Decision-Making Using Service Level Objectives

ALEX HIDALGO

Alex Hidalgo is a Site Reliability Engineer and author of the forthcoming *Implementing Service Level Objectives* (O'Reilly Media, September 2020). During his career he has developed a deep love for sustainable operations, proper observability, and use of SLO data to drive discussions and make decisions. Alex's previous jobs have included IT support, network security, restaurant work, t-shirt design, and hosting game shows at bars. When not sharing his passion for technology with others, you can find him scuba diving or watching college basketball. He lives in Brooklyn with his partner Jen and a rescue dog named Taco. Alex has a BA in philosophy from Virginia Commonwealth University. sometimesitsalex@gmail.com @ahidalgosre

Service level objectives, or SLOs, are quickly becoming the latest industry buzzword. Engineers want them, leadership demands them, and job postings increasingly ask for experience with them. However, SLOs are meaningless unless they are understood as more than just the latest industry jargon. There are true, real-world benefits to adopting an SLO-based approach to reliability. I will explain why they are important and how you can use them most effectively to have discussions that lead to better decisions.

Using service level objectives to measure the reliability of services is getting more attention than ever before. This is partly due to the success of the first two Google-authored site reliability engineering (SRE) books. But it also seems that many people actually resonate with the approach and find it an intuitive concept to follow. While it is possible that many organizations are forcing their teams to adopt SLOs via mandate just to ensure they're on board with the latest buzzwords, it also seems likely that many people are finding true value in the approach.

I found only one study tracking the adoption rates of SLO-based approaches in this book: https://www.oreilly.com/library/view/slo-adoption-and/9781492075370/. Instead, I'll have to rely on the general anecdotal evidence I have in terms of how many companies are rolling out products to help people measure SLOs, how many conference talks are focused on them, and how often I personally find myself engaged with people who want to learn more about the process.

While SLO-based approaches to reliability are certainly useful to many people, I also cannot ignore the fact that the very phrase has become a buzzword that is starting to lose meaning. It's not uncommon for words, phrases, and concepts that gain traction and desirability to have their original meanings forgotten. Service level objectives are no different. They provide many benefits, but some of their most important aims have unfortunately become obfuscated by more readily available ones.

SLO-based approaches to reliability give you many benefits, and there are many reasons why organizations may choose to adopt them. Unfortunately, for many they have just become "a thing you do." This is not to say that every organization looking to adopt such an approach has overlooked the benefits of SLOs, but few manage to use them to their full potential.

Let's explore some of the ways you can use the information that service level objectives provide to make better decisions through data. Making better decisions is at the very heart of the SLO approach, and that's often the part that is overlooked.

But first, let's outline how this approach works in a little more detail.

SLO Components

There are three primary components to an SLO-based approach. The first is *service level indicators*, or SLIs. A good SLI is a measurement that tells you how your service is performing from the perspective of your users. In this case, when I say users, they could be anything from paying customers to coworkers to other services that depend on yours—there doesn't strictly have to be a human attached to the other end. In this article, I'll mostly be talking about human users who interact with web services since they are intuitively accessible concepts

that almost all of us interact with on a daily basis; however, the concepts and approaches apply to any service and any type of user, even if those users are just other computer systems.

After SLIs, you have *service level objectives*, which are targets for how you want your SLI to perform. While a service level indicator may tell you how quickly a web page loads, an SLO allows you to do things like set thresholds and target percentages. An example SLI might be "Web pages are fully rendered in the user's browser within 2500 ms." Building off of that, an SLO might read, "The 95th percentile of web page render times will complete within 2500 ms 99.9% of the time." Service level objectives allow you to set reasonable targets. Nothing is ever perfect, and 100% is impossible for just about everything, but by using SLOs, you can ensure that you're striving for a reasonable target and not an unreachable one.

Finally, you have *error budgets*. An error budget is a way of keeping track of how an SLO has performed over time. If you acknowledge that only 99.9% of the 95th percentile of your web page render times have to complete within 2500 ms, you are also acknowledging that 0.1% of them don't have to. Error budgets give you a way to do the math necessary to determine whether your adherence to your SLO target is suitable for your users, not just in the moment but over the last day, week, month, quarter, or year.

SLIs, SLOs, and error budgets are all data—data that allows you to ask important questions that can drive better decision-making.

- Is our SLI adequately measuring what our users need and expect? If not, we need to figure out a new way to measure this.
- Is our SLO target meaningfully capturing the failure rates our users can tolerate? If not, we need to pick a new target or new thresholds.
- What is our error budget status telling us about how our users have actually experienced our service over time? If we've exceeded the error budget, perhaps we drop feature work and focus on reliability instead.

Decisions about User Experience

A meaningful SLI is one that captures the user experience as closely as possible. Following our simple example from above, it is pretty intuitive to think about the fact that the users of a web service need their pages to load—and to load in an amount of time that won't annoy them. But there is so much more to the user experience than just the concepts of availability and latency. A web service is not doing its job just by being able to render pages in a timely manner. If you're only measuring things like availability and latency, the only data SLO-based approaches can provide you are ones that focus on improving your availability and latency.

A web service is often much more than just serving data to people. Imagine that your web service is a retail site. In such a case, you suddenly have many other user journeys to consider. If you want people to be able to purchase items from you, they need to be able to do exactly that and not just have web pages display in their browser.

For example, a standard retail website often has some sort of *shopping cart* feature—one where a user can add a potential purchase to a list of items they might want to check out with later. This shopping cart feature has to do a lot of things in order to be reliable.

The first is that it needs to do what it is supposed to: if a user wants to add an item to their shopping cart, they should be able to do exactly that. Additionally, it needs to be persistent; a shopping cart isn't much good if it only remains consistent with the wishes of a user for a short amount of time.

It also has to be accurate. There is no sense in allowing customers to add to a list of items they might want to purchase if that list doesn't represent the items they have actually chosen.

Finally, how the user interacts with the shopping cart has to work properly. If an item is added or removed, it should actually be added or removed. If the user expects an icon representing how many items they have in their cart to be updated when they add a new item, that icon should actually update in real-time.

These examples all represent different data points that meaningful SLIs can give you—and these data points help you make decisions. If it's simply the case that your site isn't loading well or quickly enough, you might just need to introduce more resources. However, if the shopping cart isn't working well, the problem could be anything from the JavaScript powering user interactions to the service that talks to the database to the data-storage systems that are ultimately responsible for holding the ones and zeroes. By having the data that multiple meaningful SLIs provide you, you can make better decisions about what you should be measuring in the first place and what areas of your system require the most attention.

Decisions about Tolerable Failures

One of the most attractive aspects of measuring services with SLOs is that the entire discipline acknowledges the fact that nothing is ever perfect. All complex systems fail at some point in time, and because of this fact it is fruitless to aim for 100%. Additionally, it turns out that people already know and are okay with this—whether they're consciously aware of it or not.

For example, if you start streaming a video via a video-streaming service, you have a certain expectation for how long it should take for such a video to buffer before it begins playing in real time. However, if it takes much longer than normal to buffer every once in a while, you likely won't care too much. Most people won't abandon a streaming video platform if one in every 100 videos

takes 10 seconds of buffering time instead of three seconds. Failure in the sense that the streaming platform had to buffer too long occasionally is just fine—it just can't happen too often. If videos take 10 seconds to start every single time, people might become annoyed and look for other options.

A good service level objective lies somewhere just beyond what you need for users of your service to be happy. If people are okay with one in every 100 streaming attempts buffering for longer than normal, you should set your SLO target at something like one in every 200 streaming attempts. Exactly where you set this target is up to the data you have available to you and the feedback you're able to collect from your users. The important part is that your SLO should be more strict than the level at which users might decide to leave and use a different option. No matter how refined your SLO target is, you're not always going to reach it, and you don't want your business or organization to suffer if you don't.

Acknowledging failure and accounting for it are at the very base of how SLO-based approaches work. Understand that nothing is perfect, but use SLO data to help you decide how close to perfect you should attempt to be—or risk losing users.

Decisions about Work Focus

Once you have a meaningful SLI and a reasonable SLO target, you can produce an error budget. Error budgets are simply just another data point you can use to make decisions. They're the most complicated part of the stack, but once you can find yourself using error budgets to drive your decision-making, you'll truly understand how the entire SLO-based approach works.

Error budgets are the ultimate decision-making tool once you've established SLIs and SLOs. By measuring how you've performed over a time window, you can drive large-scale decisions that could impact anything from the focus of your team for a single sprint to the focus of an entire company for a quarter.

For example, let's say you have a reasonable measurement of how reliable one particular microservice has been. Using your error budget, you can now also see that you haven't been reliable about 10% of the time over the last month. At this point you can use this data to inform a few different discussions that can fuel decisions.

One example is that you simply haven't been performing well enough, and that you believe that your SLI measurement and your SLO target are well-defined. If this is the case, you might choose to pivot one or more members of your team to focusing on reliability work instead of feature work. You could do this for anything, like the length of an on-call shift to a full sprint or even until you've recovered all of your budget. There are no hard-and-fast rules at play here. Error budgets, like everything else, are just data to help you decide what to do.

Another example is that you've completely depleted your error budget but have reason to think this exact situation is unlikely to arise again. An example of this kind of event could be anything like the disruption of power at a datacenter or just a historically bad bug pushed to production. Just because you've depleted your error budget over time doesn't mean you have to take action. Sometimes it absolutely makes sense to do so: perhaps you need to introduce a better testing infrastructure to your deployment pipeline, or maybe you need to install additional circuits or distribute your footprint geographically to avoid further power disruptions.

The point is that it's totally okay to look at how you've performed in terms of reliability over time and say, "This time we can just continue our current work focus." Error-budget statuses are just another data set you should use to make decisions—they shouldn't be rules that need to be followed every single time you examine your status. It doesn't matter if you're looking at the error budget status for a single small microservice that sees very little traffic or your entire service as viewed from your paying customers. Use error budgets as data to help you think about prioritization.

For a larger service, such as an entire customer-facing web service, burning through all of your error budget probably warrants some stricter decision-making. Even if it was due to your ISP that your users experienced an hour of outage last month, it still probably doesn't make sense for you to do things like perform potentially disruptive chaos engineering or experimentation in your production environment until a significant amount of time has passed. Be reasonable about how you make decisions using your error budgets, and certainly feel free to ignore their status from time to time—but never do so at the expense of your users' experience.

Conclusion

There are many benefits to SLO-based approaches that I don't have room to cover here. They can help you better communicate to other teams about how they should think about the reliability of their own services. They can be excellent tools in reporting to management and product teams. They can also be used for many things outside of computer services, such as examining whether your team's ticket load is too high or whether people aren't taking enough vacation time. An SLO-based approach is simply about thinking about people and users first, acknowledging nothing is perfect, and using some math to help you aim for reasonable targets instead.

But one of the most important parts of this approach is that it allows you to make better data-driven decisions. Don't just implement SLOs because they're popular and a buzzword, or because you heard a conference talk about them, or because upper-management has decided that every team must have one.

Implement SLOs because they give you data you can use to have better discussions and make better decisions—decisions that can help make both your team and your users happier.