How documentation enables SRE teams to manage new and existing services.

BY SHYLAJA NUKALA AND VIVEK RAU

Why SRE Documents Matter

SITE RELIABILITY ENGINEERING (SRE) is a job function, a mind-set, and a set of engineering approaches for making Web products and services run reliably. SREs operate at the intersection of software development and systems engineering to solve operational problems and engineer solutions to design, build,

and run large-scale distributed systems scalably, reliably, and efficiently.

SRE core functions include:

- **Monitoring and metrics:** Establishing desired service behavior, measuring how the service is actually behaving, and correcting discrepancies.

- **Emergency response:** Noticing and responding effectively to service failures in order to preserve the service's conformance to its SLA (service-level agreement).

- **Capacity planning:** Projecting future demand and ensuring that a service has enough computing resources in appropriate locations to satisfy that demand.

- **Service turn-up and turn-down:** Deploying and removing computing resources for a service in a data center in a predictable fashion, often as a consequence of capacity planning.

- **Change management:** Altering the behavior of a service while preserving service reliability.

- **Performance:** Design, development, and engineering related to scalability, isolation, latency, throughput, and efficiency.

SREs focus on the life cycle of services—from inception and design, through deployment, operation, refinement, and eventual decommissioning.

Before services go live, SREs support them through activities such as system design consulting, developing software platforms and frameworks and capacity plans, and conducting launch reviews.

Once services are live, SREs support and maintain them by:

- **Measuring and monitoring** availability, latency, and overall system health.

- Reviewing planned system changes.
- Scaling systems sustainably through mechanisms such as automation.
- Evolving systems by pushing for changes that improve reliability and velocity.
- Conducting incident responses and blameless postmortems.

Once services reach end of life, SREs decommission them in a predictable fashion with clear messaging and documentation.

A mature SRE team likely has well-defined bodies of documentation associated with many SRE functions. If you manage an SRE team or intend to start one, this article will help you understand the types of documents your team needs to write and why each type is needed, allowing you to plan for and prioritize documentation work along with other team projects.

A SRE's Story

Before discussing the nuances of SRE documentation, let's examine a night and day in the life of Zoë, a new SRE.

Zoë is on her second on-call shift as an SRE for Acme Inc.'s flagship AcmeSale product. She has been through her induction process as a team member, where she watched her colleagues while they were on-call, and she took notes as well as she could. Now she has the pager.

As luck would have it, the pager goes off at 2:30 A.M. The alert says "Ragnarok job flapping," and Zoë has no idea what it means. She flips through her notes and finds the link to the main dashboard page. Everything looks OK. She does a search on the Acme intranet to find any document referencing Ragnarok, and after precious minutes go by, she finds an outdated design document for the service, which turns out to be a critical dependency for AcmeSale.

Luckily, the design document links to a "Ragnarok Ops" page, and that page has links to a dashboard with charts that look like they might be useful. One of the charts displays a traffic dip that looks alarming. The Ops page also references a script called ragtool that can apparently fix problems like the one she is seeing, but this is the first time she has heard of it. At this point, she pages the backup on-call

SRE for help because he has years of experience with the service and its management tools. Unfortunately, she gets no response. She checks her email and finds a message from her colleague saying he is offline for an hour because of a health emergency. After a moment of inner debate, she calls her tech lead, but the call goes to voicemail. It looks like she has to tackle this on her own.

After more searching to learn about this mysterious ragtool script, she finds a document with one-line descriptions of its command-line options, which also tells her where to find the script. She runs `ragtool-restart` and crosses her fingers. Nothing changes, and in fact the traffic drops even more. She reads frantically through more command-line options but is not sure whether they will do more harm than good. Finally, she concludes that `ragtool-ebalance e-dc=atlanta` might help, since another chart indicates that the Atlanta data center is having more trouble. Sure enough, the line on the traffic chart starts creeping upward, and she thinks she is out of the woods. MTTR (mean time to repair) is 45 minutes.

The next day Zoë has a postmortem discussion about the incident with her team. They are having this discussion because the incident was a major outage causing loss of revenue, and their manager has been asking them to do more postmortems. She asks the team how they would have handled the situation differently, and she hears three different approaches. There appears to be no standard troubleshooting process. Her colleagues also acknowledge that the "flapping" alert is poorly named, and that the failure was a result of a well-known bug in the product that hasn't been a high priority for the developer team.

Finally, Steve, her tech lead, asks, "Which version of ragtool did you use?" and then points out that the version she used was very old. A new release came out a week ago with brand-new documentation describing all its new features and even explaining how to fix the "Ragnarok job flapping" problem. It might have reduced the MTTR to five minutes.

The existence of the new version of ragtool comes as a surprise to about half the team, while the other half is somehow familiar with the new version

and its user guide. The latest script and document are both under Steve's home directory, in the bin/folder, of course. Zoë writes this down in her notes for future reference, hoping devoutly that she will get through this shift without further alerts. She wonders whether her tech lead or anyone else will follow up on the problems uncovered during the postmortem discussion, or whether future SREs are doomed to repeat the same painful on-call experience.

Later that day Zoë attends an SRE onboarding session, where the SRE team meets with a product development team to talk about taking over their service. Steve leads the meeting, asking several pointed questions about operational procedures and current reliability problems with the service, and asking the developer team to make several operational and feature changes before the SRE team can take it over. Zoë has been to a few such meetings already, which are led either by Steve or another senior SRE. She realizes the questions asked and the actions assigned to the developers seem to vary quite a bit, depending on who is leading the meeting and what types of product failures the SRE team has dealt with in the past week.

She wishes vaguely that the team had more consistent standards and procedures but doesn't quite know how to achieve that goal. Later, she hears two of the developers joking near the coffee machine that many of the questions seemed quite unrelated to carrying a pager, and they had no idea where those questions came from. She wishes product development teams could understand that SREs do a lot more than carry pagers. Back at her desk, however, Zoë finds several urgent tickets to resolve, so she never follows up on those thoughts.

Luckily, all the characters and episodes in this story are fictional. Still, consider whether any part of the story resembles any of your real-life experiences. The solution to this fictional team's struggles is entirely obvious, and the next section expands on this solution.

The Importance of Documentation

In the early stages of an SRE team's existence, the organization depends

heavily on the performance of highly skilled individuals on the team. The team preserves important operational concepts and principles as nuggets of “tribal knowledge” that are passed on verbally to new team members. If these concepts and principles are not codified and documented, they will often need to be relearned—painfully—through trial and error. Sometimes team members perform operational procedures as a strict sequence of steps defined by their predecessors in the distant past, without understanding the reasons these steps were initially prescribed. If this is allowed to continue, processes eventually become fragmented and tend to degenerate as the team scales up to handle new challenges.

SRE teams can prevent this process decay by creating high-quality documentation that lays the foundation for such teams to scale up and take a principled approach to managing new and unfamiliar services. These documents capture tribal knowledge in a form that is easily discoverable, searchable, and maintainable. New team members are trained through a systematic and well-planned induction and education program. These are the hallmarks of a mature SRE team.

The remainder of this article describes the various types of documents SREs create during the life cycle of the services they support.

Documents for New Service Onboarding

SREs conduct a production readiness review (PRR) to ensure a service meets accepted standards of operational readiness, and that service owners have the guidance they need to take advantage of SRE knowledge about running large systems.

A service must go through this review process prior to its initial launch to production. (During this stage, the service has no SRE support; the product development team supports the service.) The goal of the prelaunch PRR is just to ensure the service meets certain minimum standards of reliability at the time of its launch.

A follow-on PRR can be performed before SRE takeover of a service, which may happen long after the ini-

tial launch. For example, when an SRE team decides to onboard a new service, the team conducts a thorough review of the production state and practices of the new service. The goals are to improve the service being onboarded from a reliability and operational sustainability perspective, as well as to provide SREs with preliminary knowledge about the service for its operation.

SREs conducting a PRR before service takeover may ask a more comprehensive set of questions and apply higher standards of reliability and operational ease than when conducting a PRR at the time of the initial launch. They may intentionally keep the launch-time PRR “lighter” than the service take-over PRR in order to avoid unduly slowing down the developer team.

In Zoë’s SRE story, her team had no standardized PRR process or checklist, which means they might miss asking important questions during service takeover. Therefore, they run the risk of encountering many problems while managing a new service that were easily foreseeable and could have been addressed before SREs became responsible for running the service.

An SRE PRR/takeover requires the creation of a PRR template and a process doc that describes how SRE teams will engage with a new service, and how SRE teams will use the PRR template. The template used at the time of takeover might be more comprehensive than the one used at the time of initial launch.

A PRR template covers several areas and ensures that critical questions

about each area are answered. The accompanying table lists some of the areas and related questions that the template covers.

The process doc should also identify the kinds of documentation that the SRE team should request from the product development team as a prerequisite for takeover. For example, they might ask the developer team to create initial playbook entries for standard problems.

In addition to these onboarding documents, the SRE organization must create overview documents that explain the SRE role and responsibilities in general terms to product development teams. This serves to set their expectations correctly. The first such document would explain what SRE is, covering all the topics listed at the beginning of this article, including core functions, the service life cycle, and support/maintenance responsibilities. A primary goal of this document is to ensure developer teams do not equate SREs with an Ops team or consider pager response to be their sole function. As shown in the earlier SRE story, when developers do not fully understand what SREs do before they hand off a service to SREs, miscommunication and misunderstandings can result.

Additionally, an engagement model document goes a little further in setting expectations by explaining how the SRE team will engage with developer teams during and after service takeover. Topics covered in this doc include:

- Service takeover criteria and the PRR process;

Example PRR template areas

Area	Questions
Architecture and dependencies	What is your request flow from user to front end to back end? Are there different types of requests with different latency requirements?
Capacity planning	How much traffic and rate of growth do you expect during and after the launch? Have you obtained all the compute resources needed to support your traffic?
Failure modes	Do you have any single points of failure in your design? How do you mitigate unavailability of your dependencies?
Processes and automation	Are any manual processes required to keep the service running?
External dependencies	What third-party code, data, services, or events do the service or the launch depend upon? Do any partners depend on your service? If so, do they need to be notified of your launch?

- ▶ SLO negotiation process and error budgets;
- ▶ New launch criteria and launch freeze policy (if applicable);
- ▶ Content and frequency of service status reports from the SRE team;
- ▶ SRE staffing requirements; and,
- ▶ Feature roadmap planning process and priority of reliability features (requested by SREs) versus new product functionality.

Documents for Running a Service

The core operational documents SRE teams rely on to perform production services include service overviews, playbooks and procedures, postmortems, policies, and SLAs. (Note: this section appeared in the “Do Docs Better” chapter of *Seeking SRE*.¹)

Service overviews are critical for SRE understanding of the services they support. SREs need to know the system architecture, components and dependencies, and service contacts and owners. Service overviews are a collaborative effort between the development team and the SRE team and are designed to guide and prioritize SRE engagement and uncover areas for further investigation. These overviews are often an output of the PRR process, and they should be updated as services change (for example, new dependency).

A basic service overview provides SREs with enough information about the service to dig deeper. A complete service overview provides a thorough description of the service and how it interacts with the world around it, as well as links to dashboards, metrics, and related information that SREs need to solve unexpected issues.

Playbook. Also called a *runbook*, this quintessential operational doc lets on-call engineers respond to alerts generated by service monitoring. If Zoë’s team, for example, had a playbook that explained what the “Ragnarok job flapping” alert meant and told her what to do, the incident could have been resolved in a matter of minutes. Playbooks reduce the time it takes to mitigate an incident, and they provide useful links to consoles and procedures.

Playbooks contain instructions for verification, troubleshooting, and escalation for each alert generated from network-monitoring processes. Playbooks typically match alert names

generated from monitoring systems. They contain commands and steps that need to be tested and reviewed for accuracy. They often require updates when new troubleshooting processes become available and when new failure modes are uncovered or dependencies are added.

Playbooks are not exclusive to alerts and can also include *production procedures* for pushing releases, monitoring, and troubleshooting. Other examples of production procedures include service turnup and turndown, service maintenance, and emergency/escalation.

Postmortem. SREs work with large-scale, complex, distributed systems, and they also enhance services with new features and the addition of new systems. Therefore, incidents and outages are inevitable given SRE scale and velocity of change. The postmortem is an essential tool for SRE, representing its formalized process of learning from incidents. In the hypothetical SRE story, Zoë’s team had no formal postmortem procedure or template and, therefore, no formal process for capturing the learning from an incident and preventing it from recurring, so they are doomed to repeat the same problems.

SRE teams need to create a standardized postmortem document template with sections that capture all the important information about an outage. This template will ideally be structured in a format that can be readily parsed by data-analysis tools that report on outage trends, using postmortems as a data source. Each postmortem derived from this template describes a production outage or paging event, including (at minimum):

- ▶ Timeline;
- ▶ Description of user impact;
- ▶ Root cause; and,
- ▶ Action items/lessons learned.

The postmortem is written by a member of the group that experienced the outage, preferably someone who was involved and can take responsibility for the follow-up. A postmortem needs to be written in a blameless manner. It should include the information needed to understand what happened, as well as a list of action items that would significantly reduce the possibility of recurrence, reduce the impact, and/or make recovery more straightforward. (For guidance on

writing a postmortem, see the postmortem template described in *Site Reliability Engineering*.²)

Policies. Policy documents mandate specific technical and nontechnical policies for production. Technical policies can apply to areas such as production-change logging, log retention, internal service naming (naming conventions engineers should adopt as they implement services), and use of and access to emergency credentials.

Policies can also apply to process. Escalation policies help engineers classify production issues as emergencies or non-emergencies and provide recommendations on the appropriate action for each category; on-call expectations policies outline the structure of the team and responsibilities of team members.

Service-level agreement. An SLA is a formal agreement with a customer on the performance a service commits to provide and what actions will be taken if that obligation is not met. SRE teams document their service(s) SLA for availability and latency, and monitor service performance relative to the SLA.

Documenting and publishing an SLA, and rigorously measuring the end-user experience and comparing it with the SLA, allows SRE teams to innovate more quickly while preserving a good user experience. SREs running services with well-defined SLAs will detect outages faster and therefore resolve them faster. Good SLAs also result in less friction between SRE and software engineer (SWE) teams because those teams can negotiate targets and results objectively, and avoid subjective discussions of risk.

Note that having an external legally enforceable agreement may not be applicable to most SRE teams. In these cases, SRE teams can go with a set of service-level objectives (SLOs). An SLO is a definition of the desired performance of a service for a single metric such as availability or latency.

Documents for production products. SRE teams aim to spend 50% of their time on project work, developing software that automates away manual work or improves the reliability of a managed service. Here, we describe documents that are related to the products and tools SREs develop.

These documents are important because they enable users to find out whether a product is right for them to

adopt, how to get started, and how to get support. They also provide a consistent user experience and facilitate product adoption.

An **About page** helps SREs and product development engineers understand what the product or tool is, what it does, and whether they should use it.

A **concepts guide** or glossary defines all the terms unique to the product. Defining terms helps maintain consistency in the docs and UI, API, or CLI (command-line interface) elements.

The goal of a **quickstart guide** is to get engineers up and running with a minimum of delay. It is helpful to new users who want to give the product a try.

Codelabs. Engineers can use these tutorials—combining explanation, example code, and code exercises—to get up to speed with the product. Codelabs can also provide in-depth scenarios that walk engineers step by step through a series of key tasks. These tutorials are typically longer than quickstart guides. They can cover more than one product or tool if they interact.

How-to guide. This type of document is for users who need to know how to accomplish a specific goal with the product. How-tos help users complete important specific tasks, and they are generally procedure based.

The **FAQ** page answers common questions, covers caveats that users should be aware of, and points users to reference documents and other pages on the site for more information.

The **support** page identifies how engineers can get help when they are stuck on something. It also includes an escalation flow, troubleshooting info, groups links, dashboard and SLO, and on-call information.

API reference. This guide provides descriptions of functions, classes, and methods, typically with minimal narrative or reader guidance. This documentation is usually generated from code comments and sometimes written by tech writers.

Developer guide. Engineers use this guide to find out how to program to a product's APIs. Such guides are necessary when SREs create products that expose APIs to developers, enabling creation of composite tools that call each other's APIs to accomplish more complex tasks.

Playbooks contain instructions for verification, troubleshooting, and escalation for each alert generated from network-monitoring processes.

Documents for Reporting Service State

Here, we describe the documents that SRE teams produce to communicate the state of the services they support.

Quarterly service review. Information about the state of the service comes in two forms: A quarterly report reviewed by the SRE lead and shared with the SRE organization, and a presentation to the product development lead and team.

The goal of a quarterly report (and presentation) is to cover a “State of the Service” review, including details about performance, sustainability, risks, and overall production health.

SRE leads are interested in quarterly reports because they provide visibility into the following:

- *Burden of support (on-call, tickets, postmortems).* SRE leads know that when the burden of support exceeds 50% of the SRE team's resources, they must respond and change the priorities of their teams. The goal is to give early warning if this starts to trend in the wrong direction.

- *Performance of the SLA.* SRE leads typically want to know if the SLA is being missed or if the ecosystem has an unhealthy component that puts the product clients in jeopardy.

- *Risks.* SRE leads want to know what risks the SREs see to being able to deliver against the goals of the products and the business.

Quarterly reports also provide opportunities for the SRE team to:

- Highlight the benefit SRE is providing to the product development team, as well as the work of the SRE team.

- Request prioritization for resolving problems hindering the SRE team (sustainability).

- Request feedback on the SRE team's focus and priorities.

- Highlight broader contributions the team is making.

Production best practices review.

With this review SRE teams are better able to adopt production best practices and get to a very stable state where they spend little time on operations. SRE teams prepare for these reviews by providing details such as team website and charter, on-call health details, projects vs. interrupts, SLOs, and capacity planning.

The best practices review helps the SRE team calibrate itself against the rest of the SRE organization and

improve across key operational areas such as on-call health, projects vs. interrupts, SLOs, and capacity planning.

Documents for Running SRE Teams

SRE teams need to have a cohesive set of reliable, discoverable documentation to function effectively as a team.

Team site. Creating a team site is important because it provides a focal point for information and documents about the SRE team and its projects. At Google, for example, many SRE teams use g3doc (Google's internal doc platform, where documentation lives in source code alongside associated code), but some teams use a combination of Google Sites and g3doc, with the g3doc pages closely tied to the code/implementation details.

Team charter. SRE teams are expected to maintain a published charter that explains the rationale for the team and documents its current major engagements. A charter serves to establish the team identity, primary goals, and role relative to the rest of the organization.

A charter generally includes the following elements:

- ▶ A high-level explanation of the space in which the team operates. This includes the types of services the team engages with (and how), related systems, and examples.
- ▶ A short description of the top two or three services managed by the team. This section also highlights key technologies used and the challenges to running them, benefits of SRE engagement, and what SRE does.
- ▶ Key principles and values for the team.
- ▶ Links to the team site and docs.

Teams are also expected to publish a vision statement (an aspirational description of what the team would like to achieve in the long term) and a roadmap spanning multiple quarters.

Documents for New SRE Onboarding

SRE teams invest in training materials and processes for new SREs because training results in faster onboarding to the production environment. SRE teams also benefit from having new members acquire the skills required to join the ranks of on-call as early as possible. In the absence of comprehensive training, as seen in Zoë's story, the on-

SRE teams invest in training materials and processes for new SREs because training results in faster onboarding to the production environment.

call SRE can flounder during a crisis, turning a potentially minor incident into a major outage.

Many SRE teams use checklists for on-call training. An on-call checklist generally covers all the high-level areas team members should understand well, with subsections under each area. Examples of high-level areas include production concepts, front-end and back-end stack, automation and tools, and monitoring and logs. The checklist can also include instructions about preparing for on-call and tasks that need to be completed when on call.

SREs also use role-play training drills (referred to within Google as *Wheel of Misfortune*) as an educational tool for training team members. A Wheel of Misfortune exercise presents an outage scenario to the team, with a set of data and signals that the hypothetical on-call SRE will need to use as input to resolve the outage. Team members take turns playing the role of the on-call engineer in order to hone emergency mitigation and system-debugging skills. Wheel of Misfortune exercises should test the ability of individual SREs to know where to find the documentation most relevant to troubleshooting and resolving the outage at hand.

Repository management. SRE team information can be scattered across a number of sites, local team knowledge, and Google Drive folders, which can make it difficult to find correct and relevant information. As in the SRE example earlier, a critical operational tool and its user manual were unavailable to Zoë (the on-call SRE) because they were hidden under the home directory of her tech lead, and her inability to find them greatly prolonged a service outage. To eliminate this type of failure, it is important to define a consistent structure for all information and ensure all team members know where to store, find, and maintain information. A consistent structure will help team members find information quickly. New team members can ramp up more quickly, and on-call and on-duty engineers can resolve issues faster.

Here are some guidelines to create and manage a team documentation repository:

- ▶ Determine relevant stakeholders and conduct brief interviews to identify all needs;

- ▶ Locate as much documentation as possible and do a gap analysis on content;
- ▶ Set up a basic structure for the site so that new documentation can be created in the correct location;
- ▶ Port relevant existing documentation to a new location;
- ▶ Create a monitoring and reporting structure to track the progress of migration;
- ▶ Archive and tear down old documentation;
- ▶ Perform periodic checks to verify that consistency/quality is being maintained;
- ▶ Verify that commonly used search terms bring up the right documents near the top of the search results; and,
- ▶ Use signals such as Google Analytics to gauge usage.

A note on repository maintenance: it is important that docs are reviewed and updated on a regular basis. The owner's name should be visible, as well as the last reviewed date—all this information helps with the trustworthiness of the documentation. In Zoë's story she found and used an obsolete document for a critical operational tool and thereby missed an opportunity to resolve an incident quickly rather than experience a major outage. If documents cannot be trusted to be accurate and current, this can make SREs less effective, directly impacting the reliability of the services they manage.

Repository availability. SRE teams must ensure documentation is available even during an outage that makes the standard repository unavailable. At Google, SREs have personal copies of critical documentation. This copy is available on an encrypted compact storage device or similar detachable but secure physical media that all on-call SREs carry with them.

Documents for Service Decommissioning

Once services reach end of life, SREs decommission them in a predictable fashion. Here, we provide messaging and documentation guidelines for service deprecation leading to eventual decommissioning.

It is important to announce decommissioning to current service users well ahead of time and provide them with a timeline and sequence of steps. Your announcement should explain when

new users will no longer be accepted, how existing and newly found bugs will be handled, and when the service will completely stop functioning. Be clear about important dates and when you will be reducing SRE support for the service, and send interim announcements as the timeline progresses.

Sending an email message is not sufficient, and you must also update your main documentation pages, playbooks, and codelabs. Also, annotate header files if applicable. Capture the details of the announcement in a document (in addition to email), so it's easy to point users to the document. Keep the email as short as possible, while capturing the essential points. Provide additional details in the document, such as the business motivations for decommissioning the service, which tools your users can take advantage of when migrating to the replacement service, and what assistance is available during migration. You should also create a FAQ page for the project, growing the page over time as you field new questions from your users.

Technical writers provide a variety of services that make SREs effective and productive. These services extend well beyond writing individual documents based on requirements received from SRE teams.

Here is some guidance to technical writers on best practices for working with SRE teams.

- ▶ Technical writers should partner with SREs to provide operational documentation for running services and product documentation for SRE products and features.

- ▶ They can create and update doc repositories, restructure and reorganize repositories to align with user needs, and improve individual docs as part of the overall repository management effort.

- ▶ Writers should provide consulting to assess, assist, and address documentation and information management needs. This involves conducting doc assessments to gather requirements, enhancing docs and sites created by engineers, and advising teams on matters related to documentation creation, organization, redesign, findability, and maintenance.

- ▶ Writers should evaluate and improve documentation tools to provide the best solutions for SRE.

Templates. Tech writers also provide templates to make SRE documentation easier to create and use. Templates do the following:

- ▶ Make it easy for authors to create documentation by providing a clear structure so that engineers can populate it quickly with relevant information;
- ▶ Ensure documentation is complete by including sections for all required pieces of documentation; and,
- ▶ Make it easy for readers to understand the topic of the doc quickly, the type of information it's likely to contain, and how it's organized.

Site Reliability Engineering contains several examples of documentation templates. To view the templates, visit https://queue.acm.org/appendices/SRE_Templates.html

Conclusion

Whether you are an SRE, a manager of SREs, or a technical writer, you now understand the critical importance of documentation for a well-functioning SRE team. Good documentation enables SRE teams to scale up and take a principled approach to managing new and existing services. **C**

Related articles on queue.acm.org

The Calculus of Service Availability

Ben Treynor, Mike Dahlin, Vivek Rau, and Betsy Beyer
<https://queue.acm.org/detail.cfm?id=3096459>

Resilience Engineering: Learning to Embrace Failure

A discussion with Jesse Robbins, Kripa Krishnan, John Allspaw, and Tom Limoncelli
<https://queue.acm.org/detail.cfm?id=2371297>

Reliable Cron across the Planet

Štepan Davidovic and Kavita Guliani
<https://queue.acm.org/detail.cfm?id=2745840>

References

1. Blank-Edelman, D.N. *Seeking SRE: Conversations About Running Production Systems at Scale*. O'Reilly Media, 2018.
2. Murphy, N., Beyer, B., Jones, C., Petoff, J. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, 2016.

Shylaja Nukala is a technical writing lead for Google Site Reliability Engineering. She leads the documentation, information management, and select training efforts for SRE, Cloud, and Google engineers.

Vivek Rau is a Site Reliability Engineer at Google, working on Customer Reliability Engineering (CRE). The CRE team teaches customers core SRE principles, enabling them to build and operate highly reliable products on the Google Cloud Platform.

Copyright held by authors/owners.