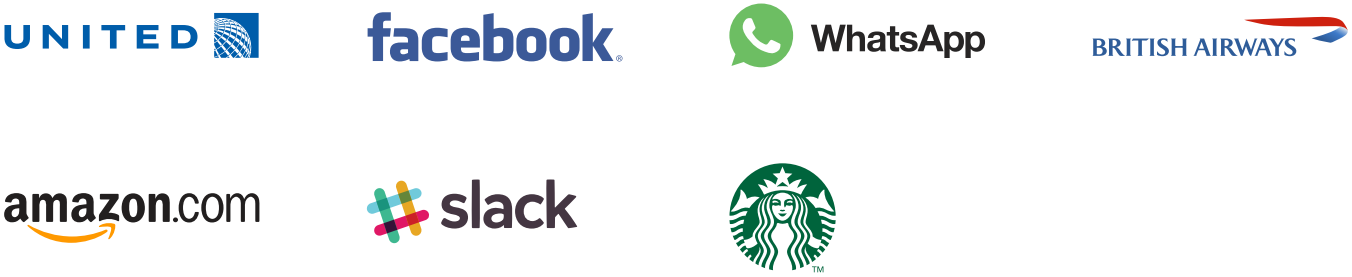
Chaos Engineering: Finding Failures Before They Become Outages

Diane Glazman will never fly British Airways (BA) again. Glazman and her husband were among the 75,000 people affected by the three-day BA system failure summer 2017. On their way from San Francisco to their son's college graduation in Edinburgh, they were stranded in London—without their luggage—at the beginning of what was to be the three-week dream tour of Scotland. "Listening to the excuses was frustrating because nothing explained why BA was so unprepared for such a catastrophic failure," says Glazman.

BA lost an estimated $135 million due to that outage. The culprit turned out to be a faulty uninterruptable power supply device (UPS)—the corporate cousin to the $10 gadget you can find in your corner Radio Shack. And that loss figure doesn't count the forever-gone trust of customers like Glazman, who will look elsewhere for transatlantic flights next time she travels.

BA of course isn't alone for having suffered financially for having its systems down. There were also United Airlines (200 flights delayed for 2.5 hours, thousands of passengers stranded or missed connections), Starbucks (couldn't accept any payments but cash in affected stores), Facebook (millions of users offline and tens of millions of ads not served during the 2.5 hours of downtime), and WhatsApp (600 million users affected, 5 billion messages lost). And when Amazon S3 went down in March 2017, it collectively cost Amazon's customers $150 million.

Suffered major outages in 2017



In fact, 2017 was a banner year for systems outages—and for the cost of them.

The 2017 ITIC Cost of Downtime survey finds that 98% of organizations say a single hour of downtime costs more than $100,000. More than eight in 10 companies indicated that 60 minutes of downtime costs their business more $300,000. And a record one-third of enterprises report that one hour of downtime costs their firms $1 million to more than $5 million (see Figure 1). The average cost of a single hour of unplanned downtime has risen by 25% to 30% since 2008 when ITIC first began tracking these figures.

% of Companies Reporting	Cost of 1hr of Downtime
33%	$1M – $5M+
80%	$300,000+

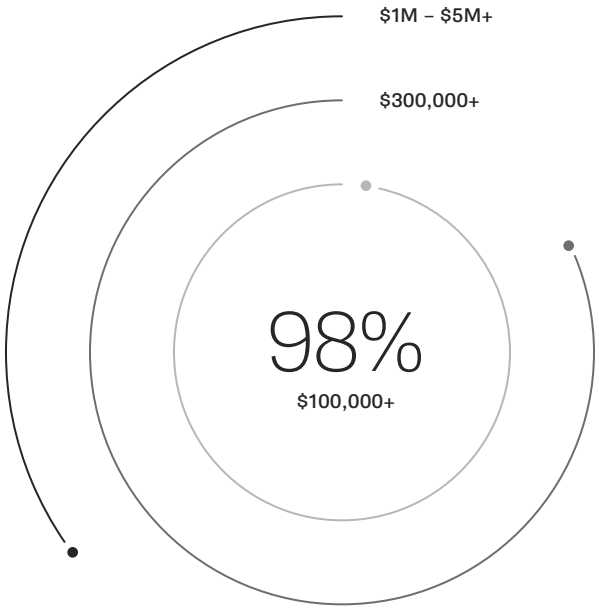


Figure 1: Cost of 60 minutes of downtime

So how can organizations cut the risk of downtime? The answer: break your systems on purpose. Find out their weaknesses and fix them before they break when least expected.

Break your systems on purpose. Find out their weaknesses and fix them before they break when least expected.

It's called chaos engineering, and it's being adopted by leading financial institutions, internet companies, and manufacturing firms throughout the world. Such businesses understand that the trillions of dollars lost annually due to downtime is not acceptable to their customers, their stockholders, and their employees.

A more complex, distributed world

In the traditional corporate computer environment of 30 years ago, software ran in a highly controlled environment that had few moving parts or variables.

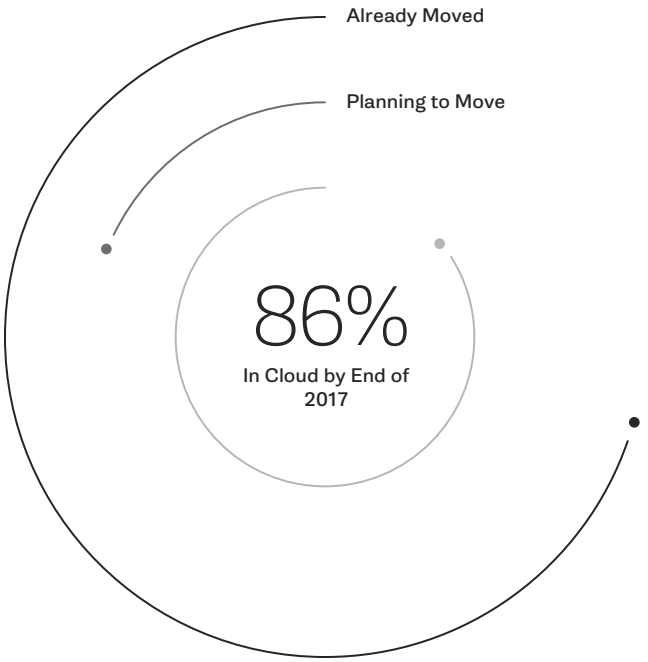
But in the new business world that depends on the internet, globally connected systems, a mix of cloud and bare-metal infrastructure, and more moving parts than you can count, your software depends on a lot of infrastructure and services that are outside your control to run smoothly.

70%

Already Moved to the Cloud

16%

Planning to Move



First and foremost, there's the rapid shift to the cloud. A full 70% of companies have already moved at least one application to the cloud, according to IDC, with 16% more planning to do so by the end of 2017.

Then there's the rise of microservices. Most enterprises are developing their software today on a microservice architecture. Applications are built as small and independent but interconnected modular services. Each service runs a unique process meant to meet a particular business goal. For example, one microservice might track inventory levels of products. Another might handle serving personalized recommendations to customers.

Then there are all the web servers, databases, load balancers, routers, and more that must work together to form a coherent whole. This is not easy.

The good news is that as this modular, distributed infrastructure continues to evolve, businesses can do things with software that simply weren't possible before. By shifting to what are also called "loosely coupled services" that can be developed and released independently of each other, developed by similarly organized teams who are empowered to make changes, time to market for businesses can be radically reduced.

But these new capabilities come at a price.

Today, businesses face a serious “complexity gap.” It’s difficult for even the most technically astute individuals to understand how all the different bits and pieces work together. The ability of your IT professionals to manage the ever-evolving sophistication of computer infrastructure falls short of what’s needed. With so many points of possible failure in your systems, this makes your business extremely vulnerable.

This is happening at a time when people—businesspeople as well as consumers—are increasingly dependent on the internet and the services that corporations deliver that depend on it. The number of internet users worldwide in 2017 is 3.58 billion, up from 3.39 billion in 2016.

53% of mobile users abandon webpages that take more than three seconds to load.

Even fewer people today can operate without a phone. In 2017 the number of mobile phones reached 4.77 billion, and is expected to pass the five billion mark by 2019. And all these users--both consumers and business--are demanding: Doubleclick found in 2016 that 53% of mobile site visits are abandoned if a web page takes longer than three seconds to load.

As a result, the stakes have never been higher for companies to maintain the uptime of their systems.

Chaos engineering, a primer

Imagine that it's 1796, and you have been selected to be injected with Edward Jenner's brand-new smallpox vaccine. You are told that he is going to put the actual virus into your bloodstream. You are also told that you won't get sick. Instead, it will make it impossible for you to get sick with this particular virus, because it will make your system stronger. You might have recoiled, thinking that the risk was too great. Yet you allow yourself to be vaccinated. And, of course, you are much better off than those who refuse the treatment.

By injecting a system with something that has the potential to disrupt it, you can identify where the system may be weak, and can take steps to make it more resilient.

This is exactly what chaos engineering does. All computers have limits, and possible points of failure. By injecting a system with something that has the potential to disrupt it, you can identify where the system may be weak, and can take steps to make it more resilient.

That covers the systems under your purview. Problems that occur with your cloud service providers are out of your control, and you can't resolve outages by adding extra boxes or power supplies.

Think of all your data living in the cloud, in Amazon S3 or DynamoDB, and the hosted services you depend on, such as Salesforce or Workday. If they fail, you're at their mercy. Chaos engineering isn't just essential for your applications, it's essential for the companies behind those applications, which is why Netflix, Uber, and Amazon all have teams dedicated to chaos and reliability: they know they cannot afford to let their customers down.

Here's where chaos engineering comes in: you know you have these potential points of failure and vulnerabilities. So why wait until there's a problem?

Imagine attempting to break your systems. On purpose. Before they fail on their own. Because that is what chaos engineering does. By triggering failures intentionally in a controlled way, you gain confidence that your systems can deal with those failures before they occur in production.

The goal of chaos engineering is to teach you something new about your systems' vulnerabilities by performing experiments on them. You seek to identify hidden problems that could arise in production prior to them causing an outage. Only then will you be able to address systemic weaknesses and make your systems fault-tolerant.

Critical to chaos engineering is that it is treated as a scientific discipline. It uses precise engineering processes to work. Four steps in particular are followed.

1. **Form a hypothesis:** Ask yourself, "What could go wrong?"

2. **Plan your experiment:** Determine how you can recreate that problem in a safe way that won't impact users (internal or external).

3. **Minimize the blast radius:** Start with the smallest experiment that will teach you something.

4. **Run the experiment:** Make sure to carefully observe the results.

5. **Celebrate the outcome:** If things didn't work as they should, you found a bug! Success! If everything went as planned, increase the blast radius and start over at #1.

6. **Complete the mission:** You're done once you have run the experiment at full scale in production, and everything works as expected.

Some examples of what you might do to the hypothetic system when performing a chaos engineering experiment:

- **Reboot or halt the host operating system.** This allows you to test things like how your system reacts when losing one or more cluster machines.
- **Change the host's system time.** This can be used to test your system's capability to adjust to daylight saving time and other time-related events.
- **Simulate an attack that kills a process.** This can be used to simulate application or dependency crashes.

The point of simulating potentially catastrophic events is to make them non-events that are irrelevant to our infrastructure's ability to perform as required.

Naturally, you immediately address any potential problems that you uncover with chaos engineering. Indeed, the point of simulating potentially catastrophic events is to make them non-events that are irrelevant to our infrastructure's ability to perform as required.

Chaos engineering differs from the regular testing that everyone does as a matter of course in several ways. Normal testing is done during build / compile activities, and doesn't test for different configurations or behaviors or factors beyond your control. Additionally, routine testing doesn't account for people--for training and preparing them for the failures they will be responsible for fixing live, in the middle of the night.

Benefits of chaos engineering

Companies like Amazon, Netflix, Salesforce, and Uber have been using chaos engineering for years to make their systems more reliable. For internet companies whose very existence depends on their ability to be "up" at all times, chaos engineering was a necessity. Now businesses in other industries—financial services in particular—are starting to follow suit, and implement chaos engineering programs of their own.

The benefits of chaos engineering include the following:

- Help technology professionals see how systems behave in the face of failure, as their assumptions are often incomplete or inaccurate
- Validate that hypothetical defenses against failure will work when needed by exercising them at scale in production environments
- Provide the ability to revert systems back to their original states without impacting customers, employees, or consumers
- Save time and money spent responding to systems outages

How do you know if a chaos engineering program is working? The top-level measure is overall system availability. For example, companies like Amazon or Netflix measure how available they are by whether their customers can use their product. They define availability in terms of "9s." Four nines availability mean that a system is available 99.99% of the time. Five nines availability is better, meaning the system is available 99.999% of the time. Six 9s are even better. Specific applications and sub-services are often measured using this metric as well.

Translate these numbers into actual outage time, and you see why it matters (see Figure 2). You can see why six nines is today considered the gold standard of reliability.

Another metric is the frequency and duration of outages. Yet another metric is measuring the operational burden of staff of system outages. How often did you have to page an IT support professional? How frequently did they have to answer a call at 2am to firefight an issue?

Chaos engineering is also good for disaster recovery (DR) efforts. If you regularly break your systems using tight experimental controls, then when your systems go down unexpectedly, you're in a much better position to recover quickly. You have your people trained, and you can respond more promptly. You can even put self-healing properties in place so you can continue to maintain service with minimal disruption.

If we break a system in a controlled and careful manner and we make sure we can recover from it, then, when outages happen unexpectedly, we're in a better position because our human workers are being trained to respond systematically to failure, as opposed to being called at 2am.

Systems can also self-heal, auto-recover so that they can operate in a degraded state and still maintain their service levels. The goal is resilience, rather than stability. Resilience meaning systems can gracefully handle inevitable failure without impacting users.

Using chaos engineering for disaster recovery is also important for compliance reasons. Sarbanes Oxley II as well as industry- or geography-specific regulatory mandates require that you can recover quickly from a disaster. But efforts to comply are often done at the theoretical level, as so-called “table-top” exercises, and are therefore incomplete.

No. of 9s	**Amount of downtime**
Four nines (99.99% availability)	52 minutes 36 seconds
Five nines (99.999% availability)	5 minutes 15 seconds
Six nines (99.9999% availability)	22 seconds

Figure 2: Translating 9s of availability into minutes of downtime

Best practices in chaos engineering

Many businesses are skeptical that deliberately trying to crash systems will make them stronger. And they are correct that there are risks. However, there are also best practices that mitigate those risks.

Minimize the "blast radius." Start with the smallest chaos experiment you can perform that will teach you something about your system. See what happens. Then increase the scope as you learn and as your confidence grows.

Don't be a chaos monkey. Chaos Monkey was Netflix's famous—or infamous—tool that randomly rebooted servers. Unfortunately, today many people believe that chaos engineering means randomly breaking things. The reason this is not an optimal approach is that "random" is difficult to measure. You are not approaching the problem using experimental methods. The idea behind chaos engineering is to perform thoughtful, planned, and scientific experiments instead of simply random failure testing.

Build (using open source)	Buy
Limited set of tools available	Growing availability of solutions
Everyone who does it is reinventing the wheel	Piggyback on earlier successes
Costly and time-consuming to train internal team	Avoid engineer and administrator burnout
Unsupported open source releases open up security vulnerabilities	Security embedded in solution
No “kill switch” or safety valve to stop out-of-control experiments from taking down production systems	Kill switch to avoid impacting users

Figure 3: The debate between building and buying chaos engineering tools

Start in a staging environment. Yes, you must eventually test in your production system, but it makes sense to start in a staging or development environment and work your way up. Start with a single host, container, or microservice in your test environment. Then try to crash several of them. Once you've hit 100% in your test environment, you reset to the smallest bit possible in production, and take it from there.

Avoid the "drift into failure." This concept, invented by flight accident investigation expert Sydney Dekker, refers to the fact that tension always exists in systems between efficiency and safety. Since businesses need to be mindful of costs, they tend to operate on the edge of safety. So once you understand a particular kind of failure and you've tested it, you want to automate the testing of it in a continuous deployment pipeline so you maintain that competence.

Always have a kill switch. This is akin to an "undo" button or safety valve. Make sure you have a way to stop all chaos engineering experiments immediately, on the spot, and return all systems to their normal state. If your chaos engineering causes a high-severity incident (SEV), then track it carefully and do a full post-mortem analysis of what went wrong.

Fix known problems first. Never conduct a chaos experiment in production if you already know that it will cause severe damage, possibly affecting customers—and your reputation. Always try to fix known problems first.

Conclusion: eschew complacency

Systems will break. And as systems continue to grow in complexity, they will break more often. If you're not prepared for that, they'll break in unexpected ways at unexpected times and bring your software or service down. You'll have unhappy customers and unhappy employees, and the costs are probably higher than you think.

Too many businesses are complacent. They think that just because they haven't had a major system outage before--or one that directly impacted customers--that they're safe. Or they think that the cost of deploying chaos engineering is more than the cost of simply fixing any problems that arise. They're wrong. Companies that don't address resiliency issues with chaos engineering may end up hiring tens or even hundreds of systems administrators just to maintain system uptime. That adds up.

In today's interconnected, internet-based world, no one is safe from system failure. The only way for it to not impact your customers, employees, partners, your reputation, and your bottom line, is to proactively address it upfront. Chaos engineering is the optimal way to do this.