acmqueue

Weathering the Unexpected

Failures happen, and resilience drills help organizations prepare for them.

Kripa Krishnan, Google

Whether it is a hurricane blowing down power lines, a volcanic-ash cloud grounding all flights for a continent, or a humble rodent gnawing through underground fibers—the unexpected happens. We cannot do much to prevent it, but there is a lot we can do to be prepared for it. To this end, Google runs an annual, company-wide, multi-day Disaster Recovery Testing event—DiRT—the objective of which is to ensure that Google’s services and internal business operations continue to run following a disaster.

DiRT was developed to find vulnerabilities in critical systems and business processes by intentionally causing failures in them, and to fix them before such failures happen in an uncontrolled manner. DiRT tests both Google’s technical robustness, by breaking live systems, and our operational resilience by explicitly preventing critical personnel, area experts, and leaders from participating. Where we are not resilient but should be, we try to fix it. (See the sidebar, “Google DiRT: The view from someone being tested,” by Tom Limoncelli.)

SETTING EXPECTATIONS

For DiRT-style events to be successful, an organization first needs to accept system and process failures as a means of learning. Things *will* go wrong. When they do, the focus needs to be on fixing the error instead of reprimanding an individual or team for a failure of complex systems.

An organization also needs to believe that the value in learning from events like DiRT justifies the costs. These events are not cheap—they require a sizable engineering investment, are accompanied by considerable disruptions to productivity, and can cause user-facing issues or revenue loss. DiRT, for example, involves the work of hundreds of engineering and operations personnel over several days, and things don’t always go according to plan. DiRT has caused accidental outages and in some cases revenue loss. Since DiRT is a company-wide exercise, however, it has the benefit of having all the right people available at a moment’s notice to contain such events should they arise.

However, to benefit the most from such recovery events, an organization also needs to invest in continuous testing of its services. Large, DiRT-style, company-wide events should be less about testing routine failure conditions such as single-service failovers or on-call handoffs, and more about testing complex scenarios or less-tested interfaces between systems and teams. Complex failures are often merely a result of weaknesses in smaller parts of the system. As smaller components of the system get tested constantly, failures of larger components become less likely.

A simple example is testing an organization’s ability to recover from the loss of a data center. Such a loss may be simulated by powering down the facility or by causing network links to fail. The response would theoretically involve a sequence of events, from redirecting traffic away from the lost data center to a series of single-service failovers in some specific order. All it would take to choke the recovery process, however, is the failure of a single instance of a core infrastructure service—such as

DNS (Domain Name System) or LDAP (Lightweight Directory Access Protocol)—to failover. Testing the failover of such a service can and should happen continuously and should not have to wait for a DiRT event.

GROWING THE PROGRAM

A good way to kick off such an exercise is to start small and let the exercise evolve. It is quite easy to make this a large and complex affair right from the start, but doing so would probably come with unexpected overhead and complications.

Starting small applies not only to the number of teams involved in the exercise, but to the complexity of the tests. A few easy-to-remember rules and a simple, repeatable exercise format go a long way toward engaging teams quickly. If not all teams buy in, then work with the few that do, and as the exercise proves itself useful, more teams will participate. (Schwag won't hurt either!)

Google's experience serves as an example: DiRT in its original form focused only on testing critical user-facing services. The initial bar was that all major user-facing teams wrote tests and that the tests were safe and caused no disruption, although we did realize that some of the tests were not very useful. This got teams "playing." Over a few iterations, the exercise attracted many more teams and included fewer low-quality/low-value tests.

The same can be said for test designs. While the quality of tests matters a lot and directly affects the value of the exercise, Disaster Recovery Testing events do not have to begin with overly complicated tests or the perfect set of tests (which doesn't exist). DiRT started with individual groups testing failure scenarios specific to their services. The overarching "disaster" was merely theoretical. In a subsequent DiRT exercise, the first major outage tested was that of our primary source-control management servers, which exposed several nonreplicated critical functions dependent on this system. As each piece was fixed, we progressed to a larger disaster involving a major "earthquake" in the Bay Area.

We simulated the earthquake by taking down a data center in the area that housed a number of our internal systems. While the outage uncovered several services that were singly homed, it also exposed other interesting dependencies. For example, to avoid being affected by the outage, some teams decided to failover services from the data center to their workstations. Since the "earthquake" occurred near Google headquarters in Mountain View, the testing team disconnected the Mountain View campus as well—which meant all these failovers had failed. Also, what many people didn't anticipate was that the data-center outage caused authentication systems to fail in unexpected ways, which in turn locked most teams out of their workstations.

When the engineers realized that the shortcuts had failed and that no one could get any work done, they all simultaneously decided it was a good time to get dinner, and we ended up DoS'ing our cafes. In keeping with the DiRT goals, several of these issues were fixed by the next test.

Today, production and internal systems, network and data-center operations, and several business units such as HR, finance, security, and facilities test during DiRT. In the most recent DiRT exercise, we brought down several data-center clusters, infrastructure hubs, and offices without notice. Most of the scenarios were resolved painlessly.

It is very important to mention that *well* before Google even considered the concept of DiRT, most operations teams were already continuously testing their systems and cross-training using formats based on popular role-playing games. As issues were identified, fixes were folded into the design

process. For many of these teams, DiRT merely provided a safe opportunity to test riskier failure conditions or less-tested interactions with other systems and teams.

WHAT TO TEST

There are several angles to consider when designing tests for DiRT. The simplest case, as described earlier, is service-specific testing. This category tests that a service and its components are fault-tolerant. These tests are usually contained, needing only the immediate team to respond, and they uncover technical and operational issues including documentation gaps, stale configurations, or knowledge gaps in handling critical emergencies. Ideally, these tests become part of the service's continuous testing process.

More involved technical test cases create scenarios that cause multiple system failures in parallel. Examples include data-center outages, fiber cuts, or failures in core infrastructure that manifest in dependent services. Such tests have a lot more value if the team that designs them is cross-functional and incorporates technical leads and subject-matter experts from various areas in the company. These are the people who understand the intricacies of their services and are in excellent positions to enumerate dependencies and failure modes in order to design realistic and meaningful scenarios.

The goal of this category of tests is to identify weaknesses in the less-tested interfaces between services and teams. Such scenarios can be potentially risky and disruptive, and they may need the help of several teams to resolve the error condition. DiRT is an excellent platform for this category of testing since it is meant to be a companywide exercise and all teams necessary for issue resolution are available on demand.

An often-overlooked area of testing is business process and communications. Systems and processes are highly intertwined, and separating testing of systems from testing of business processes isn't realistic: a failure of a business system will affect the business process, and conversely a working system is not very useful without the right personnel.

The previous "earthquake" scenario exposed several such examples, some of which are described here:

The loss of the Bay Area disconnected both people *and* systems in Mountain View from the rest of the world. This meant that teams in geographically-distributed offices needed to provide round-the-clock on-call coverage for critical operations. The configuration change that was needed to redirect alerts and pages to these offices, however, depended on a system that was affected by the outage. Even for teams with fully global expertise, things did not go smoothly as a result of this process failure.

A more successful failover was an approvals-tracking system for internal business functions. The system on its own was useless, however, since all the critical approvers were in Mountain View and therefore unavailable. Unfortunately, they were also the only people who had the ability to change the approval chain.

In the same scenario, we tested the use of a documented emergency communications plan. The first DiRT exercise revealed that exactly one person was able to find the plan and show up on the correct phone bridge at the time of the exercise. During the following drill, more than 100 people were able to find it. This is when we learned the bridge wouldn't hold more than 40 callers. During another call, one of the callers put the bridge on hold. While the hold music was excellent for the soul, we quickly learned we needed ways to boot people from the bridge.

As another example, we simulated a long-term power outage at a data center. This test challenged the facility to run on backup generator power for an extended period, which in turn required the purchase of considerable amounts of diesel fuel without access to the usual chain of approvers at HQ. We expected someone in the facility to invoke our documented emergency spending process, but since they didn't know where that was, the test takers creatively found an employee who offered to put the entire six-digit charge on his personal credit card. Copious documentation on how something should work doesn't mean anyone will use it, or that it will work if they do. The only way to make sure is through testing.

Of course, tests are of almost no value if no effort is put into fixing the problems that surface during the tests. An organizational culture that embraces failure as a means of learning goes a long way toward getting teams both to find and to resolve issues in their systems routinely.

RISK MITIGATION

DiRT tests can be disruptive, and failures should be expected to occur at any point. Several steps can be taken to minimize potential damage.

At minimum, all tests need to be thoroughly reviewed by a cross-functional technical team and accompanied by a plan to revert should things go wrong. If the test has never been attempted before, running it in a sandbox can help contain the effects. The flip side to sandboxing, though, is that these environments may have configurations that are significantly different from those in production, resulting in less realistic outcomes.

There are ways of testing without disrupting services: at Google, we “whitelist” services we already know won't be able to survive certain tests. In essence, they have already failed the test and there is no point in causing an outage for them when the failing condition is already well understood. While services can “prefail” and exempt themselves, there is no concept of “prepassing” the test—services have to make it through to “pass.”

A centrally staffed command center that understands and monitors all the tests going on at any given time makes DiRT a safer environment for testing. When the unforeseen happens, the team in the command center (made up largely of technical experts in various areas) jumps in to revert the test or fix the offending issue.

THE TEAM

At DiRT's core are two teams: a technical team and a coordination team.

The technical team is responsible for designing all major tests and evaluating all tests written by individual teams for quality and impact. The technical team is also responsible for actually causing the larger outages and monitoring them to make sure things don't go awry. This is also the team that handles unforeseen side effects of tests.

The coordinators handle a large part of the planning, scheduling, and execution of tests. They work very closely with the technical team to make sure that the tests do not conflict with each other and that preparation (such as setting up sandboxes) for each of these tests is done ahead of DiRT.

Both teams populate the DiRT command center. At the helm is usually someone with a sufficiently large Rolodex. When not much is going on, the command center is filled with distractions; it houses very smart people with short attention spans who are low on sleep and high on caffeine. When things go wrong, however—and they do—they are alert, on target, and

fully focused on firefighting and getting the error communicated, resolved, or rolled back—and, furthermore, filed for fixing.

The command center is also home to the person with one of the most fun 20-percent projects at Google: the storyteller who concocts and narrates the disaster, ranging from the attack of the zombies to a bizarre psychological thriller featuring an errant fortune-teller.

CONCLUSION

Whatever their flavor, Disaster Recovery Testing events are an excellent vehicle to find issues in systems and processes in a controlled environment. The basic principle is to accept that failures happen and that organizations need to be prepared for them. Often, a solid executive sponsor and champion is instrumental in setting the right tone for the exercise. In Google's case, VP of operations Ben Treynor has championed both learning from continuous testing and preemptively fixing failures.

It is true that these exercises require a lot of work, but there is inestimable value in having the chance to identify and fix failures before they occur in an uncontrolled environment.

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

KRIPA KRISHNAN is a technical program manager at Google who has been running the company's disaster recovery program (DiRT) for six years. She also leads the Google Apps for Government effort and has worked in other areas as well, including storage and billing. She is currently working on privacy and security infrastructure initiatives for Google Apps. Prior to Google, Kripa worked with the Telemedicine Program of Kosovo to set up telemedicine infrastructure and a virtual education network in the region. In a previous life, she ran a theater and performing arts organization in India for several years.

Google DiRT: The View from Someone being Tested

There's no telling where the zombies might strike next.

Thomas A. Limoncelli, Google

This is a fictionalized account of a Google DiRT (Disaster Recovery Testing) exercise as seen from the perspective of the engineers responsible for running the services being tested. The names, location, and situation have been changed.

[Phone rings]

ME Hello?

MARY Hi, Tom. I'm proctoring a DiRT exercise. You are on call for [name of service], right?

ME I am.

MARY In this exercise we pretend the [name of service] database needs to be restored from backups.

ME OK. Is this a live exercise?

MARY No, just talk me through it.

ME Well, I'd follow the directions in our operational docs.

MARY Can you find the doc?

[A couple of key clicks later]

ME Yes, I have it here.

MARY OK, bring up a clone of the service and restore the database to it.

Over the next few minutes, I make two discoveries. First, one of the commands in the document now requires additional parameters. Second, the temporary area used to do the restore does not have enough space. It had enough space when the procedure was written, but the database has grown since then.

Mary files a bug report to request that the document be updated. She also files a bug report to set up a process to prevent the disk-space situation from happening.

I check my e-mail and see the notifications from our bug database. The bugs are cc:ed to me and are tagged as being part of *DiRT2011*. Everything with that tag will be watched by various parties to make sure it gets attention over the next few months. I fix the first bug while waiting for the restore to complete.

The second bug will take more time. We'll need to add the restore area to our quarterly resource estimation and allocation process. Plus, we'll add some rules to our monitoring system to detect whether the database size is nearing the size of the restore area.

ME OK, the service's backup has been read. I'm running a clone of the service on it, and I'm sending you an instant message with an URL you can use to access it.

[A couple of key clicks later]

MARY OK, I can access the data. It looks good. Congrats!

ME Thanks!

MARY Well, I'll leave you to your work. Oh, and I'm not supposed to tell you this, but at 2 p.m. there will be some... fun.

ME You know my on-call shift ends at 3 p.m., right? If you happen to be delayed an hour...

MARY No such luck. I'm in California and 3 p.m. your time is when I'll be leaving for lunch.

A minute after the exercise is over I receive an e-mail message with a link to a post-exercise document. I update it with what happened, links to the bugs that were filed, and so on. I also think of a few other ways of improving the process and document them, filing feature requests in our bug database for each of them.

At 2 p.m. my pager doesn't go off, but I see on my dashboard that there is an outage in Georgia. Everyone in our internal chat room is talking about it. I'm not too concerned. Our service runs out of four data centers around the world, and the system has automatically redirected Web requests to the other three locations.

The transition is flawless, losing only the queries that were "in flight," which is well within

our SLA (service-level agreement).

A new e-mail appears in my inbox explaining that zombies have invaded Georgia and are trying to eat the brains of the data-center technicians there. The zombies have severed the network connections to the data center. No network traffic is going in or out. Lastly, the e-mail points out that this is part of a DiRT exercise and no actual technicians have had their brains eaten, but the network connections really have been disabled.

[Again, phone rings]

MARY Hi! Having fun yet?

ME I'm always having fun. But I guess you mean the Georgia outage?

MARY Yup. Shame about those technicians.

ME Well, I know a lot of them and they have big brains. Those zombies will feed for hours.

MARY Is your service still within SLA?

I look at my dashboard and see that with three data centers doing the work normally distributed to four locations the latency has increased slightly, but it is within SLA. The truth is that I don't need to look at my dashboard because I would have gotten paged if the latency was unacceptable (or growing at a rate that would reach an unacceptable level if left unchecked).

ME Everything is fine.

MARY Great, because I'm here to proctor another test.

ME Isn't a horde of zombies enough?

MARY Not in my book. You see, your SLA says that your service is supposed to be able to survive two data-center outages at the same time.

She is correct. Our company standard is to be able to survive two outages at the same time. The reason is simple. Data centers and services need to be able to be taken down occasionally for planned maintenance. During this window of time another data center might go down for unplanned reasons (such as a zombie attack). The ability to survive two simultaneous outages is called N+2 redundancy.

ME So what do you want me to do?

MARY Pretend the data center in Europe is going down for scheduled preventive maintenance.

I follow our procedure and temporarily shut down the service in Europe. Web traffic from our European customers distributes itself over the remaining two data centers. Since this is an orderly shutdown, zero queries are lost.

ME Done!

MARY Are you within the SLA?

I look at the dashboard and see that the latency has increased further. The entire service is running on the two smaller data centers. Each of the two down data centers is bigger than the combined, smaller, working data centers; yet, there is enough capacity to handle this situation.

ME We're just barely within the SLA.

MARY Congrats. You pass. You may bring the service up in the European data center.

I decide to file a bug, anyway. We stayed within the SLA, but it was too close for comfort. Certainly we can do better.

I look at my clock and see that it is almost 3 p.m. I finish filling out the post-exercise document just as the next on-call person comes online. I send her an instant message to explain what she missed.

I also remind her to keep her office door locked. There's no telling where the zombies might strike next.

TOM LIMONCELLI is a site reliability engineer in Google's New York office.

© 2012 ACM 1542-7730/12/0900 \$10.00