insights.sei.cmu.edu

DevOps Case Study: Netflix and the Chaos Monkey

7-9 minutes

DevOps can be succinctly defined as a mindset of molding your process and organizational structures to promote

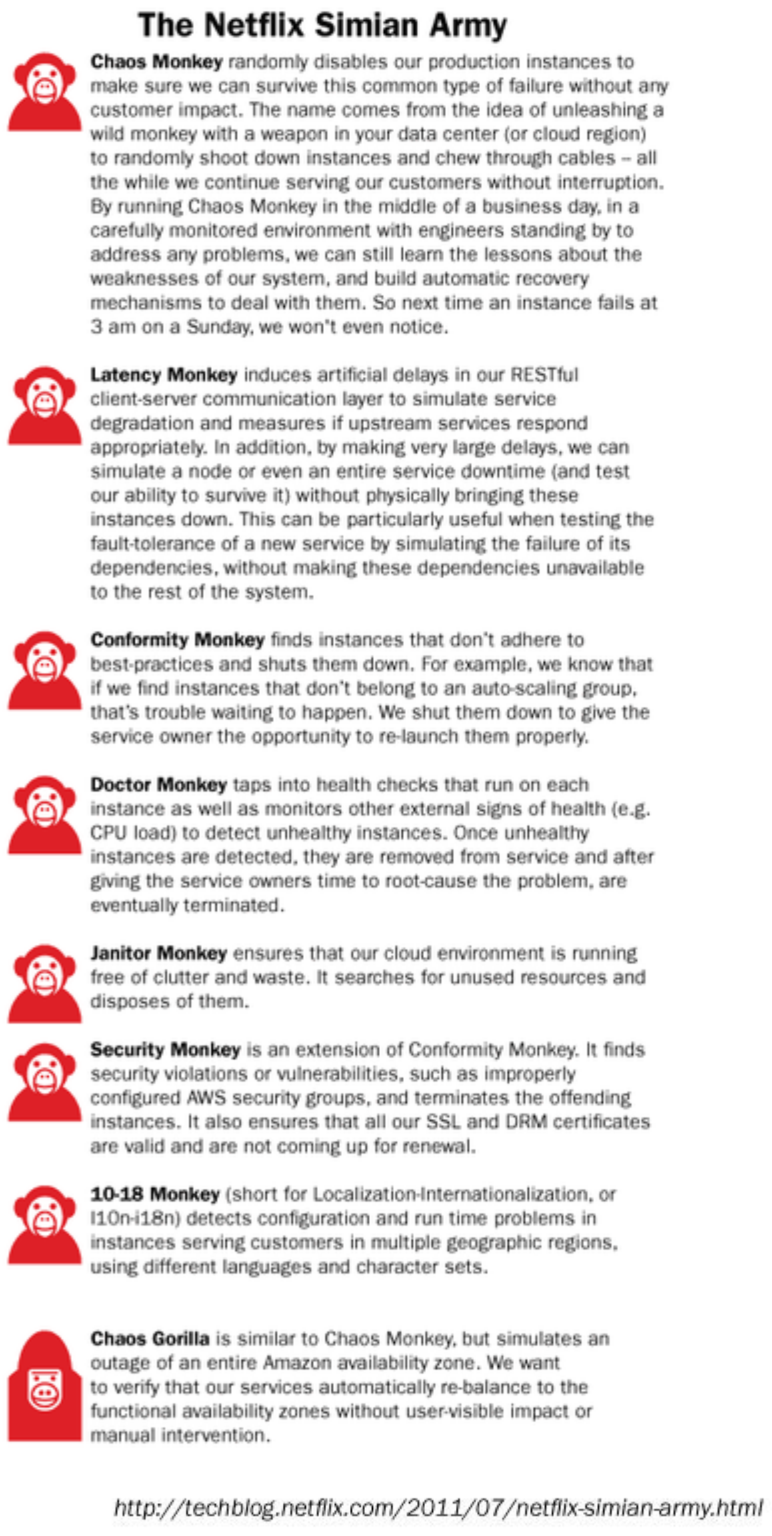
- business value
- software quality attributes most important to your organization
- continuous improvement

As I have discussed in previous posts on DevOps at Amazon and software quality in DevOps, while DevOps is often approached through practices such as Agile development, automation, and continuous delivery, the spirit of DevOps can be applied in many ways. In this blog post, I am going to look at another seminal case study of DevOps thinking applied in a somewhat out-of-the-box way: Netflix.

Netflix is a fantastic case study for DevOps because their software-engineering process shows a fundamental understanding of DevOps thinking and a focus on quality attributes through automation-assisted process. Recall, DevOps practitioners espouse a driven focus on quality attributes to meet business needs, leveraging automated processes to achieve consistency and efficiency.

Netflix's streaming service is a large distributed system hosted on Amazon Web Services (AWS). Since there are so many components that have to work together to provide reliable video streams to customers across a wide range of devices, Netflix engineers needed to focus heavily on the quality attributes of reliability and robustness for both server- and client-side components. In short, they concluded that the only way to be comfortable handling failure is to constantly practice failing. To achieve the desired level of confidence and quality, in true DevOps style, Netflix engineers set about *automating failure*.

If you have ever used Netflix software on your computer, a game console, or a mobile device, you may have noticed that while the software is impressively reliable, occasionally the available streams of videos change. Sometimes, the 'Recommended Picks' stream may not appear, for example. When this happens it is because the service in AWS that serves the 'Recommended Picks' data is down. However, your Netflix application doesn't crash, it doesn't throw any errors, and it doesn't suffer from any degradation in performance. Netflix software merely omits the stream, or displays an alternate stream, with no hindered experience to the user--exhibiting ideal, elegant failure behavior.



The Netflix Simian Army

Chaos Monkey randomly disables our production instances to make sure we can survive this common type of failure without any customer impact. The 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. By running Chaos Monkey in the middle of a business day, in a carefully monitored environment with engineers standing by to address any problems, we can still learn the lessons about the weaknesses of our system, and build automatic recovery mechanisms to deal with them. So next time an instance fails at 3 am on a Sunday, we won't even notice.

Latency Monkey induces artificial delays in our RESTful client-server communication layer to simulate service degradation and measures if upstream services respond appropriately. In addition, by making very large delays, we can simulate a node or even an entire service downtime (and test our ability to survive it) without physically bringing these instances down. This can be particularly useful when testing the fault-tolerance of a new service by simulating the failure of its dependencies, without making these dependencies unavailable to the rest of the system.

Conformity Monkey finds instances that don't adhere to best-practices and shuts them down. For example, we know that if we find instances that don't belong to an auto-scaling group, that's trouble waiting to happen. We shut them down to give the service owner the opportunity to re-launch them properly.

Doctor Monkey taps into health checks that run on each instance as well as monitors other external signs of health (e.g. CPU load) to detect unhealthy instances. Once unhealthy instances are detected, they are removed from service and after giving the service owners time to root-cause the problem, are eventually terminated.

Janitor Monkey ensures that our cloud environment is running free of clutter and waste. It searches for unused resources and disposes of them.

Security Monkey is an extension of Conformity Monkey. It finds security violations or vulnerabilities, such as improperly configured AWS security groups, and terminates the offending instances. It also ensures that all our SSL and DRM certificates are valid and are not coming up for renewal.

10-18 Monkey (short for Localization-Internationalization, or l10n-i18n) detects configuration and run time problems in instances serving customers in multiple geographic regions, using different languages and character sets.

Chaos Gorilla is similar to Chaos Monkey, but simulates an outage of an entire Amazon availability zone. We want to verify that our services automatically re-balance to the functional availability zones without user-visible impact or manual intervention.

http://techblog.netflix.com/2011/07/netflix-simian-army.html

To achieve this result, Netflix dramatically altered their engineering process by introducing a tool called [Chaos Monkey](), the first in [a series of tools collectively known as the Netflix Simian Army](). Chaos Monkey is basically a script that runs continually in all Netflix environments, causing chaos by randomly shutting down server instances. Thus, while writing code, Netflix developers are constantly operating in an environment of unreliable services and unexpected outages. This chaos not only gives developers a unique opportunity to test their software in unexpected failure conditions, but incentivizes them to build faulttolerant systems to make their day-to-day job as developers less frustrating. This is DevOps at its finest: altering the development process and using automation to set up a system where the behavioral economics favors producing a desirable level of software quality. In response to creating software in this type of environment, Netflix developers will design their systems to be modular, testable, and highly resilient against back-end service outages from the start.

In a DevOps organization, leaders must ask: *What can we do to incentivize the organization to achieve the outcomes we want? How can we change our organization to drive ever-closer to our goals?* To master DevOps and dramatically improve outcomes in your organization, this is the type of thinking you must encourage.

Then, most importantly, organizations must be willing to make the changes and sacrifices necessary (such as intentionally, continually causing failures) to set themselves up for success. As evidence to the value of their investment, Netflix has credited this 'chaos testing' approach to giving their systems the resiliency to handle the 9/25/14 reboot of 10 percent of AWS servers without issue. The unmitigated success of this approach inspired the creation of the Simian Army, a full suite of tools to enable chaos testing, which is now available as open source software.

Every two weeks, the SEI will publish a new blog post offering guidelines and practical advice for organizations seeking to adopt DevOps in practice. We welcome your feedback on this series, as well as suggestions for future content. Please leave feedback in the comments section below.

Additional Resources

To view the webinar *Culture Shock: Unlocking DevOps with Collaboration and Communication* with Aaron Volkmann and Todd Waits please click here.

To view the webinar *What DevOps is Not!* with Hasan Yasar and C. Aaron Cois, please click here.

To listen to the podcast D*evOps--Transform Development and Operations for Fast, Secure Deployments* featuring Gene Kim and Julia Allen, please click here.

To read all of the blog posts in our DevOps series, please click here.