The role and responsibilities of SREs in software engineering

company, there are some general trends for how SRE teams tend to organize themselves. This article focuses on how SRE teams share responsibilities across members while at the same time recognizing the strengths each member brings to the team as they work towards a common reliability goal.

When someone becomes an SRE, they already have a level of specialized knowledge and a work history in a handful of technical disciplines that are critical to keeping distributed systems online and reliable.

In other posts we describe the qualities of a Top-Notch SRE as well as what Site Reliability Engineering is, this article dives deeper into the various roles that engineers fill within SRE teams, even as all members share the ultimate responsibility for constantly maintaining and increasing the reliability of their system or service.

What Does an SRE Team Do?

The role of Site Reliability Engineering in an organization is to keep the organization focused on what ultimately matters to customers: making sure the platforms and services customers rely on are available when customers want to use them. This is different from DevOps, which is more of a philosophy of removing silos and sharing responsibilities for a faster deployment cadence with a wide range of implementation styles available.

In other words, an SRE's purpose is to keep the site reliable. This mandate informs their priorities and approaches to their work.

A Site Reliability Engineering team prioritizes reliability work using service level objectives (SLOs). SLOs are specific means of measuring the performance of the site, system, or service and are based on service level indicators (SLIs), which measure the service level provided to customers. Ultimately, these are used to create service level agreements (SLAs), which are the promises made to customers. The SLAs are the most lenient and the SLOs the most strict, to make sure we hold ourselves to a higher level of accountability than we promise our customers.

serve as a warning that the system behavior is getting close to the SLA and requires immediate attention to restore healthy operations.

A good example of something you might use as a metric here is request latency. This is how long it takes for a service to return a response to a request. Some processes are more complex and take longer, but ideally a response should come as fast as possible. After measuring the service under normal operating conditions and determining what a normal response time should be (this is the SLI), the SRE team can then decide how much performance degradation is acceptable when the service is under a heavy load and response times begin to slow down. That is used to create the SLA, the promise the company makes to customers.

The SLA will be set using a latency number that is padded slightly with a cushion that the team can meet and is still acceptable to the customer. Then, the team sets an SLO that uses a number that is better than the SLI, aiming to improve on the existing performance and keep latency down to a level far safer and better. SRE teams always push themselves and the technology they manage to improve, which is important because customers expect more and more each year, meaning that our SLAs become more strict over time and we must, therefore, adjust our SLOs and internal commitments to compensate and keep ahead of customer expectations.

These metrics are agreed-upon by the team and then set an *error budget*, which is the quantified amount of downtime or lowered performance levels that the team is willing to allow in a 30 day period. Teams know that unexpected failures happen and that we should expect nothing to have perfect 100% uptime. The goal for the team is to always exceed the SLO parameters, aiming to spend less than the error budget allows, and use SLIs to measure success.

With this in mind, SRE teams measure, and test, and explore their systems. Whatever is found to cause the most trouble is where work is prioritized for improvement. Going back to our example, if an SLO was set for latency and the service performs more poorly than the level that was agreed upon, fixing and improving this will take priority over further optimizing a service that is currently meeting its SLOs. The goal is to minimize long-term toil like repetitive and reactive work through automation and enhanced efficiency while reducing the cost of failure. Chaos Engineering is useful here both to test that enhancements and fixes work as well as to find potential problems before they occur. Proactively seeking ways to improve reliability is a vital part of SRE, as Backcountry discovered as they improved from a major SLO issue on Black Friday in 2018 to having zero incidents on the same day in 2019.

Determined Within a Team?

In an SRE team, all team members share responsibility for system maintenance, incident management, automation, and Chaos Engineering. Anything a team member can do that contributes to enhanced system reliability and availability is within scope. Anything that does not contribute to that bottom line is out of scope.

Each person who joins an SRE team comes with a past and with unique skills that fill in knowledge or experience gaps within the team. Some companies, like Google, hire SREs based on how well the person would fit in the SRE culture and how deep and useful their individual skill set seems. Only *after* being hired is the person assigned to an SRE team, typically a team that the company knows could benefit from the new hire's skills.

SREs get hired because they are smart, imaginative, and capable and also because they have a strong interest in large-scale distributed systems. Any of the following types of technical skills and experience can benefit an SRE team, if the candidate who is experienced in some of the following disciplines is also someone who likes to continually learn and grow. See our sample SRE job description and interview questions article for an example.

Systems administrators, systems engineers, and **operations experts** have deep and comprehensive knowledge of operating systems, especially multi-user systems like servers. They know how to make software run and stay running on those systems. They also know how to physically build, configure, wire up, and install hardware and can also do analogous tasks in virtual settings. **Linux and networking admins** are especially beloved and valued when they understand parts of the system that others find mysterious and arcane.

People with experience performing operations functions are great to have around when you need someone who knows how to quickly handle things like runaway processes on a server that you don't want to or can't reboot because it will cause an even larger problem. Their perspective is useful in a crisis as well as for those times when the team is creating runbooks or designing chaos experiments.

Software developers and software engineers love to solve puzzles. They create solutions that fulfill the business needs. They understand how to instruct a computer to perform needed work. In a development team, most of the job involves listening, reading, and analyzing what is needed. They then use their problem-solving skills to write code and code tests while planning how to deploy and

Quality assurance (QA) and testing engineers create and implement strategies for finding flaws and problems before software is deployed to production systems. They are experts at automation, at designing tests, and at imagining potential problem areas and attack vectors. Their goal of ensuring consistent product quality marries well with the SRE goals and their experience helps them fit in to SRE teams quickly. They are often among the most astute chaos experiment designers and help teams find ways to be even more intentional about finding and fixing problems before anyone else does. Pair this with the slow move away from QA testing, which really came of age during the era of waterfall software development, and moving from a dying field into the new frontier of testing at runtime and it makes a lot of sense for testing experts to apply their insights as SREs.

Build automation pipeline experts are generally a special subset of software engineers who have experience across the entire cycle from code development all the way to publishing to production environments. They create automated build triggers like git hook scripts that activate when new code is merged into the software repository to begin an automated build process. That build process includes automated unit and function tests and ultimately automated deployment. In the past they have monitored and measured their process and improved it based on the bottlenecks and inefficiencies they have found, without compromising output quality. They make great SRE team members because they know where to look when builds fail, how to repair and enhance a continuous integration/continuous delivery (CI/CD) pipeline, and are already experts at automating time-consuming workloads.

Database administrators (DBAs) think critically about data sets and how to model them both for usefulness by humans and also for efficient storage, organization, and access issues in computing systems. Their jobs involve capacity planning, database schema creation and structure adjustments, data security and access control, data query creation, database software maintenance, data backups and archive maintenance, database software failover schemes, and performance optimization.

If it involves data, there is probably a DBA involved. And that's a good thing because really, all computers do is receive input data, process it according to program instructions, and output the processed data. SRE teams need database experts, even with the rise of database as a service (DBaaS) offerings. As former DBAs join SRE teams, they bring a powerful foundation and perspective to the world of big data, myriad database types, and the need to combine data from disparate sources. SRE teams benefit from the DBA focus on data security and reliability and also their focused knowledge of how to fix or restore data access when problems occur.

team member does toward that end is divvied up within the team. Sometimes team members focus their work using their strongest skills and experience. At other times, team members are given a safe space to learn new skills and gain new experiences by working on projects outside of their main skill set, perhaps alone or while paired with an experienced coworker.

Cross-pollination and mutual teaching are common SRE team traits. Team members continually help each other grow and strive to learn from one another. This empowers SRE teams to consistently outperform other types of enterprise software development and support teams on the main objective of keeping sites reliable. When a crisis arises, team members often focus their efforts on areas of their greatest strengths, but not always. They don't need to. The next section explains why.

Which SREs are Responsible for Incident Management?

In short, all of them. Every member of the SRE team is responsible for incident response and management and for reducing the time spent to fix problems when they arise. It is the team that is responsible for meeting the SLOs that have been committed to. The good news is that SRE teams are set up for maximum success because of this combination of shared responsibilities and diverse membership.

Who Addresses Service Disruptions and Downtime?

In the old days, whoever was responsible for site or system uptime or recovering the site from an outage would carry around a pager that would make unpleasant sounds to interrupt their lives at the first sign of an incident. When problems would arise, someone at the company would dial a phone number and the pager would sound an alarm telling the person carrying it to call in and find out what's going on.

It's simpler today, but the slang term *pager duty* remains. A person who has pager duty is on call. They are the person or people responsible for immediate incident response. For mission-critical systems, it is more common to have on-call shifts, where specific team members are designated as the point people for incidents during their work hours, rather than having people off-site being

Most SRE teams aim for a 50-50 balance of on-call duty and project work. About half of any team member's time is specifically focused on answering and dealing with any incident that comes up and the other half of their time is for working on projects related to things like software feature improvements, site efficiency tweaks, and so on.

Everyone takes a turn in the on-call rotation. On-call duty can be stressful. Good SRE managers aim to limit the amount of calls assigned to any one person and make sure that they also have time for adequate follow-up, such as writing up incident reports, planning and holding blameless retrospectives (also called postmortems), and thinking about long-term prevention solutions to keep the same incident from recurring. They also set a goal of making sure that every team member uses the same processes, procedures, and escalation paths, all of which are clearly written down and easily accessible.

Do SREs Follow a Process or Procedure?

Yes. Multiple and it's important that these processes are routinely maintained and kept up to date. There is a process for training and practice before going on-call for the first time. There is also a runbook, also called a playbook, for each of the various alerts that might be received containing high-level instructions for how to respond. This section details both.

For training and practice before going on-call the first time, there is generally a set of focus areas for people to practice. Google includes entries like these in theirs, from their book, *The Site Reliability Workbook*:

- Administering production jobs
- Understanding debugging info
- "Draining" traffic away from a cluster
- Rolling back a bad software push
- Blocking or rate-limiting unwanted traffic
- Bringing up additional serving capacity
- Using the monitoring systems (for alerting and dashboards)
- Describing the architecture, various components, and dependencies of the services

tutorials created and designed by the team. Learners present what they are learning to the rest of the team in short sessions, which gives an opportunity for Q&A and gentle corrections and guidance from teammates.

One useful tool during onboarding are the runbooks that exist primarily for use when alerting indicates a potential incident. Using runbooks to help onboard new team members gives them direct experience checking and exploring the things they will need to examine during an on-call incident.

It is the responsibility of each SRE team to create and maintain the team's runbooks for the system or service they manage. Runbooks are important because they document useful diagnostic procedures and places to look for insight or information during an incident to reduce mean time to repair (MTTR) by eliminating the brain fog of stress as a factor in response time. A problem comes up, here's what we check, here's what we do, done.

Runbooks are written by team members. This is where the diverse backgrounds seen in an SRE team can again benefit the entire program. Different people will have knowledge of potential problem spots and pain points and each can contribute to the documentation. The possibilities can be ranked by likelihood of usefulness by the team, giving multiple options.

For major incidents, the work may be parallelized having multiple team members each exploring a different problem vector. When everything in the runbook is tried, having diverse backgrounds lends to faster resolution as team members will talk and perhaps stumble upon imaginative options that wouldn't have occurred to any of them alone.

In addition, there are defined processes and policies for normal, everyday activities like configuration management, provisioning of resources, and coding standards, just to keep things standardized and working as smoothly as possible.

Again, these processes grow stale as systems and environments evolve so they should be routinely maintained. Chaos Engineering is a useful tool to test the utility of runbooks and incident management processes.

Learn 9 SRE best practices for incident management

Read the Guide →

How do SRE Roles Differ from DevOps Roles?

DevOps team members tend to specialize into roles like these:

- Product owner, managing and coordinating the product and interacting with customers
- Team lead, in charge of delegating responsibilities across the team
- Software developer, writing code and unit tests
- Cloud or system architect, designing and thinking about apps and services in production and making sure infrastructure needs are met
- Monitoring and alerting leader, who designs and watches for alerts
- Automation expert, who is responsible for the CI/CD pipeline and other processes and tooling

This is not an exhaustive list of roles you may find in a DevOps team. However, it does help illustrate the main difference between SRE roles and DevOps roles. In a DevOps team, members tend to specialize, not to the extreme of waterfall software development siloing of development teams and operations teams where they barely speak to one another, but in the sense that team members generally have one role and work in that most of the time.

A Site Reliability Engineering role may change over time as it is common for team members to move around from responsibility to responsibility, learning each aspect of what the team itself is responsible for. While DevOps team members often have titles corresponding to the role they fill in the team, SRE team members are typically called Site Reliability Engineers.

One interesting development in the enterprise space is that some companies create DevOps teams and include people on those teams with the SRE title and the typical "keep the site running and available" job description. However, these SREs frequently don't get the same experience as fully-committed all-SRE teams. They don't develop most of the code. They don't control all the build pipelines. They have a portion of the responsibility range, but without the insights that working across a set of major roles provides. Instead, they are treated like the on-call engineers instead of

In SRE, the job is to keep things running and running better and better all the time. Cross training throughout all the various responsibilities helps with that focus. The similarities and differences between SRE and DevOps are covered in greater detail in Site Reliability Engineering vs DevOps - Can they Coexist or do they Compete?

Which SREs Write and Review Code?

Everyone. All of them. Some may be asked to be the lead on one particular thing, but they will still have insight and access to other code in use across the team's responsibilities. Why? Because everyone takes turns being on-call, supporting the entire system or service and knowing where code exists and what it does is vital to making good, quick, and clear-headed decisions in a crisis.

Let's think about the types of code that an SRE might encounter on the job. This can include everything from compiled code to shell scripts. A broad range of experience is helpful when a person is hired, and either way a broad range of knowledge will be acquired on the job.

First, you have the application or service that the team supports. This is generally written in a language like Java, Rust, Go, or even C, often with differences for front and back end needs. There will be bug fixes, feature improvements and additions, and efficiency or reliability improvements to write and to review.

Next, there is code for automation of various tasks. You will see code for controlling cloud platform server creation and destruction for capacity management and cost control as traffic waxes and wanes, load balancing and automated failover for helping improve high availability (HA) metrics. Some of this depends on your tools and may consist of static configuration files written in something like YAML with modules written in languages like Ruby and Python. Your build pipeline may use languages like Groovy or Clojure. You are likely to find shell scripts as well for many diverse purposes.

An SRE team may decide to standardize on one or two programming languages that everyone likes and knows. Alternatively, a team may inherit or create a system that uses multiple languages in multiple places, each chosen because a certain language is more efficient for a specific task while others work better for other tasks. So much depends on the people in the team and the system they support.

Engineering?

When an SRE culture exists at a company, then it is already a sign that the enterprise cares deeply about how their systems and services operate and see the relationship between business health and reliability. SRE teams are constantly tuning and testing the operational readiness and efficiency of their services today. Chaos Engineering is an important tool in their toolkit.

Chaos Engineering is a means of helping SRE teams build confidence in the real world behavior of their systems. It is impossible to accurately test how an updated service will function in a large-scale system with thousands of moving parts with unit and integration testing alone. You need to run chaos experiments to see how the system will behave under emergent behavior at runtime. Run as many of these tests in staging as possible, and then graduate your testing into the real production environment because, ultimately, production is vastly different and we really can't be sure about how a service will perform under real-world stress unless we test it in the real-world.

This is where things like canary deployments, where you push only one or a small number of instances of a new microservice into production and examine how it works alongside existing older-release instances. Here you can test whether bug fixes perform as needed in production, how new features measure up to expectations.

Chaos Engineering allows you to build a view of how your system behaves by using controlled experiments. These experiments will temporarily stress your system to simulate how it responds to different conditions. By running these controlled experiments as part of your operations you'll be able to identify issues and weaknesses that traditional unit and integration testing simply won't see.

You can test aspects of your existing system components by injecting small failures or problems into your networking, resource usage, or available capacity to see how the system holds up to stress, before real-world incidents push components past their breaking points and cause outages.

The value of SRE and Chaos Engineering is a shared focus on the customer's perspective and what is right and good for the greater business. This table shows how moving from left to right provides ever-improving reliability and happier customers.

Technology Team Problem	DevOps Philosophy	SRE Prescription	Chaos Engineering (CE) Prescription

Working in silos	Reduce Organizational Silos	Share Code/Service Ownership	GameDaysOn-Call Training using CE
Scared of failure	Accept Failure as Normal	Chaos Engineering Failure InjectionSLOsBlameless Postmortems	Chaos BootcampsIncident Reproduction
No Change/Too Much Change	Implement Gradual Change	Reduce costs of failureError Budgets	Chaos Engineering BudgetRunbook validation using CE
Too much manual work/toil	Leverage Tooling & Automation	Automate this year's job away	Automated Chaos EngineeringCI/CD Chaos Engineering
No measurement /metrics	Measure Everything	Measure reliabilityMeasure durabilityMeasure toil	Measure time to detectionMeasure time to attackMeasure time to recoveryMeasure time to halt

You can learn more about Chaos Engineering and ask questions of real-world practitioners in the Gremlin-sponsored Chaos Engineering Slack, which has participants from across the industry well beyond Gremlin.

Why You Should Join an SRE Team

The ability to fill a specialized role within a team that shares generalized responsibility is a powerful way to enable SREs to keep reliability the focus, no matter the task at hand. People with experience in any of the areas that are useful in achieving reliability have the potential to be valuable team members and are in great demand. In addition, SREs are paid quite well, making this not only an interesting career, but also one that provides good compensation packages.

Guide Chapters

A primer on SRE for engineering leaders

Site Reliability Engineering (SRE) is the outcome of combining IT operations responsibilities with software development. With SRE there is an inherent expectation of responsibility for meeting the service-level objectives (SLOs) set for the service they manage and the service-level agreements (SLAs) we promise in our contracts.

SRE vs DevOps: Can they coexist or do they compete?

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.

How to become a top notch SRE

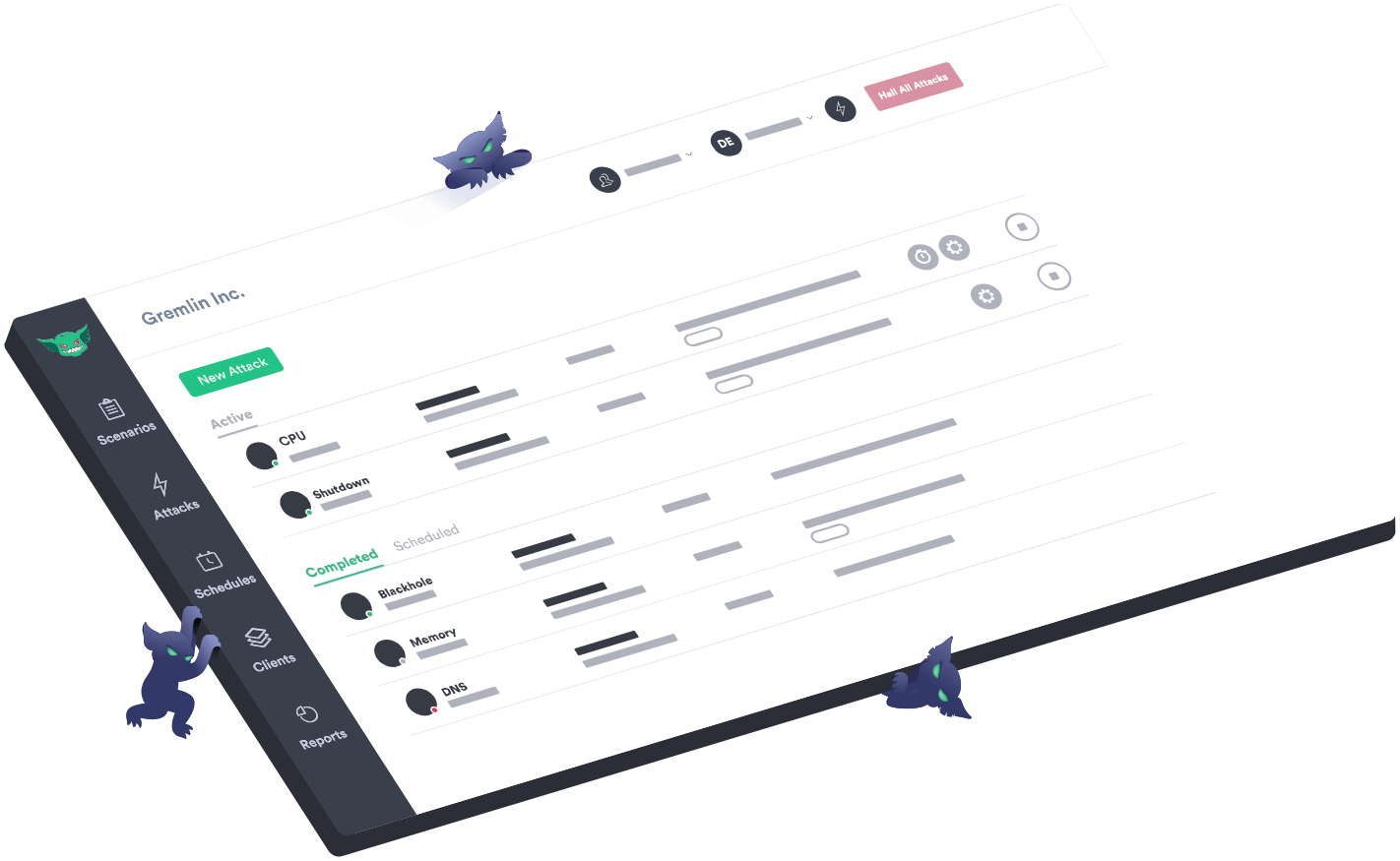
and now that you have heard about Site Reliability Engineering (SRE) you think this sounds like something you would like to do as your next step. This article will help you learn in greater detail what you need to know to not only be successful, but one of the best SREs.

How much money do SREs make?

Wondering about the average Site Reliability Engineer salary? Or how much top-notch SREs at best-in-class organizations are compensated? We did some research and are sharing our findings here.

SRE interview questions and job descriptions

What do Site Reliability Engineers do and what exactly are they responsible for within an engineering organization? While the specifics will depend on your company, there are some general trends for how SRE teams tend to organize themselves. This article focuses on how SRE teams share responsibilities across members while at the same time recognizing the strengths each member brings to the team as they work towards a common reliability goal.



COMPANY RESOURCES

Contact

Press

Privacy

Security

INDUSTRIES

SaaS

Finance

Retail

FEATURED

What is Chaos Engineering?

What is Chaos Monkey?

Reliability Calculator

What is Site Reliability Engineering?

ROI Calculator

The 2021 State of Chaos Engineering Report

How to achieve reliability in distributed systems

All systems operational