SRE vs DevOps

DevOps. Site Reliability Engineering (SRE). Are they different or just different names for the same thing? This article explores that question in depth by delving into each and then comparing them.

DevOps is an important paradigm shift to bridge the gap between the typically siloed development teams and operations teams. Traditionally these two teams rarely communicate, much less collaborate on work. Development writes code and throws it over the metaphoric wall to operations whose job it is to deploy that code, with all its dependencies and configurations, and keep it running.

Site Reliability Engineering is the next stage implementation of DevOps. DevOps is a philosophy with a wide range of implementation styles available. SRE is more prescriptive about how things are to be done and what the priorities of the team explicitly are, specifically, the job is to keep the site reliable and available and only things that contribute to this goal are prioritized.

## What is DevOps?

The shortest definition of DevOps is combining development and operations teams for the purpose of moving code into production as quickly and smoothly as possible. The philosophy behind DevOps is that teams who share the responsibilities for both code writing as well as maintenance to keep it running well once in production are more efficient.

### What is the Role of DevOps in an Organization?

According to Google, the primary role of DevOps in an organization is to “increase software delivery velocity, improve service reliability, and build shared ownership among software stakeholders.” This is done via a cultural and organizational movement, one that requires focus and buy-in from stakeholders, because it really is a new way of thinking about software development.

### What are the Benefits of DevOps?

DevOps benefits an organization by improving the speed of software delivery with more frequent releases comprised of smaller changes. This is a competitive advantage, allowing companies to bring products to market faster, whether feature additions or stability/bug fixes. We split out large software into services or microservices, making updates and replacements easier and faster, and since we have trained teams overseeing them and implementing good practices like failover schemes and [Chaos Engineering](https://www.gremlin.com/community/tutorials/chaos-engineering-the-history-principles-and-practice/) to enhance reliability, we minimize the opportunity for failure due to networking and messaging problems.

DevOps also improves software stability, because even though changes are pushed to production frequently, those changes are small and therefore have far less potential to cause disruption. Further, small changes are easy to roll back quickly in the event of an unforeseen problem, making it safer to push those frequent changes.

Another benefit is in the availability and security of teams’ software delivery capability. When we are using a toolchain and build process frequently, we work out the problems and the process gets smoother and easier over time. Of course, we also automate it, which itself has great benefits. All this leads to reduced opportunity for errors, bugs, and security holes.

### What is Site Reliability Engineering?

Site Reliability Engineering (SRE) is the outcome of combining system operations responsibilities with software development and software engineering. SREs accept a broad range of responsibility relating to software code. If they write it, they build it, they ship it, and they own it in production.

One interesting metaphor in common use is that the class SRE implements the DevOps interface. In other words, classes in object-oriented programming often include more specific behaviors than what interfaces define and sometimes classes implement multiple interfaces. In that sense, SRE includes practices and recommendations that are sometimes more precise or additional to what DevOps describes.

We define Site Reliability Engineering in detail in [What is Site Reliability Engineering? A Primer for Engineering Leaders](https://www.gremlin.com/site-reliability-engineering/).

### What are the Benefits of Site Reliability Engineering?

Where DevOps brings greater collaboration and velocity to companies, the main benefit of Site Reliability Engineering is greatly enhanced uptime. The strong focus on keeping a software platform or service running is the foundation of SRE. The goal is to keep things operational “no matter what,” meaning that significant effort and emphasis is placed on things like redundancy, disaster mitigation and prevention, and ultimately, reliability.

For an SRE, uptime is key. Even beyond what is promised, the goal is always to find better and better ways to prevent problems that can cause downtime and to keep things up and running. The unexpected happens, and we all know it, so perfection is not the focus. Instead, the focus is on learning from past problems, preventing recurrence, and anticipating as many potential problems as possible. [Top-notch SREs](https://www.gremlin.com/site-reliability-engineering/how-to-become-a-top-notch-sre/) do all of these well and are [paid accordingly](https://www.gremlin.com/site-reliability-engineering/how-much-money-do-sres-make/).

It is not a coincidence that companies such as Evernote and Home Depot with solid SRE teams can demonstrate significantly improved uptime, as shown in these case studies from [Google](https://landing.google.com/sre/workbook/chapters/slo-engineering-case-studies/).

### What is the role of Site Reliability Engineering in an Organization?

The role of Site Reliability Engineering in an organization is to keep the organization focused on what ultimately matters to customers: the platforms and services customers want must be available when customers want to use them. [Team members](https://www.gremlin.com/site-reliability-engineering/the-role-and-responsibilities-of-sres-in-software-engineering/) use a variety of tools, programming languages, and a broad skill set, making the job one that is constantly stimulating and interesting. See our sample [SRE job description and interview questions](https://www.gremlin.com/site-reliability-engineering/sre-interview-questions-and-job-descriptions/) article for more.

## How does DevOps Work?

DevOps works by building a culture of collaboration from the beginning. Teams must work to establish trust between members, and by sharing responsibilities of all the stages of software development team members can make more informed decisions about the code that they write, test, deploy, and maintain. This flies in the face of past software development methodologies that relied on an assembly line of multi-stage testing deployments, review committees comprised of people across the business, and careful, often tedious, checklists.

It has always been a challenge in a waterfall setting to get code from idea to implementation to production efficiently. Even a major bug fix from a quality software engineer would require navigating organizational silos, setting up meetings and a sign off from multiple departments, many of whom might have only a passing interest in the system or service involved. It is not uncommon for a feature update to take six to nine months to make it into productions and provide value to customers. This is untenable in today’s marketplace.

Instead, DevOps teams are entrusted by the business to remember the big picture while writing code, because those same people must work together to deploy that code to production and maintain it. The very same team is responsible for bugs, outages, or anything else related to the code they have written.

Teams are empowered to experiment and innovate. They own the code. They own the process. They own the deployment. They also hold the power to make improvements and try out new ideas without approval from anyone outside the team.

The team is accountable for the reliability of their code and deployment and are otherwise given wide leeway to determine their own processes, change approvals, management, and needs. This requires a cultural shift and a great degree of trust, including trust among team members and also trust from management.

### Four Metrics for the Success of DevOps

Google’s [Jez Humble](https://continuousdelivery.com/about/) defined [four metrics for the success of DevOps](https://thenewstack.io/googles-formula-for-elite-devops-performance/):

**Lead time for changes** measures how much time you must plan in advance for a proposed software change to make it to production. Decreasing that is vital for increased deployment cadence. Low performers take a week or even a month. High performers only need a day or less.

**Deployment frequency** has a direct impact on how rapidly it is possible for software users to benefit from bug fixes and new or enhanced features. Ultimately, elite companies deploy multiple times per day!

**Time to restore service** is the amount of time required to bring services back up when a problem occurs. Getting your number down under one hour is ideal. Eliminating the need entirely is an unreasonable expectation in an era of increased deployment velocity that sometimes introduces breaking changes. Note that this and the next entry do not mean failure of the overall system, but only failure of an individual service. If you are using canary deployments, the failure of a new service instance should have no impact on the numerous instances of the previous stable release and therefore there should not be a customer impact, even though you encounter problems.

**Change failure rate** measures how frequently a deployed release has to be rolled back due to it not working properly. The best teams have a rate between zero and 15%. Things like code reviews, testing, and good design help, but our systems are so complex and under constant change that we should expect some service failures.

### How Do We Make This Happen?

How do we accomplish any of this? First, we need good measurement. Observability. We monitor our systems and use what we learn to inform our business decisions. Bottlenecks and squeaky wheels get attention, sooner rather than later.

Failure notifications are sent proactively based on data thresholds set in monitoring tools. We actively work to automate failure mitigation and try to set activation based on monitoring data thresholds set well below actual failure levels, so that even if a node fails or networking falters, end user needs are already routed to other paths and our customers never notice there was a problem.

DevOps requires establishing a cultural norm that accidents are normal and that failures happen--and that neither should be a lightning rod for blame. Eliminating blame enhances a team’s ability to focus on how to fix problems and experimentation rather than worrying about reputations and battling anxieties. Increasing the rate of change will also increase potential failures, so DevOps cultures need to be comfortable with failure while also focusing on recovery and backups.

### DevOps Technology and Tools

Some of the technical solutions that effective teams use in their DevOps workflows include:

* Version control for all code, including configuration management and secrets management, using tools such as [git](https://git-scm.com/), with [GitHub](https://github.com/) or [GitLab](https://about.gitlab.com/) for centralized master repository access or similar
* A trunk-based development model where developers and engineers pull from a master branch frequently and push changes that are as atomic and small as possible as frequently as possible in separate pull or merge requests
* Continuous integration using tools like [Jenkins](https://jenkins.io/), [Spinnaker](https://www.spinnaker.io/), [Travis CI](https://travis-ci.org/) or similar
* Deployment automation, typically using the same CI tools
* Test automation, including security tests starting as early in the pipeline as possible, using tools like [Selenium](https://selenium.dev/), [Postman](https://www.getpostman.com/automated-testing), [mabl](https://www.mabl.com/" \t "_blank) or similar
* Incident management, for when things go wrong, using tools such as [Opsgenie](https://www.atlassian.com/software/opsgenie" \t "_blank), [PagerDuty](https://www.pagerduty.com/platform/), and [VictorOps](https://victorops.com/" \t "_blank) or similar

## How do DevOps and Site Reliability Engineering Compare: SRE vs DevOps

Systems fail, sometimes publicly and at great cost. Airlines have been hit with [system-wide ticketing outages](https://www.cnbc.com/2019/04/29/major-us-airlines-hit-with-systemwide-outages.html) causing significant inconvenience and the company responsible said, “No downtime is acceptable” as they apologized for the downtime. Costco’s website [crashed for several hours on Thanksgiving Day](https://www.businessinsider.com/costco-website-outage-costs-millions-in-sales-2019-11), costing them an estimated $11 million. CenturyLink had an outage lasting over 24 hours that included [disruption to the vital 911 emergency service](https://www.theverge.com/2018/12/28/18159110/centurylink-internet-911-outage-fcc-investigating). These are just highlights from 2019.

Can we prevent outages in an era of such great velocity? We have gone from annual software releases to daily releases, from running software as a monolith to running hundreds of microservices, from on prem hosting on hundreds of physical hosts to Kubernetes, containers, and cloud hosts numbering sometimes into the hundreds of thousands.

This is where it is vital to join DevOps with Site Reliability Engineering perspectives and implementation.

Site reliability engineering may be thought of as a specific implementation of DevOps, even though they were developed separately. There are many similarities in intent and foundational perspectives. Differences mainly result from a narrowing of team focus.

Both DevOps Engineers and Site Reliability Engineers begin with a belief that change is necessary to improve. No software remains stagnant. No system idles unchanged forever. Whether it is fixing bugs or evolving and adding features, things change. Capacity needs wax and wane and infrastructure cannot remain static. Everything must and will eventually change or die out.

Both have a strong focus on working together as a team with shared responsibilities and an assumption of collaboration. No one works in a silo. Ownership is shared from initial code creation to software builds to deployment to production and maintenance. Keeping everything working is everyone’s responsibility, even if there is some role-based focus for individual team members, the responsibility remains everyone’s.

While both consider atomic changes a shared value, with reliability as the main focus, managing change is vital for SRE. Both promote making software changes as small as possible, because small deltas usually merge more smoothly and are easier to roll back when a problem arises. However, the R is SRE is “reliability” and that focus promotes this value to a higher standing.

How these small changes are merged and then integrated into a build and deployed may differ from a tooling perspective across DevOps and SRE, but both share a strong preference for automation where possible. SRE tends to take this to the logical extreme where it can, seeking to automate the CI/CD pipeline, testing, chaos experiments, and more.

SRE teams work to automate nearly every action that is performed more than once or twice by a human, removing any possible toil from the daily routine in favor of using human intellectual capacity to find and enact improvements. This may happen in a DevOps team, but it is rarely a focus.

The tools used by each type of team are generally similar and may be nearly identical, with the exception of team-written tools specific to that team’s responsibilities. The main similarity is a perspective that is focused on APIs and abstracted interactions rather than direct entanglements between systems or for administration and management tasks. Some tools are created in-house, some are adapted open source tools, and some are purchased proprietary tools.

A huge similarity is the requirement for good measurement and observability. Data, especially good data, is vital to both DevOps and SRE. One big difference is that SRE teams always focus on service level objectives (SLOs), keeping them and improving systems to maximize effectiveness based on them. DevOps tend to think about what the data tells them about the system, how it is running, where it is weak or failing, and so on. SREs tend to be more specifically practical, thinking about how to use the same data to improve performance on one or more SLOs, even using machine learning techniques to have systems adapt themselves to changing circumstances.

Both DevOps and SRE teams share the expectation that bad things happen. System components fail. Humans accidentally input the wrong instructions. Networks get overloaded and latent or fail. With this expectation, focus is put on how to prevent and then how to fix quickly when prevention fails. There is no blame placed on anyone. Looking at failures after they are repaired in a blameless way with a blameless retrospective or postmortem permits teams to focus on how to prevent a recurrence of the same problem rather than keeping silent out of fear of repercussion. Better systems result.

The biggest difference between DevOps and SRE is not in perspective or wider philosophy. The cultures are also very similar. The biggest difference is that SRE has an intentionally narrowed focus on keeping services and platforms available to customers while DevOps tends to focus on overall processes, which is much broader.

The two have different foundational guiding principles at the lowest layer as DevOps simply believes it has found a better way to meet the needs of the company and its customers while SRE believes it exists to keep a site reliable. It is interesting that both perspectives, developed separately, have some to embrace such astoundingly similar practices.