Book Review

Implementing Service Level Objectives by Alex Hidalgo

LAURA NOLAN

Laura Nolan's background is in Site Reliability Engineering, software engineering, distributed systems, and computer science, with a career split roughly evenly between software engineering and SRE-like roles. She has contributed to a number of SRE books, including *Site Reliability Engineering, Seeking SRE*, and *97 Things Every SRE Should Know*. Outside of work, Laura is a part-time student of technology ethics at Dublin City University and is an active campaigner against autonomous weapons. Laura currently serves on the board of USENIX.
laura.nolan@gmail.com

In the past two years, Service Level Objectives (SLOs) have become almost synonymous with Site Reliability Engineering (SRE). SLOs are a reliability target—a threshold of availability and correctness that the users of a service should be satisfied with and that the service ought to be able to meet under normal circumstances.

Site Reliability Engineering [1], published in 2016, set out SLOs as a foundational topic: "It's impossible to manage a service correctly, let alone well, without understanding which behaviors really matter for that service and how to measure and evaluate those behaviors." The *Site Reliability Workbook* [2] upped the ante in 2018, saying, "[SREs'] day-to-day tasks and projects are driven by SLOs: ensuring that SLOs are defended in the short term and that they can be maintained in the medium to long term. One could even claim that without SLOs, there is no need for SREs."

SLOs appear to be simple—we just need to choose how many nines we want—and SLO adoption has often been held up as the first step on any organization's path towards adopting SRE practices. There is a school of thought that sees SRE as a cookie-cutter approach that can be generically applied to any service: just define your SLOs, configure your error-budget-based alerting, build a release pipeline with canarying and rollback, automate away the bulk of your repetitive work, adopt the Incident Management System, and do blameless postmortems and voila—your service will be reliable. Now, these are all worthwhile practices for sure, but are they enough? I believe not. They will get you part of the way there, and it's a good roadmap for productionizing a greenfield project. However, any sizable real-world system will have its own challenges, rough edges, and sharp corners. You need depth and a lot of context to run systems well, not just a cookie-cutter shallow SRE process. If you want to be an SRE for a database tier, you will need to learn a lot about databases in general and your database in particular to do it well. If you are SRE for Java-based services, you need to understand the JVM as well as your services' design, and so on.

Because context is so important, I personally believe that setting up a structured weekly production meeting with comprehensive notes and solid tracking of action items is actually a better first starting point than SLOs with a team new to SRE—you use it immediately to build shared context on services and identify burning fires and pain points that can be mitigated quickly. This shared context becomes a useful foundation for defining SLOs. But deep service expertise is not generic—it's qualitative, not quantitative, and it takes time to build. It's not as easy to write an article or a book chapter about it. You can't create a platform to sell Context-as-a-Service, there are no clever-sounding acronyms, and there are no graphs for executives. In short, it isn't going to help you sell anything or get you promoted (not directly, anyway).

As SREs go, therefore, I'm something of an SLO skeptic. However, even I concede that though SLOs may not be a silver bullet, they are nonetheless useful, and stable SRE teams ought to ensure that their services have appropriate SLOs. SLOs do have a lot of benefits: they can provide an explicit "contract" of sorts between services provided by different teams, helping create clarity about expected reliability and customer needs. SLOs can help you set alerting thresholds and feed into decision making about priorities (without being the sole input to that process).

Nines Are Not Enough

It's been encouraging to see the conversation around SLOs gain nuance and depth in the past year, compared to the fairly basic treatments in the original 2016 *Site Reliability Engineering* [1] and the 2018 *Site Reliability Workbook* [2]. Mogul and Wilkes' HotOS 2019 "Nines Are Not Enough" paper [3] is required reading for anyone interested in the topic—it includes an analysis of some of the real-world complexities and tradeoffs of providing SLOs, including the role of customer behavior; no system can defend an SLO when arbitrary behaviors are allowed. Narayan Desai's talk on SLOs, "The Map Is Not the Territory" [4], discusses a different set of difficulties, particularly around how the aggregation process can mask significant customer pain, especially for low-QPS services or unevenly distributed errors that affect some customers much more than others. Finally, August 2020 saw the release of an entire book [5] dedicated to SLOs: Alex Hidalgo's *Implementing Service Level Objectives*.

Hidalgo introduces the core concepts—SLOs, service level indicators (SLIs), and error budgets—in a similar way to the SRE book [1] and SRE workbook [2] but spends time considering SLOs for significantly more "shapes" of services: not only simple RPC or HTTP services, but also datastores, compute platforms, pipelines, and batch jobs. Data reliability gets an entire chapter, written by Polina Giralt and Blake Bisset, which proposes 13 properties of data, such as freshness, accuracy, and completeness, along with discussions on how to measure these. This chapter is particularly welcome: many of us are running either datastores or pipelines or both. The properties of data-intensive systems are both more complicated than those of the simple request-processing systems normally used to illustrate SLOs, as well as usually more difficult to measure.

Chapter 4, "Choosing Good SLOs," is the foundation for the whole book. Again, it covers a lot of the same ground as the SRE book and SRE Workbook, but with some valuable additions. There is a particularly good discussion of the organization and operational problem that arises from having too many SLOs—resist the temptation to think that every important metric should have an SLO associated with it. The discussion on SLO composition, meaning how to think about your services' reliability in relation to that of the services you depend on, is valuable and doesn't shy away from detail. Toby Burress and Jaime Woo's chapter on probability and statistics develops this further, alternating between theory of probability and statistics and concrete applications to difficult SLI calculations (such as infrequent batch jobs, requests that can be retried), and latency in queueing systems.

Burress and Woo's chapter and the "Architecting for Reliability" chapter by Salim Virji are very useful treatments of the math involved in building (or modifying) services to meet a desired SLO.

Hidalgo places a lot of emphasis on getting SLIs right, which is very worthwhile because this is often much more difficult in practice than the introductory examples from the SRE book suggests. There is significant material on the details of computing SLIs, including fairly well-known best practices such as use of percentiles rather than means and how to deal with time windows, as well as less well-known practical problems such as infrequent events and noisy or low-quality data. This is built on later by Ben Sigelman's chapter on measuring SLIs and SLOs, which discusses the tradeoffs involved in computing SLIs from time-series databases and structured event databases (or logs) and distributed traces. Sigelman's chapter usefully points out a number of traps for the unwary, such as relying on metrics reported by potentially malfunctioning services as opposed to other systems' view of those services.

Niall Murphy's chapter on SLO-based alerting rounds out the section of the book that is focused on the technical details of implementing SLOs. I particularly like that this section presents a progressive set of steps for improving your alerting in a brownfield situation. There is a valuable discussion of how to set up separate long-duration and short-duration alert thresholds to detect both major short term-problems and significant but slower-burning issues that are consuming your error budget at a higher than anticipated rate. This valuable alerting pattern is not used widely enough, but it is an excellent mechanism for catching serious but not immediately catastrophic problems without causing excessive pager noise.

The "Worked Example" chapter puts together a set of SLOs for several user-facing and internal systems with a variety of architectures and requirements. The author does a consistently good job of putting the end-user experience front and center here and relating it to the SLOs and SLIs proposed. However, most of the systems and SLOs proposed are fairly simple, and this chapter could do more to reinforce Giralt and Bisset's chapter on data systems, or Murphy's chapter on alerting.

The book closes with a series of less technical chapters on the theme of building an SLO culture, discussing topics like setting up SLOs in organizations new to the concept, how your SLOs may evolve as your service changes over time, how to make your SLOs discoverable to other teams, and how to advocate for SLOs. The final chapter (on SLO reporting) contains a fairly lengthy polemic on why SLO reporting is superior to reporting based on Mean Time To Recovery (and similar metrics)—Hidalgo is right to say that these kinds of measurements are subjective and not generally meaningful because incidents are so different from each other.

Any engineer who works day-to-day with reliability, metrics, monitoring, and alerting ought to have a copy of this book. Even those who don't necessarily want to see how deep the SLO culture-change rabbit hole goes will gain much from the technical chapters, which can inform your monitoring and alerting strategies and even the tradeoffs made in your system architecture.

References

[1] "Service Level Objectives," Chapter 4 in B. Beyer, J. Petoff, N. R. Murphy, and C. Jones, eds., *Site Reliability Engineering: How Google Runs Production Systems* (O'Reilly Media, 2016).

[2] "Implementing SLOs," Chapter 2 in B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, and S. Thorne, eds., *Site Reliability Workbook: Practical Ways to Implement SRE* (O'Reilly Media, 2018).

[3] J. C. Mogul and J. Wilkes, "Nines Are Not Enough: Meaningful Metrics for Clouds," in *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS '19),* pp. 136–141: https://dl.acm.org/doi/pdf/10.1145/3317550.3321432.

[4] N. Desai, "The Map Is Not the Territory," at SRECon19 EMEA 2019: https://www.usenix.org/conference/srecon19emea/presentation/desai.

[5] A. Hidalgo, *Implementing Service Level Objectives* (O'Reilly Media, 2020).