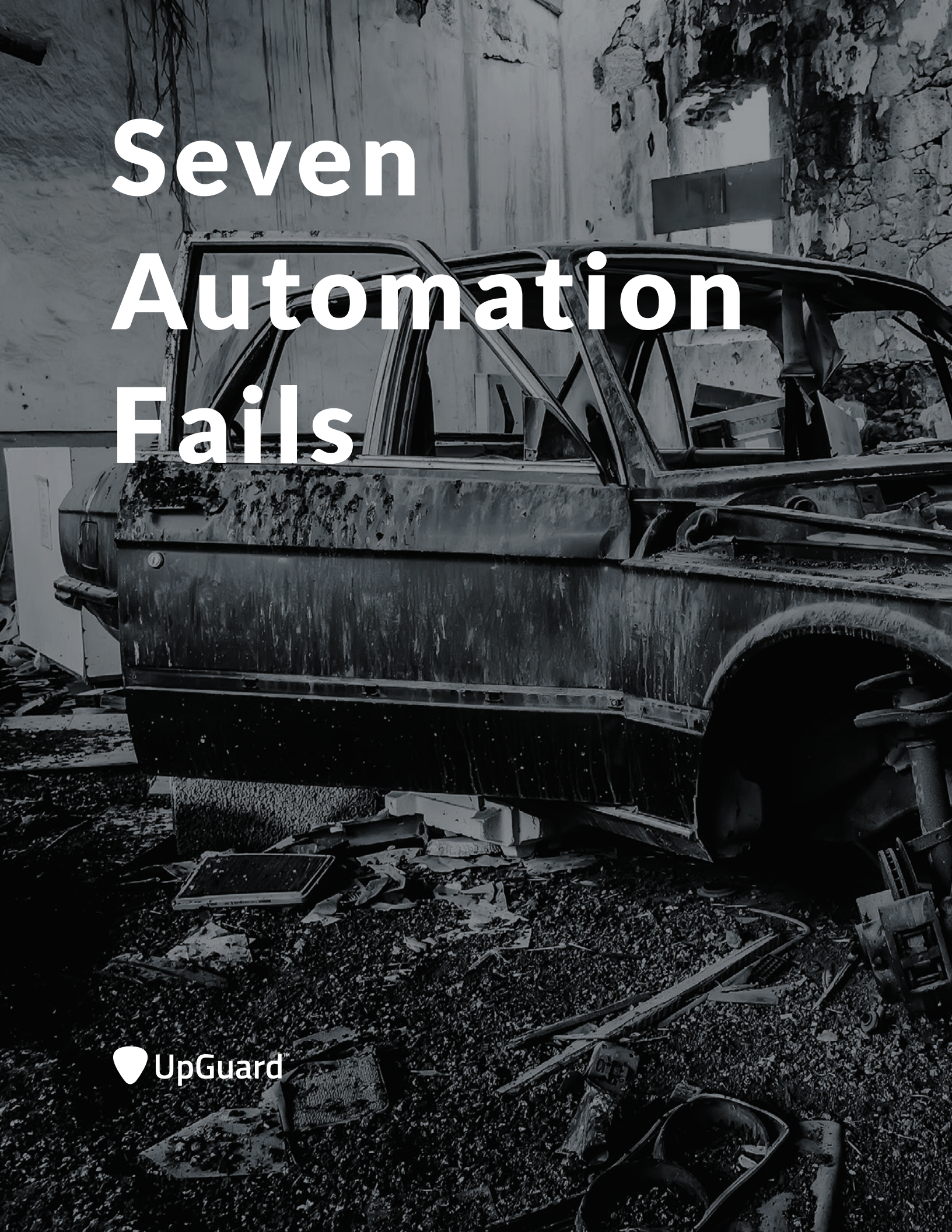


Seven Automation Fails

UpGuard™

CASE ONE

Opsmatic
Sep 24, 2014

It started with what should have been a simple Chef recipe:

"At 14:20 PDT Opsmatic went down in flames. The Chef run restarted every single instance in our infrastructure."

Automation often faces the same problems once contended with under traditional server administration. One such issue is the assumption, before making a change, that everything is the way it should be. When Opsmatic's routine server maintenance ended up shutting down their whole operation, it was because things weren't exactly as they had thought.

"After a bit more digging, we sorted out that chef-metal had been relying on the Ubuntu user being present on all our machines along with a specific private key. Something had caused the home directory for the Ubuntu user to be deleted."

But how could a standard user like this have been missing from production systems? Any system administrator can tell you the horror of finding out that something you did a long time ago as a test was not completely removed from the environment, later causing unexpected changes. In Opsmatic's case, a Chef recipe called "remove_default_users" had been created during the early stages of the company's AWS experimentation. Now, long after the test, that recipe was somehow still running against the production servers, unbeknownst to the staff maintaining them.

"The remove_default_users recipe was clearly dead weight; we had gotten a little sloppy and let a bit of invisible technical debt accumulate. In order to prevent the same thing happening again, we immediately deleted the recipe."

Like many major outages, this incident was the result of a long, causal sequence of mistakes, none of which were caught until they added up to a giant problem.

What if they had UpGuard...

If Opsmatic had instituted an UpGuard policy for which users should be on servers, it would have been immediately apparent that the Ubuntu user was missing from the production servers. A proactive alert would have been generated from UpGuard, informing the relevant people that the actual users on the servers did not match their expectations.

From here, the sysadmins would have began troubleshooting, and likely discovered the legacy Chef recipe that caused it, before anything had gone down unexpectedly. More importantly, they would have known that kicking off chef-metal would take down their servers, and could have avoided doing so until they had found the root cause of the problem among the Chef recipes.

CASE TWO

Knight Capital Aug 01, 2012

"SMARS is an automated, high speed, algorithmic router that sends orders into the market for execution."

Knight Capital not only automated its administrative IT processes, but algorithmic trading itself. This means that changes and unplanned errors happen very quickly, affecting the transit of real money. This is why in 2012 a single error caused Knight Capital to lose $172,222 per second for 45 minutes straight.

When operating a data center at scale, whole clusters of servers often run a single function. This distributes the load across more computing resources and provides better performance for high traffic applications. However, this model depends on all of the servers in the cluster (or grouping, if not an actual cluster) having the same configurations, so that applications behave the same -- no matter which particular server in the cluster they are using. But configurations, even if identical at provisioning, always drift apart.

"During the deployment of the new code, however, one of Knight's technicians did not copy the new code to one of the eight SMARS computer servers."

Despite all of its automation, Knight Capital was still manually deploying code across server banks. An inevitable human error caused one of its eight servers to have a different configuration from the others. From this point forward, the IT staff were operating under the misconception that these servers were identical, when in fact they were not.

Furthermore, decommissioned Power Peg code that Knight Capital had long abandoned sat available on the misconfigured server. Artifacts of previous experiments and methods often outlast their implementations, sitting dormant long after IT staff have forgotten about it.

"Orders sent with the repurposed flag to the eighth server triggered the defective Power Peg code still present on that server. As a result, this server began sending child orders to certain trading centers for execution."

Unfortunately for Knight, these misconfigurations were all tied to their core business processes—algorithmic stock trading—and when the triggering event, a repurposed flag, finally occurred, caught by the Power Peg code, money began changing hands immediately. The error ultimately cost Knight Capital $465 million dollars in trading loss.

What if they had UpGuard...

With UpGuard, Knight would have established a policy that the 8 SMARS servers had to be identical. They would have used a known good configuration of those servers to generate the policy by which all of other assets are judged. When the eighth server was overlooked by the sysadmin, the policy would have failed, and the error would have been caught before the triggering event occurred. Furthermore, UpGuard would have detected the presence of the defunct Power Peg code, alerting IT staff to the potential security risk posed by such code before it ever executed.

CASE THREE

Delta Airlines Jan 29, 2017

"Delta teams are expeditiously working to fix a systems outage that has resulted in departure delays for flights on the ground."

Large logistics operations rely on automated systems to achieve the necessary speed to perform at scale. Ideally, these automation systems produce resilient services. But airline companies have struggled in recent years to keep those systems functional, costing them hundreds of millions of dollars and customer goodwill.

"Delta Air Lines grounded its domestic flights for hours late Sunday due to what was described as automation issues."

Such dysfunction is possible because automation is not a solution to operational problems, but merely a way to make them happen faster and at a larger scale. Just like traditional, manual methods of systems administration, automated systems suffer from misconfigurations.

"Delta told investors the glitch cost the airline more than $150 million."

When these misconfigurations occur, entire systems go down, because the changes are pushed out quickly through the automated mechanisms. This interrupts flight operations, delays planes, and siphons money out of the business.

What if they had UpGuard...

For years, UpGuard has helped businesses deploy automation effectively, by first answering key operational questions that often fly under the radar: do you know what you have, and can you control it?

With better visibility into their operations, Delta could reduce the risk to their business when making planned changes. Furthermore, if they understood which unplanned changes were happening in their environment, they would be able to shore up any misconfigurations before they cascade into a major outage.

CASE FOUR

Gmail
Jan 24, 2014

Even technology giants have the occasional automation related outage. An hour of downtime might not seem like a lot, but when it's Google on the line, that hour is highly visible.

"Thanks to a bug in "an internal system that generates configuration", Gmail was down for at least 20 (and up to 50) minutes on Friday."

Google has always been on the bleeding edge of technology, so it's no surprise that they automate configuration management. When they make a change, they have to make that change across thousands of servers. However, when the wrong change is executed, it also propagates far and wide within a matter of seconds.

"The incorrect configuration was sent to live services..., caused users' requests for their data to be ignored, and those services, in turn, generated errors."

The lesson here is that configuration automation is not the same as configuration management. Automation ensures that changes get pushed out across all systems. It does not necessarily ensure that those changes are good.

Understanding whether or not a change is good can be more difficult than it sounds. With the increasing complexity of modern data centers and the hybridization between cloud platforms, on-site servers, SaaS providers, and other technologies, getting a handle on what exactly is happening—and whether or not it's supposed to be happening—requires technology that can glean insights from a massive data set and present those insights to the business so that they can be acted upon quickly.

What if they had UpGuard...

For years, UpGuard has helped businesses deploy automation effectively, by first answering key operational questions that often fly under the radar: do you know what you have, and can you control it?

With better visibility into their operations, Delta could reduce the risk to their business when making planned changes. Furthermore, if they understood which unplanned changes were happening in their environment, they would be able to shore up any misconfigurations before they cascade into a major outage.

CASE FIVE

Dropbox
Jan 10, 2014

"We use thousands of databases to run Dropbox. Each database has one master and two replica machines for redundancy. In addition, we perform full and incremental data backups and store them in a separate environment."

Dropbox went down for three hours when a planned upgrade went awry. A "buggy script" caused updates to be applied to production servers, causing live services to fail.

"A subtle bug in the script caused the command to reinstall a small number of active machines. Unfortunately, some master-slave pairs were impacted which resulted in the site going down."

Fortunately for Dropbox, their backup and recovery strategy allowed them to restore most services within three hours. However, no IT team likes to scramble under an outage and race to implement emergency procedures that should work, in theory.

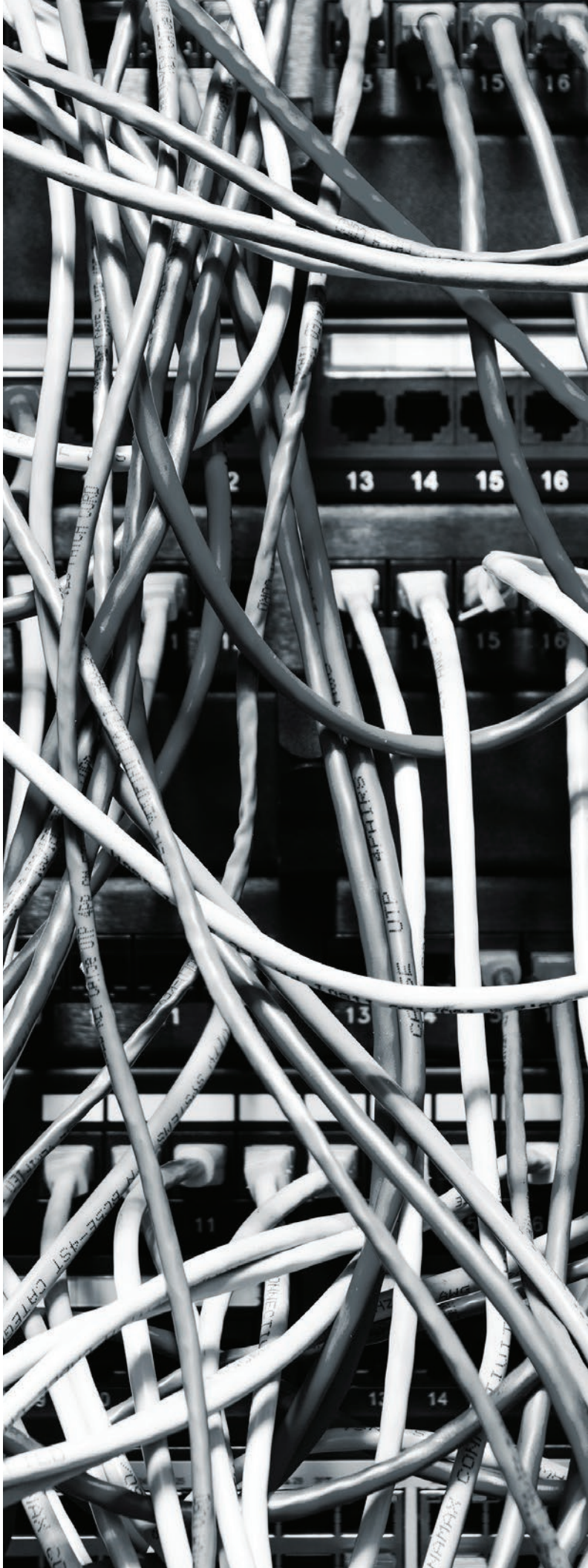
"To restore service as fast as possible, we performed the recovery from our backups. We were able to restore most functionality within 3 hours, but the large size of some of our databases slowed recovery, and it took until 4:40 PM PT today for core service to fully return."

But could the outage have been prevented altogether? If we look at the conclusions reached by Dropbox in their post mortem, we can see that continuous, improved state verification was the first key practice mentioned to avoid this type of outage in the future.

"We've added an additional layer of checks that require machines to locally verify their state before executing incoming commands. This enables machines that self-identify as running critical processes to refuse potentially destructive operations."

What if they had UpGuard...

UpGuard provides advanced state verification at the largest scales and unlike homebrew scripts or domain specific languages, we've created a platform where the whole team can see the distributed state of a whole system in one place.



CASE SIX

AWS & DynamoDB Sep 20, 2015

"On Sunday, at 2:19 AM PDT, there was a brief network disruption that impacted a portion of DynamoDB's storage servers. Normally, this type of networking disruption is handled seamlessly and without change to the performance of DynamoDB. But, on Sunday morning, a portion of the metadata service responses exceeded the retrieval and transmission time allowed by storage servers. As a result, some of the storage servers were unable to obtain their membership data, and removed themselves from taking requests."

Amazon's AWS cloud platform suffered an outage that cascaded from a simple network disruption into broad service failure when some automated processes timed out. Amazon has built a very advanced, integrated cloud platform, but still suffers from network outages like traditional on-premise data centers.

"When the network disruption occurred on Sunday morning, and a number of storage servers simultaneously requested their membership data, the metadata service was processing some membership lists that were now large enough that their processing time was near the time limit for retrieval. Multiple, simultaneous requests for these large memberships caused processing to slow further and eventually exceed the allotted time limit. This resulted in the disrupted storage servers failing to complete their membership renewal, becoming unavailable for requests, and retrying these requests."

While it was a DynamoDB timeout issue that ultimately caused the problem users experienced, the network disruption was the precipitating incident that kicked off the whole event. Digital services depend on digital infrastructure, just as physical services, such as freight haulage, requires physical infrastructure, such as roads.

The service and everything it depends on to function successfully form a single system of risk that must be analyzed and maintained to ensure the service works.

What if they had UpGuard...

Amazon, like Google, has an extremely advanced, large scale infrastructure, and nonetheless still has unexpected outages that impact business. UpGuard helps IT teams build resilience against the risks posed by digital infrastructure. We scan network devices (and everything else) to ensure that they meet business expectations at all times. When a change violates those expectations, we let you know, so you can fix it before it becomes an outage.

CASE SEVEN

Healthcare.gov Oct 01, 2013

"During the two years before the disastrous opening of HealthCare.gov, federal officials in charge of creating the online insurance marketplace received 18 written warnings that the mammoth project was mismanaged and off course but never considered postponing its launch."

The government experienced the fallout from a major software failure when they rolled out Healthcare.gov, the Affordable Care Act (ACA) web enrollment tool. Everyone knew that this was a massive undertaking and that the stakes were high, as the delivery of many people's healthcare insurance was on the line. But the project had significant problems all the way up to the release date, and beyond, due to a lack of integration, visibility, and testing.

"In the summer of 2013, CMS officials asked CGI Federal, the main contractor building HealthCare.gov, to demonstrate a simple feature called Account Lite, intended to let consumers create accounts before enrollment began. CGI was behind schedule and, when it finally did the demonstration, federal workers found 105 defects."

One crucial piece of Healthcare.gov was Account Lite, the mechanism by which people would create their accounts and gain access to their healthcare options. This particular module had so many problems that it was clear to almost everyone that they had a disaster waiting to happen. Yet contractors moved forward anyway.

"Before the site opened, CMS had not tested it end to end to see how the parts worked together. "You can't test what is not built," a contractor told the investigators."

Modern software delivery methods have evolved to address these exact problems. The DevOps movement was formed to unify silos that had formed among the various subgroups of an IT department, with all eyes focused on achieving the big picture result. As Healthcare.gov had huge, complex deliverables which took a long time to complete, the testing phase was completely ignored until the very last minute.

"HealthCare.gov crashed within two hours of its launch."

Unsurprisingly, the software release failed miserably, and millions of people were unable to sign up for their healthcare. This outage had political implications, with critics of the ACA using the outage as evidence of the administration's incompetence in delivering a healthcare program. The site was eventually stabilized, but only after all the work that should have been integrated before the release was accomplished—after the crash.

What if they had UpGuard...

UpGuard validates every piece of the DevOps toolchain. Automating software delivery requires speed and precision, as well as continuous testing so that feedback loops are as small as possible. Shortened release cycles break obstacles down into smaller pieces and help bugs and other problems stand out and get fixed continuously inside the development cycle, instead of the day of release, in one lump sum.