Site Reliability Engineering

Philosophies, habits, and tools for SRE success

New Relic®

Table of Contents

Introduction

The day-to-day responsibilities of developers and operations engineers are increasingly evolving as high-growth companies look for new ways of improving stability, reliability, and automation-first practices. Because of the need to reduce downtime (with less manual intervention) as systems scale, many organizations are adopting the site reliability engineer (SRE) role.

The phrase “site reliability engineering” is credited to Benjamin Treynor Sloss, vice president of engineering at Google. Sloss joined Google in 2003 and was tasked with building a team to help ensure the health of Google’s production systems at scale. According to Sloss, site reliability engineering is “what happens when you ask a software engineer to design an operations function.” Site reliability engineering is a cross-functional role, assuming responsibilities traditionally siloed off to development, operations, and other IT groups.

Sloss's team wrote the original book on site reliability engineering, so if you're wondering what a great modern SRE practice should look like in a DevOps world, the **Google Site Reliability Engineering book** is a fantastic point of reference.

In it, Sloss writes, "It is a truth universally acknowledged that systems do not run themselves. How, then, should a system—particularly a complex computing system that operates at a large scale—be run?"

Google's answer has been to hire software engineers to do the work usually handled in traditional organizations by IT operations teams. "Our site reliability engineering teams focus on hiring software engineers to run our products and to create systems to accomplish the work that would otherwise be performed, often manually, by sysadmins," explains Sloss.

From Google to the rest of the world

After the book was first published, the role was rapidly adopted in a wide range of companies, prompting technology news and analysis site TechCrunch to wonder, back in 2016, "**Are site reliability engineers the next data scientists?**" The following year, **LinkedIn named SRE "one of the most promising jobs" in tech**. Speaking in 2018, Beth Long, a software engineer at **Jeli**, told us, "My impression is that there's a slow trickle-down to smaller companies. Google, and Netflix, and Amazon, and Heroku—these companies have had SREs for a long time, because they have the resources and the scale that demand it. You're starting to see that role appear in smaller companies where they realize 'Oh, we need someone to play this role.'"

The demand for SREs remains higher than ever.

Three years on, this remains true. As more organizations are building distributed microservice-style systems that run at scale, the demand for SREs remains higher than ever.

Starting your SRE journey

It is important to note that the SRE role will vary considerably from one organization to another. While job descriptions and day-to-day tasks for SREs vary, the role's utility is quickly becoming apparent to those software organizations that have adopted it. So, where does that leave you?

Whether you're still figuring out how to create a site reliability practice at your company or trying to improve the processes and habits of an existing SRE team, the more you know about the subject, the better—especially since what works for a massive company such as Google might not work for a small or midsize outfit. To that end, this ebook shares the philosophies, habits, and tools of successful SREs, along with New Relic's definition, guidelines, and expectations for the role.

CHAPTER 1:

SRE Philosophy and Principles

Google defines an SRE as an operationally minded software engineer, but what does that mean? At Google, SRE teams are responsible for both capacity planning and provisioning. The teams are different from purely operational teams in that they seek software engineering solutions to problems. To enforce this, Google caps the amount of time SREs spend on purely operational work at 50%. This means that, at a minimum, 50% of a Google SRE's time should be allocated to engineering tasks, such as automation and improvements to the service.

The goals, risks, and trade-offs of Site Reliability Engineering

When first thinking about an SRE team's role, you might assume that increasing reliability, generally measured by monitoring system uptime, would be the primary goal, but beyond a certain point that turns out not to be the case. This is because factors outside of the SRE teams' control come into play, such as network reliability. There is also a trade-off between reliability and development team velocity.

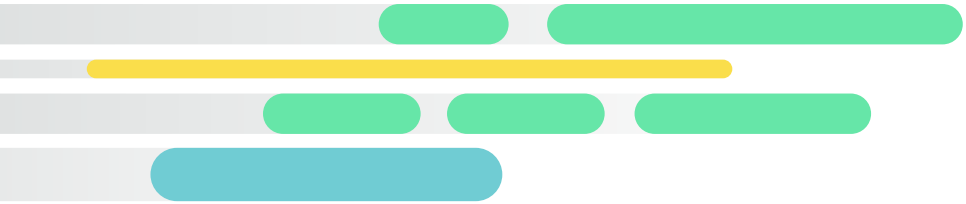
Because of this, site reliability engineering will generally seek to balance the risk of unavailability with the goal of rapid innovation and efficient service operations. In the **Google SRE book**, Marc Alvidrez writes, "We strive to make a service reliable enough, but no more reliable than it needs to be. That is, when we set an availability target of 99.99%, we want to exceed it, but not by much: that would waste opportunities to add features to the system, clean up technical debt, or reduce its operational costs."

One way to think about and manage this trade-off is to consider the point in its life cycle at which a given product or service is. For a relatively young product, setting a stringent goal for the uptime of a service will likely be counterproductive, because it will reduce the pace of innovation and experimentation in an undesirable way. Conversely, as a product reaches maturity and has a base of customers that depends on it, downtime becomes more problematic and can potentially have a direct impact on the service provider's bottom line. At this point, increasing the target for uptime makes sense.

Golden Signals

While measuring service availability is a good starting point, particularly for user-facing services, SRE teams will typically have several other business-oriented key metrics that they also track. These metrics, which often include the four **Golden Signals**, are best thought of as defining what it means for a given system to be "healthy."

Different types of applications will have distinct metrics. For example, user-facing services might care about availability, latency, and throughput, while big data systems tend to focus on throughput and end-to-end latency. It is worth noting that the measurement isn't an end in and of itself. What is important is how it indicates the quality of user experience and system effectiveness.



Using SLOs and SLIs to measure reliability

Service level objectives (SLOs) are a common way to measure a service provider's performance and can be equally important to site reliability engineering success. Clearly defined and measured SLO metrics at the product and service level help organizations to:

- Tune investment and overall prioritization to meet reliability goals and meaningfully adjust those high-level reliability goals to fit company strategy
- Maintain and build customers' confidence
- Enable teams to decide when and how to focus efforts on reliability
- Allow engineers to make better assumptions about risk tolerance and how fast they can go, and reason better about dependencies and reduce unnecessary toil

As an example, Stephen Weber, a Senior SRE at New Relic, told us that the **New Relic core data platform** has three key metrics: The first is correctness (Were the correct results returned?), and the second is latency (Did it respond in an acceptable amount of time?). "And then the third is, to ensure that they are getting good latency, a safety valve technique to stop processing and provide partial results (also known as graceful degradation). And so they have a third SLI of keeping that to a minimum." The three metrics together form this SLO for the core data platform.

If teams consistently exceed their SLOs (for example, 99.9% availability for all services), they may be able to move faster, take on more risk, and deliver more features. If a team

The hallmark of a good SLI/SLO is the metric's relevance to the business outcomes, often the user experience.

is in danger or isn't meeting its SLOs, it's a signal to back off and pause to focus on reliability so that the team can start moving faster again.

SRE teams may also have other service level indicators (SLIs) that they use to measure reliability that are not necessarily part of their SLO. These performance metrics track some facet of the business; for example, an SLI for a database service could be something like, "The fraction of user queries that are successfully completed within 200 milliseconds without error."

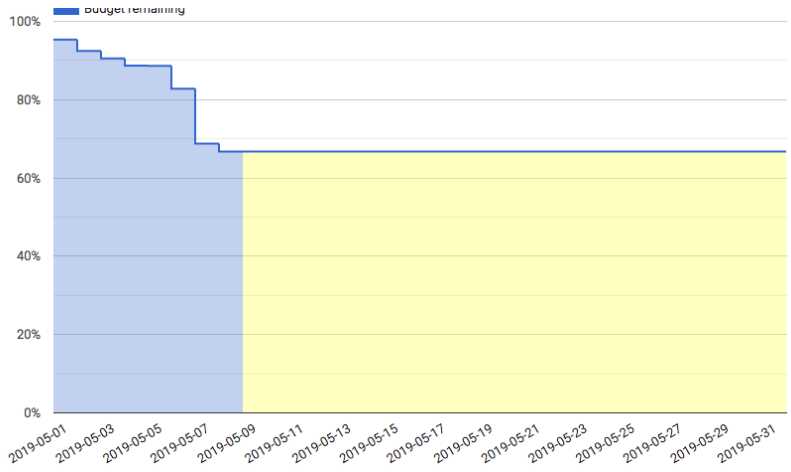
To measure reliability, teams turn to metrics like mean time between failures (MTBF), mean time to detect (MTTD), and mean time to resolution (MTTR), all of which help organizations define their "risk matrices." These become powerful tools for prioritizing issues and risks that will have a quantifiable impact on SLOs, and they also allow organizations to downshift on issues that may not be especially urgent.

The hallmark of a good SLI/SLO is the metric's relevance to the business outcomes, often the user experience. For example, a high error rate or slow response time has a negative impact on the user experience. High CPU utilization might have a negative impact on the user experience, but the relationship between high CPU and a bad user experience is harder to establish.

Error budgets

Finally, although perhaps not essential, it can be helpful to define a quarterly error budget based on a service's SLO. The error budget provides a clear, objective metric that determines how unreliable the service is allowed to be within a single quarter.

Teams can have burn-down charts that show how quickly they are going through their error budget, and adjust work accordingly. Interestingly, at Google, if a service is providing 100% uptime, they will take the service down so dependent services are forced to know how to react.



Of course, if the error budget is too tight, it can slow the pace of development. Having an error budget in place allows you to reason about this and make a decision to perhaps relax it in order to be able to increase development team velocity. In that situation, the product and SLA engineers might decide to increase the allowable error count to enable faster development. Some organizations sort their apps into "high reliability" and "high velocity" and set stricter/looser error budgets accordingly.

Whatever the setting, an error budget is important because it aligns incentives and emphasizes joint ownership between software engineering and product development.

CHAPTER 2:

What Makes an SRE Successful?

When choosing an SRE, a candidate's technical contributions will depend on how a particular organization defines or approaches the role: One company might require more software engineering and coding experience, whereas another organization might place a higher value on operations or QA skills. Whatever the balance, what sets the "great" apart from the "good enough" is often a combination of habits and traits that complement technical expertise.

Here's how you'll know you've found a fantastic SRE.

SREs see the (much) bigger picture

Successful software developers understand how their code helps drive the overall business, and great SREs have their own version of this trait. "You're looking for someone who is thinking about the bigger picture outside of the day-to-day," said Jason Qualman, a Senior Software Engineer at New Relic. "A successful SRE is someone who can understand and interpret things at a higher level." Changes can create risks or impacts down the road, not just in that current moment, and a good SRE is sure to perform a thorough analysis before making any changes.

The ability to consider how their work will affect the rest of a particular system, team, or the larger infrastructure is the kind of extreme pragmatism that SREs need. There's little long-term upside in a siloed approach that throws a change over the wall with no concern for how it might affect the person sitting on the other side.

"We are making decisions very low in the stack," Qualman said of the SRE. Those decisions will affect people much further up the stack. Good decisions enable seamless transitions.

see bigger picture

SREs are curious and empathetic

Kat Dober and Stephen Weber, both Senior SREs at New Relic, cite curiosity as a key trait they look for in an SRE.

"You're looking for people who have that engineering mind-set," according to Dober. "You want to know how it works. You want to know the ways that it might fail. You want to be thinking about those from the beginning."

Weber agrees: "Oftentimes, the improvements that have been most beneficial started with 'Oh, that's funny.' And then you keep digging into that."

A related trait is customer empathy, according to Weber. "Maybe your page-load average is pretty good, but if some subset of customers are experiencing really long load times, you need to see that bad experience," said Weber.

curious
& empathetic

SREs automate at every opportunity

While there will always be some manual exploration involved in the role, SREs look to reduce work "toil." Toil has a specific meaning at Google, given by Vivek Rau in the SRE book as "the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows."

In a **follow-up** article to his chapter published via Google Research, Rau et al provide a case study for reducing toil for the SRE team supporting Google's Bigtable service. "Bigtable SRE was able to create a snowball of work reduction: each incremental reduction of toil created more engineering time to work on future toil reduction.... by 2014, the team was in a much-improved place operationally—they reduced user requests from a peak of more than 2200 requests per quarter in early 2013 to fewer than 400 requests per quarter."

The key to achieving this was to gradually increase the amount of automation for the various common types of support requests. The more general lesson is that SREs will typically focus on automation as a key technique for reducing painful manual tasks and toil.

"Automation really comes in once you understand your problem space or once you understand your infrastructure, and there are things that you know are going to have to be done continually," Dober said. "For example, think about how you're going to configure all your hosts, or how you're going to get a piece of code from the repo that it's in, packaged up into an artifact or container, and deployed across your infrastructure. Automating those tasks reduces toil but it also makes sure the tasks get done consistently and correctly every time."

automate every opportunity

Qualman agrees. “A lot of this role is thinking about inefficient and time-consuming things people are doing and putting a stop to them as soon as possible,” he said. “Instead of kicking a can down the road on manual work, you’re saying, ‘I’m going to take the time to automate this right now and stop anyone else from having to do this painful thing.’”

This obsessive focus on automation is a key tenet of SRE—and DevOps—philosophy; in fact, ”The DevOps Handbook” has a chapter that discusses the counterintuitive effects of manual acceptance processes. And “automation” and its variants seem to appear more often than any other word in SRE job descriptions. It’s not that unexpected to see “Automate, automate, automate, and then…automate!” as a key responsibility in an SRE job listing.

This obsessive focus on automation is a key tenet of SRE—and DevOps—philosophy.

SREs are change agents

The confidence to advocate for SRE initiatives is another skill that distinguishes the best SREs. Part of the job, simply put, involves convincing other people to do things they initially might not want to do; for example, convincing a software engineer focused on quickly shipping a product feature to think about ways to scale that feature over the next several years.

This is something that can be more easily accomplished if the SREs are directly embedded in the product teams. Speaking at **QCon Plus**, **Johnny Boursiquot**, a Site Reliability Engineer at Salesforce's Heroku, talked about SRE adoption in a presentation called "**The SRE as a Diplomat**," during which he recommended the practice of embedding SREs in existing product teams as a way of driving change. "No two organizations implement the practices of site reliability engineering the same way," Boursiquot observed, "a fact that is seldom recognized when rolling out an SRE function for the first time."

Expanding on this theme he said:

"While there exists a set of best practices for its adoption, those that take on the task of championing SRE within their organization know that those prescriptive approaches do not provide all the pieces necessary for that adoption to be a smooth and immediately impactful one.

"Nowhere is this challenge of adoption more prevalent than in organizations where teams have complete ownership of a service from its development to its ongoing operational

change agents

needs. In these organizations, it is common, even necessary, for team-specific practices to develop. This total ownership model works well to move business objectives forward in the early part of a system's life cycle but eventually and insidiously morphs to become unaddressed technical debt when maturing teams need to adopt shared reliability practices and tooling.

Great SREs have to be effective salespeople.

"Bridging this gap between the intent of leadership and the practical implication within teams requires change agents, in the form of SREs, to be embedded within these teams. Teams that see themselves as self-sufficient are not always incentivized to work with a traditional and external SRE function requiring changes on how they operate, even if those changes will markedly improve things. Regardless of the reasons, building bridges across these teams requires that we first establish trust. Of course, one way to facilitate this trust building is to embed SRE directly within those teams."

In other words, great SREs have to be effective salespeople; they have to be able to sell their colleagues on processes and projects that might appear to involve some near-term pain or go against legacy norms. "You need to be able to dig in and say, 'stop' and 'no,' which can be difficult to do in some engineering organizations," according to Beth Long.

Hiring managers are best served by not pigeonholing the SRE role to one particular background.

SREs embrace new tools and approaches (when necessary)

Because site reliability engineering is still fairly new, many engineers who currently hold the title worked in other jobs before assuming the role. Some SREs might have a developer background, while others may come from traditional operations or sysadmin backgrounds, so hiring managers are best served by not pigeonholing the SRE role to one particular background. A traditional QA engineer might have a good skill-set for the SRE position, for example.

No matter your background, the SRE role requires you to be pragmatic and willing to adapt. It challenges you to move out of your comfort zone and develop new skills. "I interact with many different systems, different programming languages, different styles of YAML that I never really thought I would ever do, versus when I was a developer," said Weber. "Writing five different programming languages in a day is not necessarily unusual, so you just need to be willing to be flexible and jump in."

new tools & approaches

CHAPTER 3:

SRE Tools and Processes

For an SRE, part of being pragmatic means being willing to dump processes, procedures, and tools that may have been well-intentioned but are no longer productive.

Just as there's no universal job description for SREs, there's no standard toolset for the role either. However, great SREs always seek to optimize reliability tools and processes and evangelize them throughout the organization.

It makes absolute sense—optimization is key to a successful SRE practice and for proper implementation of DevOps principles. But what tools should SREs standardize on? Each team needs to decide what's best for them. The good news is, there are plenty of choices.

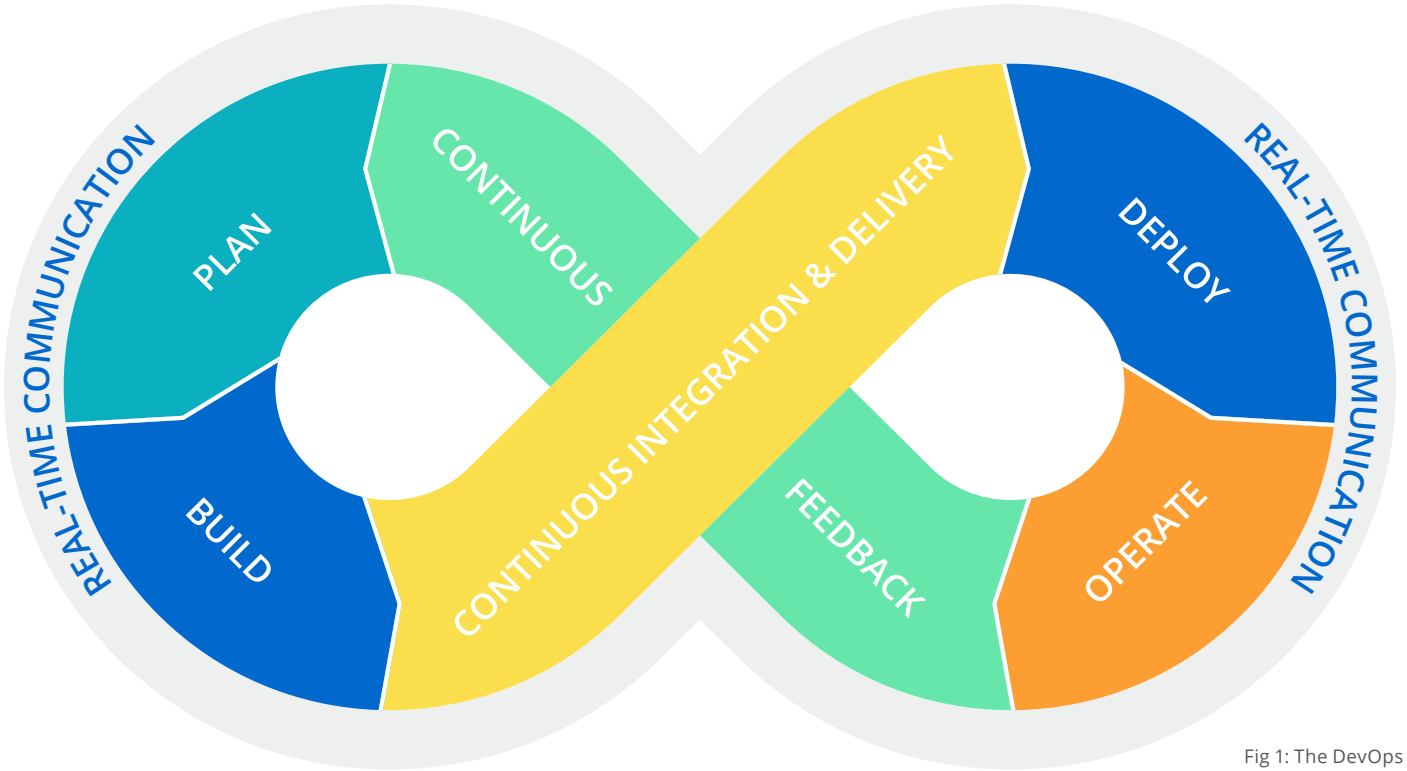


Fig 1: The DevOps toolchain

Stages of the DevOps (and SRE) toolchain

If you created a “stages of the SRE” toolchain, it probably wouldn’t surprise you if it looked a lot like the DevOps toolchain (Fig.1).

The increasing use of the public cloud and the corresponding rise in the use of Infrastructure as code tooling means that this is an area that sees particularly rapid change and churn. We can, however, outline some of the current widely used tools and practices.

PLAN:

This comprises both agile project management and tracking tools such as **Avaza**, **Jira**, **YouTrack**, **Trello**, **Pivotal Tracker**, or other task management tools.

BUILD:

Here you'll find infrastructure as code tools, such as **Ansible**, **Chef**, **Docker**, **Puppet**, and **Terraform**, which make re-provisioning environments faster, more consistent, and more reliable. Containers and orchestrators, such as **Kubernetes** and **Docker**, also play a role, allowing developers and SREs to work against disposable, virtual replicas of production.

Source control and collaborative coding tools such as **Bitbucket**, **GitHub**, and **GitLab** as well as IDEs, such as **IntelliJ IDEA** and **Visual Studio Code**, are also widely used.

CONTINUOUS INTEGRATION AND DELIVERY:

It is increasingly common for developers to check code into a shared repository several times a day, running it through a suite of automated tests, and then automatically releasing the updated code to production if the test suite passes. The approach combines CI/CD tools such as **AWS CodePipelines**, **Bitbucket pipelines**, **CircleCI**, and **Jenkins** with testing tools such as **JUnit**, **Mabl**, **Sauce Labs**, and **Selenium**. A critical point regarding continuous delivery is that while teams have software that is ready to deploy, they don't necessarily deploy it immediately. (See Deployment below.)

Many New Relic customers also build pipeline dashboards to help track this stage of the process:

OPERATE:

This typically involves monitoring tools, such as **New Relic**, alongside incident, change, and problem tracking tools, such as **Jira Service Desk** and **Statuspage**, **PagerDuty**, or **Zendesk**. At New Relic, our SREs and engineers use the **log management** capabilities and custom instrumentation.

AIOps tools, such as New Relic's **Applied Intelligence**, can proactively monitor your services for anomalies and notify you with real-time failure warnings and actionable details so you can investigate faster. Incidents can be delivered directly into tools such as PagerDuty.

DEPLOY:

Deployment is a separate step if you are doing continuous integration and delivery but not yet continuous deployment. You use the same tools as the continuous integration step above, but the key difference is whether the default is to deploy the code as soon as it is ready. There are business reasons for not doing continuous deployment, but creating frequent, small, incremental updates that ship automatically is an SRE best practice.

CONTINUOUS FEEDBACK:

Covering both the culture and processes for collecting regular customer feedback, aided by tools such as **GetFeedback**, **Slack**, and **Pendo**. The feedback part of the loop also includes metrics on performance and processes, so for example, Jira tickets for DRI (Don't Repeat Incidents) work. A release dashboard is also an example of continuous feedback.

Nothing is written in stone

The tools SREs use at any given time will depend on where an organization is on its SRE journey, and the shift we've seen to the public cloud has also changed the role considerably. New trends, including the ability to automate through AIOps tooling, will continue to redefine the role.

While less mature organizations will tend to use more specialized operations tools, more mature organizations will see more convergence between SRE and software engineering toolchains. So, while it's certain that there's no one-size-fits-all set of tools, SREs should experiment with and adopt the right tools as they seek new, more efficient ways to bring greater reliability to everything they do.

The ability to automate through AIOps tooling will continue to redefine the role.

CHAPTER 4:

The Evolving SRE Role at New Relic

Google's **Site Reliability Engineering** book does a great job of outlining what a great modern SRE practice can look like in a DevOps world. But what about SRE practices at companies that aren't as large as Google? For all that's been written about reliability practices, it's surprisingly hard to find specific, detailed descriptions of the day-to-day role that SREs play in other engineering organizations. Most descriptions on the internet contain rather vague phrases like, "SREs combine software engineering and operational skill sets" and "SREs automate all the things."

Defining the role

Creating the New Relic SRE description took time and involved input from individual SREs and executive leadership.

SREs at New Relic are engineers who focus on, and are recognized primarily for, improving the reliability of our systems. From a business perspective, the goal of the work that SREs do is to build and maintain customers' trust, and allow the business to scale by steadily decreasing the per-service and per-host operational overhead of New Relic's platform.

At a high level, SREs make this happen by:

- Championing reliability best practices
- Guiding designs and processes with an eye toward resilience and low toil
- Reducing technical complexity and sprawl
- Driving the usage of tooling and common components
- Implementing software and tooling to improve resilience and automate operations

Evolving the role

When New Relic first created its SRE function, it was based around a centralized team, very much as described by Google, but New Relic now has SREs permanently embedded into the various product teams. This latter approach is similar to the one that **Boursiquot** described earlier.

Gus Shaffer, a Senior Director of Engineering in the Telemetry Data Platform group, which has a high concentration of embedded SREs, told us that having a centralized function for reliability worked against the DevOps goal of having one team responsible for coding and release. “We found that there was an abdication of responsibility for reliability, where people are like, ‘Oh, well, there’s a reliability organization, they’re responsible for reliability,’” Shaffer explained. “When, in fact, the reliability organization was actually responsible for measuring and reporting and helping people figure out what the trends are in their reliability, and put-

“We’re building the reliability practices into the tools that people are using.”

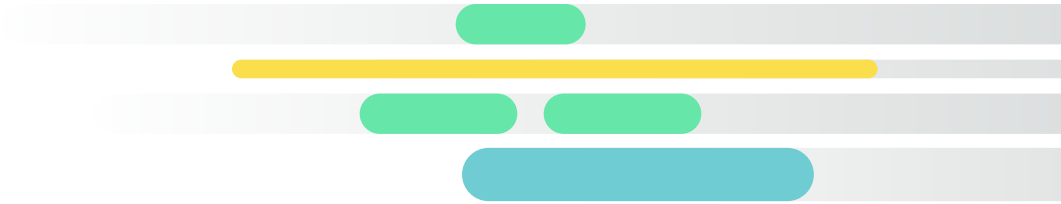
Stephen Weber, Senior SRE, New Relic

ting together processes and policies to help people do the right thing."

Weber echoed this view: "I think the biggest advantage of going from that central team to embedding on the platform teams is that we're taking on the idea of building the reliability practices into the tools that people are using."

The new structure makes it easier for the New Relic SREs to stay current with the overall product architecture. The structure change also reduces the amount of auditing work and performing the role of "bad-cop" that SREs are often required to do. Moreover, it made it easier for SREs to spend more time on development—a different way of achieving the same goal that Google's 50% cap aims for. In other words, changing the structure eliminated several problems, effectively executing an **Inverse Conway Maneuver**.

The change does come with its own set of challenges, however. One is that the lack of a centralized SRE function makes it harder to deal with cross-cutting concerns. For New Relic, an example is **Apache Kafka**, which is used for all New Relic's data pipelines. Extensive use means that it is vitally important that the platform's various clients use it as efficiently as possible. To help ensure this, New Relic is looking at introducing quotas and has spun up a production engineering team with a rotating roster of engineering staff. "We've brought in people from all these different teams so that we have subject matter expertise in all the different systems within the data platform," Shaffer explained. "It means that we have instant buy-in on making these changes, because people that are on the teams that are being impacted are part of this 'centralized' SRE team."



The SRE role at New Relic has also evolved in response to other factors. New Relic is increasingly moving toward using public cloud infrastructure rather than its own data centers. That shift has resulted in a corresponding change in how New Relic's teams work with software-defined infrastructure.

The change to the public cloud also means that using cloud resources efficiently has become an increasingly important part of the role. "The SREs are not necessarily the ones who are watching the AWS bill," Shaffer told us, "but they are responding to signals from leadership, like 'This system seems really expensive, more so than it probably should be, can you look into that?' It is also a part of the capacity management function that you don't over-provision."

What SREs do at New Relic

To summarize, the following table provides a high-level overview of the current SRE role at New Relic.

TYPE OF WORK	EXAMPLES	NOTES
Learn and enhance New Relic operational and reliability best practices, (e.g., capacity planning, SLOs, incident response), and work with teams to adopt those practices.	• Update your team's risk matrices. • Manage capacity in advance of customer demand. • Think about costs and the way we use cloud resources effectively. • Influence the team to prioritize the most important reliability work.	• This is a particular focus for new SREs and SREs working with new teams. • All SREs stay current on platform tooling and SRE community best practices.
Build or help teams adopt core shared internal components.	• Work with teams to migrate systems into a new version of our shared deployment pipeline. • Contribute code or tools to our container runtime platform. • Limit technical sprawl by guiding teams to select appropriate existing tools rather than building new ones.	• SREs are expected to use existing tools rather than introducing new tools or systems.
Improve the monitoring and observability of the New Relic platform.	• Work with teams to clean up noisy unused alerts and ensure that important problems are alerted on. • Build integrations to create new visibility into our platform.	• SREs actively use and extend existing New Relic products whenever it's possible and effective to do so and to influence product management to implement necessary features when it's not.

Set up your SREs for success

Although this SRE role description and approach works well at New Relic, it may not be right for other organizations. Regardless, it provides a useful example and helps clarify the tremendous value a great SRE practice can bring. By developing your own guidelines, you can set up SREs for success and advance the collective understanding of the vital role the SRE practice will play as it matures to support the ever-increasing complexity of computing platforms.

Finally, it's critical to create a community of practice and mentor/mentee relationships for SREs and others who care deeply about reliability and sharing best practices—that's what creates a culture of reliability.

Execution

Once you define the SRE role and have the right organizational structure and incentives in place, it all comes down to execution. A successful SRE team depends on a variety of skills and traits. You can always teach technical skills, but you can't necessarily impart equally essential qualities such as empathy and curiosity.

Some engineering cultures, such as New Relic's, prize autonomy—but that doesn't mean teams should have to tackle reliability independently. Teams (and individual SREs) need organizational support, communication, and, above all, trust to thrive.

A guiding philosophy for successful SREs might be expressed this way: Don't chase a holy grail—you can't prevent things from ever breaking. Instead, work tirelessly to see the big picture, incorporate automation, encourage healthy patterns, learn new skills and tools, and improve reliability in everything that you do. Perfection may be unattainable, but continually striving to do things better is the way to get as close as possible.