused resources
- Chaos Gorilla: Simulates an entire Amazon availability zone outage

Collectively, these and more chaos monkeys are now known as the Simian Army.

PROS

- IT and DevOps teams are better able to recognise and address problems that other testing methods might miss.

- Because of proactive and continuous testing, unplanned downtime and outages are far less likely to happen.
- improves the system's integrity
- Great for scaling up to huge, complicated systems

CONS

- Desktop software or smaller systems
- Services and programmes that are not essential to the operation of the business
- Application settings without customer service level agreements requiring constant uptime 24 hours a day
- Systems where errors are tolerated as long as they are fixed at the end of the day