the Availability Digest

www.availabilitydigest.com
@availabilitydig

Build to Fail

November 2014

What do “build to fail” and “chaos monkey” have to do with continuous availability? Plenty, as Netflix has shown. Netflix survived a massive Amazon Web Services reboot of many of its virtual machines with hardly a hiccup. It attributed this success to its policy of building applications to run continuously on systems that can fail (build to fail) and to test these applications periodically with random system failures (chaos monkey).

Netflix

Netflix streams TV shows, movies, and other video content to almost 50 million subscribers around the world. It uses Amazon Web Services (AWS) as the platform to manage its many applications.

Netflix has taken great care in the design of its applications to ensure continuous availability. If it should go down, millions of its customers would be frustrated by not being able to view the shows they wished to see during the outage.

Netflix achieves this level of availability via two mechanisms – consistent reliability design patterns and periodic random failure injections.

Designing for Availability

The first step in achieving continuous availability is to design for availability. To do this, Netflix creates micro-services for the smallest level of abstraction to minimize the effect of any service failure. It utilizes consistent reliability design patterns to tie these micro-services into applications that are distributed across many nodes.

This integration is implemented via Hystrix, an open-source facility developed by Netflix in 2012. Hystrix is designed to control interactions between distributed services. It provides tolerance to latency between nodes and to failures of nodes. It isolates points of access between services and prevents failures from cascading across the access points. It provides fallback to redundant services should a service fail.

Designing for Failure

The second step is to verify that the application designs are reliable and that they will recover from unexpected failures. Netflix periodically injects random failures into its systems to ensure that they can tolerate failures. Following each test is a “no-blame post mortem” that seeks to improve the applications so that observed failures will not recur.

This process leads to improvement in its reliability design patterns.

Netflix's Use of Amazon Web Services

Netflix uses Amazon Web Services to host its applications. AWS is an ideal platform for Netflix because it provides multiple Availability Zones (AZs) that are fault-isolated. Applications may run in multiple AZs. If one AZ should experience problems, the applications can fail over to application instances in other AZs. Netflix runs its applications in three AZs to ensure availability.

A major component of the Netflix applications is its database. Among other items, the massive database holds the personalized content recommendations for individual subscribers based on their prior viewing history. Netflix is one of Amazon's largest customers.

Netflix uses Cassandra, a NoSQL database. Apache Cassandra is an open-source database management system in which the database can be distributed across many nodes, avoiding any single point of failure. Netflix chose Cassandra because of its capability to distribute a database, unlike other databases such as Oracle. If a node fails, the database is still intact on the surviving nodes and continues to be usable. Cassandra does not guarantee consistency among its database copies, but it does guarantee eventual consistency. That is, should updates be quiesced, all databases will eventually come to an identical state.

Within AWS, the Elastic Compute Cloud (EC2) is the primary infrastructure for the Amazon cloud, providing resizable compute resources. The fundamental building block of the Elastic Compute Cloud is an *EC2 instance*. An EC2 instance is an application running in the AWS cloud as a virtual machine. Multiple EC2 instances of an application can be run in the same or different Availability Zones.

Netflix's Cassandra database is distributed across 2,700 EC2 instances. It is designed to monitor for failed instances and to create a replacement instance should one fail.

The Amazon Reboot

In September, 2014, a serious vulnerability was discovered in the open-source Xen hypervisor that AWS uses to host its virtual machines. This vulnerability allowed one EC2 instance to read the memory of another EC2 instance hosted on the same physical server. This is a violation of an important security barrier in multi-tenant virtual environments. The vulnerability did not affect all EC2 instances. Only those hosted on x86 servers that did not use Xen's paravirtualization (PV) mode were affected.

It was imperative that Amazon correct this vulnerability immediately. Consequently, Amazon announced on September 25th that it would be updating its servers and that as many as 10% of its EC2 instances would require rebooting. Amazon did not identify the EC2 instances that would be affected, but while an EC2 instance was being rebooted, the services which it provided would be unavailable. Amazon scheduled the reboots so that only one Availability Zone was affected at a time.

Netflix immediately went on alert. A reboot of an EC2 virtual machine could potentially take down some of Netflix's services. Would its "built-to-fail" services survive this real-life test? This was the first serious test of the availability of Netflix's reliability design patterns.

As it turned out, 218 of Netflix's 2,700 Cassandra instances were rebooted. 22 instances did not reboot successfully, but Netflix's automated failure recovery mechanism replaced them with other EC2 instances which it created. Netflix experienced no downtime in any of its services during the Amazon reboot.

Chaos Engineering

Clouds have had a record of massive failures, including Amazon, Google, Azure, and Rackspace.[1] Netflix's experience demonstrates that companies thinking of moving to a cloud infrastructure need to build their systems to fail.

To ensure that their applications could withstand unexpected failures, Netflix repeatedly and regularly exposed them to unexpected failures. To do this, it built its own *Simian Army* ("Simian" means monkey or ape).

The Simian Army

Netflix's Simian Army is open-source software developed by Netflix that deliberately attempts to cause failures in a system. It is designed to identify groups of systems and to randomly disable one of the systems in the group. By periodically injecting controlled failures into its systems, Netflix guarantees that they are fault tolerant.

The first member of the Simian Army was Chaos Monkey. Chaos Monkey was launched in 2010. Its task is to randomly disable production EC2 instances in AWS. Next came Latency Monkey that induces delays in client-server interactions. Chaos Gorilla simulates an outage of an entire Amazon Availability Zone.

By using these virtual vandals, Netflix ensures that its automated systems cope with real-life failures. Its no-fault post mortems at the end of each exercise lead to improvements in its systems to prevent recurrences of problems found during the exercise.

The Simian Army operates only during business hours. Strange as it may seem to inject failures during peak hours, there are several reasons for this. One is that the Netflix peak hours tend to be early evening when people have returned home from work and want to watch a movie or a TV show. Equally important is that during business hours, key technical staff are available to fix any problems that have cropped up due to the test. Most of the time, no problems result from the failure injections – after all, the Netflix systems are supposed to be fault tolerant. However, if there is a problem, it is better to have it when staff are available rather than getting a call at 3 AM in the morning.

Summary

Netflix focuses on the continuous availability of its services. It achieves this level of availability by adhering to its reliability design patterns to architect its software. It then checks that these designs have in fact led to fault-tolerance by periodically injecting failures with its Simian Army.

The result of this effort was a near-perfect survival of the Amazon reboot of 10% of Netflix's database nodes.

[1] Amazon's Cloud Downed by Fat Finger, *Availability Digest*; May, 2011.
Amazon Downed by Memory Leak, *Availability Digest*; November 2012.
Google Troubles - A Case Study in Cloud Computing, *Availability Digest*; April 2009.
Poor Documentation Snags Google, *Availability Digest*; April 2010.
Rackspace - Another Hosting Service Bites the Dust, *Availability Digest*; December, 2007.
Windows Azure Cloud Succumbs to Leap Year, *Availability Digest*; March, 2012.
Windows Azure Downed by a Single Point of Failure, *Availability Digest*; November 2013.

Acknowledgements

Material for this article was taken from the following sources:

Introducing Hystrix for Resilience Engineering, *Netflix*; November 26, 2012.
Chaos Monkey, *GitHub*; February 27, 2014.
Introducing Chaos Engineering, *Netflix*; September 10, 2014.
Xen Project discloses serious vulnerability that impacts virtualized servers, *Computer World*; October 2, 2014.
Serious about the cloud? Follow Netflix's lead and get into chaos engineering, *TechRepublic*; October 3, 2014.
How Netflix survived the Amazon EC2 reboot, *PC World*; October 3, 2014.
Three lessons from Netflix on how to live in the cloud, *InfoWorld*; October 9, 2014.