New Relic

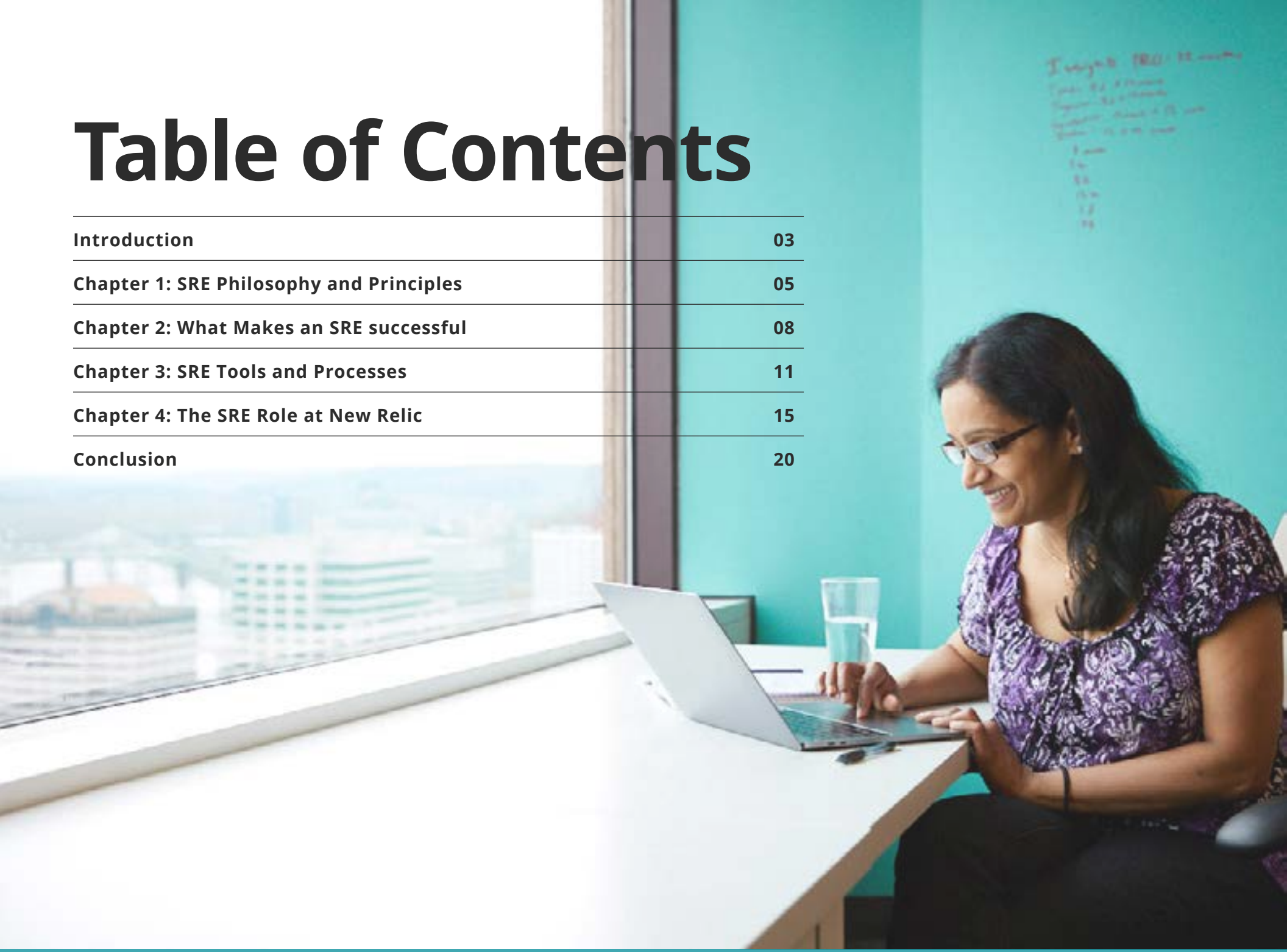
eBook

Site Reliability Engineering

Philosophies, Habits, and Tools for SRE Success

 | US +888-643-8776 | www.newrelic.com | www.twitter.com/newrelic | blog.newrelic.com

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 this need to reduce downtime (with less manual intervention) as systems scale, a new role is taking shape in many organizations: the site reliability engineer (SRE).

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—no small task. 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.

The proliferation of the SRE

No matter how you define it, the SRE role is clearly expanding into more and more companies. A recent jobs search for "Site Reliability Engineer" on Glassdoor produced more than 61,600 open positions at the time of this writing. Tech firms are certainly well represented—companies from Adobe to GitHub to Spotify (and plenty more) are all hiring SREs. But you'll also see plenty of other bellwether companies (such as GE, Chase, Walmart, and McGraw-Hill Education) and industries (including entertainment and education) seeking SRE practitioners, too.

And it's no surprise that companies of all shapes and sizes are starting to adopt the role. "My impression is that there's a slow trickle-down to smaller companies," says Beth Long, a software engineer with New Relic's Reliability Engineering team. "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.'"

As a result, folks with the right mix of talent and experience for the SRE role are increasingly in demand. Not too long ago TechCrunch asked, "Are site reliability engineers the next data scientists?" And last year LinkedIn named SRE as one of the most promising jobs in tech.

From Google to the rest of the world

Sloss' team literally wrote the 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 folks. "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.

Starting the SRE journey

While job descriptions and day-to-day tasks for SREs vary from company to company, the utility of the role is quickly becoming apparent to those software organizations who've 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 you're trying to improve the processes and habits of an existing SRE team, the more you know about the subject the better—especially since what may work for a massive company like Google may not always work for a small or mid-sized outfit. To that end, this ebook shares the philosophies, habits, and tools of successful SREs, along with New Relic's own definition, guidelines, and expectations for the role.

CHAPTER 1

SRE Philosophy and Principles

SRE Philosophy and Principles

Is SRE the purest form of DevOps?

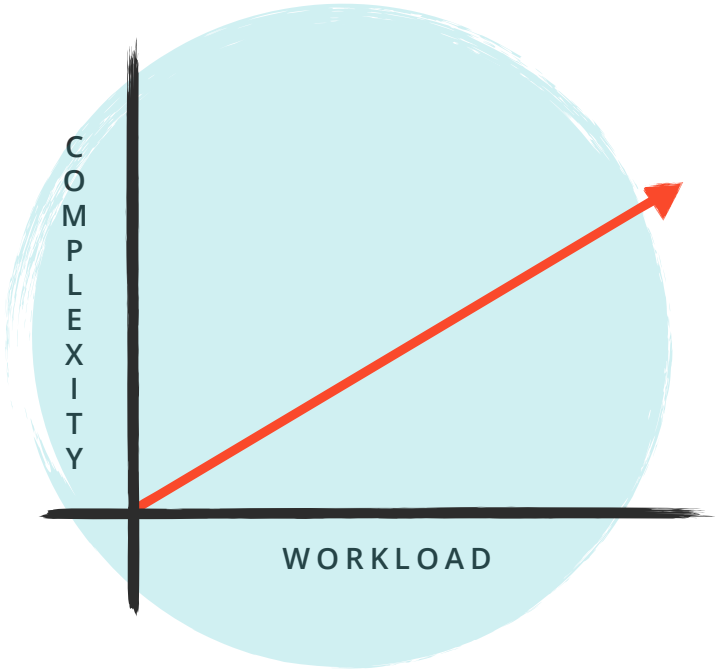
As vice president of site reliability, Matthew Flaming oversees New Relic's SRE practice. He believes site reliability engineering is perhaps "the purest distillation of DevOps principles into a single role. And it's something we try very hard to empower both internally in the organization and for our customers."

In a conversation with Flaming last year at FutureStack: New York, Liz Fong-Jones, a staff SRE at Google, helped extend this thought. While Google's software engineers always take responsibility for their code and reliability of their production systems, she said, SREs are charged with developing a particularly specialized depth of understanding of how different systems work together, how they fail, how they can be improved, and how they can best be designed and monitored—expertise that they must then share with their counterparts who are more focused on product development.

That latter part is the key reason SREs are so often embedded into product teams. SREs might take on software engineering tasks, but they're also charged with growing automation-first practices that reduce toil and improve reliability for the services offered by engineering teams. The fundamental goal is *greater reliability with less manual intervention as a system scales.*

The two axes of scaling

According to Flaming, there are two types, or "axes," of scale that software organizations must plan for. The first axis is workload—the number of physical hosts or virtual machines and other resources that must be able to grow efficiently in concert with the services that run on them. The second axis is *complexity*—the number of dependencies between those services and the growth of the organization itself. Fundamentally, site reliability engineering is about enabling both forms of scalability.



Fong-Jones supports that idea based on lessons learned during her early days at Google, circa 2009 while working on the Bigtable database service. Back then there was a lot of manual effort involved in running Bigtable as a shared service throughout the company, she said. The footprint was still relatively small, but it quickly became apparent that it would have trouble scaling. The non-SRE (or non-DevOps) solution would have been "Let's hire some more sysadmins to help handle this torrent of incoming tickets." But that approach is just throwing more people at a problem, not actually solving it for long-term scalability.

Scalability, Fong-Jones goes on to say, typically boils down to automation: "The evolution over 10 years of 'how do we go from handling dozens of footprints to hundreds or thousands of footprints?' is 'Let's automate.'"

Resource management, self-service provisioning, and other areas are important to scaling, too. Giving infrastructure ownership back to teams so that they had visibility into their workloads and could allocate and manage their own resources more effectively, rather than just firing off requests to an ops team, was a critical part of her experience on the Bigtable project, Fong-Jones said. The shift made those client teams happier, too, because they were more empowered to self-govern their infrastructure.

Self governance, according to Fong-Jones, requires creating standards for processes and tooling (covered later in this ebook). "One SRE team is going to have a really difficult time supporting 50 different software engineering teams if they're each doing their own separate thing and they're each using separate tooling," she said. Her team formalized the best practices that would enable them to scale and reach as many of their product development software engineering colleagues as possible.

CHAPTER 2

What Makes an SRE successful

What Makes an SRE Successful

First-class technical chops are obviously critical in most development and operations roles. But for 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 really thinking about the bigger picture outside of the day-to-day," says Jason Qualman, a site reliability 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 is going to affect the rest of a particular system, team, or the larger infrastructure is the kind of extreme pragmatism 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.

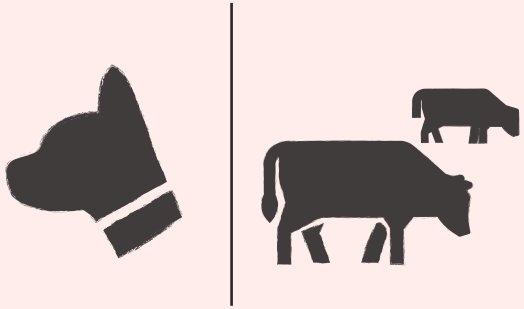
"We are making decisions very low in the stack," Qualman says of the SRE. Those decisions will affect people much further up the stack. Good decisions enable seamless transitions.

SREs automate at every opportunity

Top-notch SREs successfully increase the reliability of everything they touch without slowing their company's ability to ship software quickly. And they do it with automation—great SREs are proactive about automating away painful manual tasks and toil.

"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," Qualman says. "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.



Pets vs. Cattle

The importance of automation reflects the shift in how IT infrastructure is maintained in a modern software environment. You could treat all your servers as "pets," each with its own unique maintenance and configuration requirements, but today SRE and DevOps practitioners have evolved the best practice to treat servers as "cattle," where a herd of basically indistinguishable servers is configured en masse through automation. But servers aren't the only thing SREs automate—their goal is to automate all the little things that could get in their way: testing environments, packaging processes, user creation, bug tracking, and the list goes on...

SREs embrace new tools and approaches (when necessary)

Since 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 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 makeup for the SRE position, for example.

No matter your background, though, the SRE role should challenge you to move out of your comfort zone and develop new skills. An operations practitioner might benefit from learning a programming language or three, for instance, while someone with a development background should be willing and able to think much more deeply about operational processes and challenges than they did in the past. The best SREs embrace this kind of broad-based learning and skills development.

SREs are change agents

The confidence to advocate for SRE initiatives is another skill that distinguishes the best SREs. Great SREs live their own engineering-centric version of the self-help classic *How to Win Friends and Influence People.* 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.

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," says Beth Long.

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.

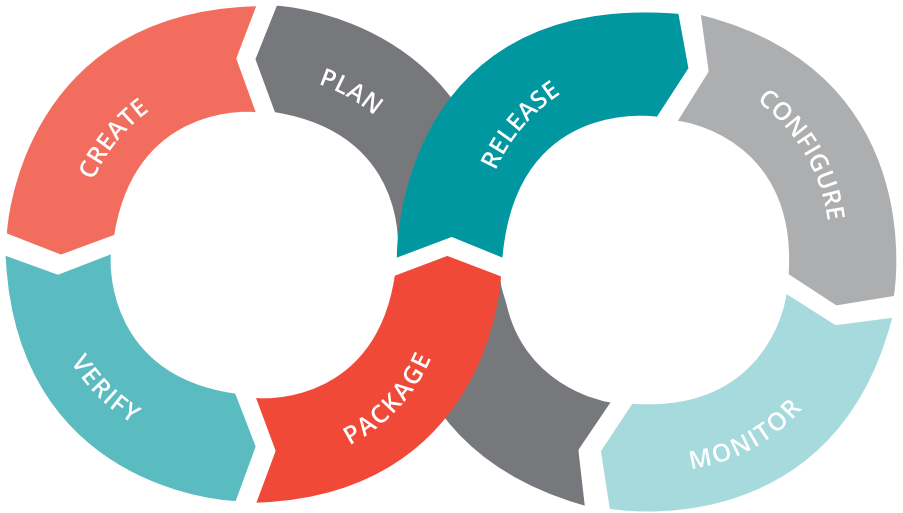
CHAPTER 3

SRE Tools and Processes

SRE Tools and Processes

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)

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), especially if you share Matthew Flaming's belief that site reliability engineering is really just DevOps expressed in its purest form.

At New Relic, SREs play an increasingly important role that combines responsibilities once siloed in traditional dev and ops teams. As a result, there isn't much difference between a "DevOps toolchain" and an "SRE toolchain," Flaming says.

Plan: Agile project management and tracking tools like JIRA or Pivotal Tracker, or other task management tools.

Create: Integrated development environments (IDEs), text editors, and shared libraries and components—as Henry Shapiro, vice president and general manager of New Relic Infrastructure, describes them: "the building blocks that you use to actually build applications." Even here, SREs have a role to play, such as encouraging development teams to avoid building everything from scratch in favor of reusing reliable code or third-party libraries.

Source-control tools like GitHub and Subversion help erase boundaries between dev and ops roles, and enjoy significant popularity among SREs tasked with managing deployment environments and processes.

Verify: Build and continuous integration/continuous delivery (CI/CD) tools like Jenkins or CircleCI.

Package: Tools to manage the build, packaging, release staging, and approval process of production-ready software, such as Rake or JFrog.

Release: Tools to manage releases and the lifecycle of an application, like Apache Maven or XebiaLabs.

Configure: Tools like Terraform and Ansible fit the "automate, automate, automate" SRE philosophy, and enable teams to automate and manage configurations across infrastructure and applications. SREs play an important role in determining what those configuration should look like from a health and reliability perspective. These tools also help automate away much of the manual work formerly needed to implement necessary rules and processes.

Both Shapiro and Qualman note that the increasing use of containers may ultimately reduce the need for these tools in many organizations. Because containerized applications include all of their dependencies and configurations in immutable configurations, container platforms like Docker and orchestration services like Kubernetes or Mesosphere are becoming indispensable to SREs

Monitor: Monitoring can mean a lot of things to a lot of people, but Shapiro notes that this stage includes tools like New Relic that collect metrics from applications and infrastructure in some form of log or analytics data, and alerts on that data via configurable dashboards and queries.

Using SLOs and SLIs to measure reliability

Service level objectives (SLOs) are a common way to measure the performance of a service provider 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 to meaningfully adjust those high-level reliability goals to fit company strategy.
- Maintain and build customers' confidence.
- Help teams decide when and how to focus efforts on reliability.
- Help engineers make better assumptions about risk tolerance and how fast they can go, as well as reason better about dependencies and reduce unnecessary toil.

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

Service level indicators (SLIs) can also be used to measure reliability. 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 repair (MTTR), and mean time to detect (MTTD), 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, but they also allow organizations to downshift on issues that may not be especially urgent.

Nothing is written in stone

The tools SREs use at any given time will depend on where an organization is at in their SRE journey. Less mature organizations will tend to use more specialized operations tools while 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, efficient ways to bring greater reliability to everything they do.

CHAPTER 4

The SRE Role at New Relic

The SRE Role at New Relic

Google's Site Reliability Engineering book does a great job of outlining what a great modern SRE practice should look like in a DevOps world. But what about SRE practices at companies that aren't the size of 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 relatively vague phrases like, "SREs combine software engineering and operational skill sets" and "SREs automate all the things."

Matthew Flaming has put a lot of thought into how New Relic defines the role so all stakeholders—from potential candidates to executive leadership—can understand exactly what New Relic expects for, and from, their SREs.

Defining the role

Creating the New Relic SRE description took time and involved input from individual SREs as well as executive leadership. This was a worthwhile investment, Flaming says, as the exercise helped clarify and shape a shared understanding of:

- Why SREs are important at New Relic.
- The vision for an SRE team.
- How SREs can most effectively contribute to the future of the New Relic platform.

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 to 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.

What SREs Do 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.	• Work with teams to update their risk matrices. • Audit for missing or outdated runbooks. • Influence teams to prioritize the most important reliability work.	• This is a particular focus of new SREs at New Relic and of SREs working with new teams. • All SREs stay current on platform tooling and SRE community best practices.
Stay current with the overall New Relic architecture and with the current state of, and top risks in, their teams' "neighborhood" in production.	• Meet with architects and SREs on other teams to discuss concerns and changes. • Use state-of-production knowledge to guide team risk matrices, operational processes, and priorities.	• All SREs should be familiar with the dependencies and underlying infrastructure of the systems they work with.
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 heavily lean toward using 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 a New Relic Infrastructure or a New Relic Insights integration 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.

What SREs Do at New Relic (cont'd)

Type of Work	Examples	Notes
Work with teams to design and implement automation, tooling, and application code to improve reliability and reduce toil.	• Identify a commonly used manual runbook and automate it with software. • Identify a common failure pattern for new deployments and implement a system to automatically detect and roll back that type of failed deploy. • Work with teams on the design of new services to ensure those services will be scalable and robust, and will integrate well with the rest of the platform. New services should leverage our best practices and share common components. • Update an application's DB connection pool to use a more reliable library.	• SREs actively participate in the design phase of new systems and features to help them be born reliable and operationally sane. • SREs drive systems toward requiring increasingly less human intervention; manual operations should become automated operations, which would then become automatic operations, requiring no human intervention. • In some cases, there's no distinction between SRE work and other application software engineering, apart from area of focus.
Mentor less senior SREs and grow the SRE community and practice at New Relic.	• Have a meeting, or lunch, once a week with a less senior SRE to discuss work challenges and solutions. • Pair with other SREs experiencing problems you've previously encountered or solved. • Document and share novel solutions and other effective strategies.	• All SREs should have an SRE mentor or mentee. Mentor/mentee relationships are not team dependent. • SREs can also have non-SRE mentors.
Perform task-based operational work (toil) required to unblock teams with operational needs where automated or self-service solutions do not yet exist for those teams.	• Track down hardware defects on servers. • Provision new network endpoints. • Run Ansible playbooks.	• This is the lowest-value type of work, and SREs should not spend more than 40% of their time on this category of work (30% for senior level or higher SREs). Ongoing issues in such areas should be escalated through the appropriate management channels. • SREs should proactively look to reduce toil through automation whenever possible.

Set your SREs up for success

Although this SRE role description 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, can set SREs up for success and advance the collective understanding of the key 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 share best practices with each other—that's what really creates a culture of reliability.

Conclusion

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 like empathy and curiosity.

Some engineering cultures, like New Relic's, prize autonomy, but that doesn't mean teams should have to tackle reliability on their own. Teams (and individual SREs) need organizational support, communication, and, above all, trust in order 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 constantly striving to do things better is the way to get as close as possible.