

BLAMELESS

BRIDGING THE GAP: DEVOPS TO SRE



Welcome to Bridging the Gap: DevOps to SRE

DevOps and Site Reliability Engineering (SRE) are both practices dedicated to improving the reliability and frequency of software releases. SRE was implemented internally at Google in 2003 but didn't gain widespread attention until 2016 with the release of Google's **SRE Handbook**. Before, most orgs focused on DevOps. **Liz Fong-Jones** and **Seth Vargo**, prominent leaders in the space, **have made it clear** that the practices aren't mutually exclusive. Organizations now have the opportunity to marry the two in order to strengthen their systems, practices, and service reliability.

As software apps and services become a more ubiquitous part of daily life, reliability becomes more important. It's really your most necessary feature — does it really matter how cutting-edge your service is if users can't access it? Implementing SRE is your best path to understanding and improving reliability. Its benefits don't end there, though. SRE is also about boosting development velocity by informing strategic risks. The **business value** of SRE shouldn't be missed!

This eBook will guide you through implementing the principles of SRE within your organization. We'll break down the obstacles you might encounter, especially as you begin to make sense of how SRE works together with DevOps. You'll also learn how the tools you already have in place give you a head start. By the end, we'll have established three solid foundations of SRE: incident response, service level objectives (SLOs), and team culture. Here we go!

Table of Contents

1 Life with SRE

2 Incident Management

How to elevate your incident management with SRE
Your incident response toolbox

3 SLOs

What SLOs can do for you
What are SLOs
This sounds pretty tough
The road to mastering SLOs

4 Culture

What culture can do for you
Be blameless
Be holistic
Put reliability first
Embrace risk

5 Plot Your Maturity

6 Summary

Life with SRE

If you're reading this eBook, it's safe to assume that your organization is set up with DevOps. You've done the work to break down silos between teams and built a strong [lifecycle](#) that makes deploys faster. Perhaps everything is going well, but you're curious if anything could be better. Site Reliability Engineering (SRE) has gained momentum among leading engineering orgs. What exactly is this new approach? And how is it different from DevOps?

Let's put it this way. While the goal of DevOps is to create alignment between developers and operators, the goal of SRE is to improve the end user experience. This plays out in how SRE measures progress and implements process. For example, SRE formalizes the idea that not every incident requires immediate attention. In fact, the concept of a service level objective (SLO) denotes a threshold for detecting whether incidents have become a potential threat to user satisfaction. If a system is not disrupting the threshold, then it's better to invest time progressing new initiatives for the next release.

To help you picture what this concept looks like in practice, imagine a dashboard lined with performance graphs. You might have a graph for availability, another on latency, etc. For the SRE, the most important insights are ones that measure [reliability](#). Reliability weighs monitoring metrics based on how much they matter to users. Site reliability engineers care about telemetry that explains how the user is experiencing the production service at that moment in time.

SRE's take a data-driven approach to incident management that puts development and operations teams on the same page, agreeing on shared goals. Teams feel more confident knowing how their service performance affects the end user and how to improve over time. Before we dive into the good stuff, be encouraged in knowing that if you already practice DevOps, you're well-equipped to start implementing SRE practices today. The goal of this eBook is to walk you through simple first steps that you can implement immediately. Let's get started!

“

If you think about DevOps as a philosophy, SRE is a prescriptive way of accomplishing that philosophy.

”

Liz Fong-Jones,
Principal Developer Advocate
for SRE at Honeycomb.io,
previously Google

Incident Management

How to elevate your incident management with SRE

Let's get a clear look at how levelling up incident management will change your organization. If you are functioning as a DevOps team today, you are certainly responding to and managing incidents as they occur but it's likely not a standardized approach across all teams with an end-to-end playbook used to streamline who responds. Closing out an incident with clear follow-up actions and learnings carried forward is critical and, unfortunately, often missed.

Non-standardized incident management	Incident management with SRE
Incident response is typically executed "on-the-fly" without established processes.	A library of runbooks gives you a head start whenever something goes wrong.
Tasks are likely repeated due to role ambiguity, which causes frustration.	Role-based checklists and shared communication channels keep everyone on the same page.
Important steps get skipped, losing the opportunity to collect data and make improvements.	Incident retrospectives with followup items carry the lessons of each incident forward.
On-call teams suffer alert-fatigue and burn out.	Balanced schedules and focused alerts keep on-call teams at their best.

Your incident response toolbox

Incident response isn't a singular action or step. It's a collection of tools you have at the ready for each time an incident occurs. Three important tools that are used by both DevOps and SRE teams are:

- Runbooks
- Classifications
- Retrospectives

Let's examine the incident response toolbox, remind ourselves why they are helpful, and introduce SRE's approach to leveraging them.



Runbooks

Purpose

Runbooks guide engineers through incident response. They're a series of steps and checks curated for different types of incidents.

How it's used in SRE

- Keep track of the steps you take each time you resolve an incident.
- Break down the details and explain the desired outcome.
- Prioritize runbooks for your most common incident types.

Challenges

- It's easy to forget note-taking during high-pressure situations. Keep focused!
- Following a structured guide can feel forced. Stay flexible and follow intuition!
- Remember to [update runbooks](#) as time continues. Improvement is important.

How to bridge the gap

- Review any data you have from previous incidents and spot where there are process redundancies. Edit those areas and leverage [automation](#) if appropriate.
- Ask the team to share their routines; codify their mental runbooks.
- Program runbooks into your systems so they report steps to responders.



QUICK TIP

Runbooks, retrospectives, and SLOs function best when everyone contributes. How do you encourage people to participate? Make it more appealing:

- It provides a safe space to share ideas and opinions.
- If everyone is there, you can all get on the same page.
- Prepping the team for adaptation to uncertainty is best done in collaboration.

Classification and role-based responses

Purpose

Categorize incidents according to significant features that set them apart. Assign specific roles and responsibilities based on the type of incident.

How it's used in SRE

- Use simple methods to [categorize incidents](#) like distinguishing between a slowdown vs. an outage.
- Determine who should be alerted at each stage and assign specific roles.
- Automate role assignments using tools.

Challenges

- It can be tricky to define categories, like deciding what's Sev0 vs. Sev1.
- Teams will take time to learn new role assignments.
- People naturally resist change, even when the current setup isn't working. Get team members involved in deciding the new process so they have buy-in.

How to bridge the gap

- Evaluate your current on-call setup and collect feedback from your engineers.
- Identify a few of the most impactful incidents your team has experienced. Use these examples to set a standard for the highest incident severity level.
- Consider all the [steps of incident management](#) and key role players.



QUICK TIP

Configure your alerting system to auto-assign roles based on an incident's classification and trigger pre-assigned task checklists. Pre-assigning roles and tasks can:

- Eliminate redundant work.
- Ensure everything that needs to be done gets done.
- Reduce time wasted during an incident.

Incident Retrospective

Purpose

Summarize the incident and jot down what can be learned. Suggest improvements in process, tools, and practices in order to manage incidents better in the future.

How it's used in SRE

- Create a retrospective for every incident. Yes, every single one. Fall back on the [basic elements of a retrospective](#) so that key information is retained.
- Make sure to include feedback from all relevant stakeholders.
- Always identify key learnings and areas for improvement.
- Post-incident meetings focus on systems thinking to support a blameless culture.

Challenges

- It will take time for people to understand the value of retrospectives.
- Finding solutions and areas for improvement can require some creativity.

How to bridge the gap

- Write down any immediate improvements you know need to be implemented.
- Add data from monitoring and observability tools into the retrospective.
- Leverage existing communication channels between dev, ops, and other teams to study and contribute to retrospectives together.
- Use a tool to automate the process of creating retrospectives.

Coming from the world of DevOps, you're well prepared to adopt these techniques. Your teams have already bought into the goal of improving speed and reliability. Ask them, *"Are we prepared to adapt if something goes wrong here that we've missed?"* Getting prepared to respond well to incidents is an investment in speed and reliability that you can't afford not to make.

SLOs

What SLOs can do for you

SLOs can revolutionize how you understand your production services. The purpose of setting service level objectives (SLOs) is to define what “success” means from the user’s perspective. Engineering teams ask, *“What’s the most important part of our service offering to this customer group, and what should we strive to provide consistently each month?”* By narrowing in on the most important metrics, you prioritize team workload, avoiding swirl and, ultimately, burnout. Let’s examine how life is different before vs. after they’re implemented.

Before SLOs	After SLOs
So many metrics are collected with monitoring tools without clear prioritization.	Fewer and focused metrics are reported to actually reflect user satisfaction.
Teams overreact and underreact to incidents.	Teams understand the business impact of incidents and respond accordingly.
Engineers are afraid to take risks in development.	Engineers accelerate new development work when error budgets are above SLO thresholds.
Crisis situations make it difficult to coordinate code freezes or new release timings.	Established SLO policies align to an agreed error budget and allow for fast responses.

What are SLOs

Before explaining service level objectives, it’s easier to take a step back and first define SLIs - Service Level Indicators. These are metrics that measure system health, but they don’t simply report on factors like uptime or latency. Instead, they’re built on top of those metrics to represent how users interact with your service.

Create an SLI that represents how users make purchases on your site, for example. The SLI might combine data such as:

- How long it takes for the selected product to be added to the cart
- How data refresh performs for the current stock of each product
- What percentage of checkout attempts complete without error

And so on. More specifically, here's what the SLI for this example could look like (in very simplified, very pseudo code):

$$\begin{aligned} & (\text{ProdAdd} < 5\text{ms} / \text{ProdAdd}) * 4 + \\ & (\text{StockUpdate} < 30\text{s} / \text{StockUpdate}) * 2 + \\ & (\text{CheckOut} = \text{success} / \text{CheckOut}) * 8 \end{aligned}$$

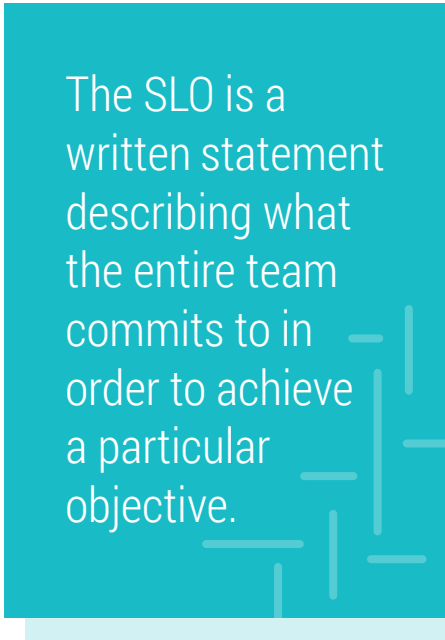
This represents:

- Ratio of times the product was added to the cart within 5ms vs all times products were added, multiplied by 4,
- plus ratio of times product data was updated less than 30s ago vs all product stock data multiplied by 2,
- plus ratio of successful checkout attempts vs all checkout attempts multiplied by 8.

This essentially tells the story of what the user cares about most. In this example, users value how quickly an item gets added to their cart more so than the accuracy of stock listings, so it's weighed twice as heavily. They also care even more about a successful checkout, so that's weighed twice as much again.

These are the metrics that give shape and meaning to service reliability. How we set goals for reliability is where we begin the discussion of agreeing on stated SLOs. The SLO is a written statement describing what the entire team commits to in order to achieve that particular objective. The SLA (which we won't dive into here) is an agreement between you, the vendor, and the customer in terms of what is an acceptable level of service.

Enough theory, what's an example? An SLO for a B2B project management tool provider, let's say, could be 99.99% of requests to see a Kanban board displaying an up-to-date version within 5ms over the year. In this situation, an example SLA could be that 99.00% of requests meet this standard over the year. The SLO provides a buffer to ensure that the SLA isn't breached.



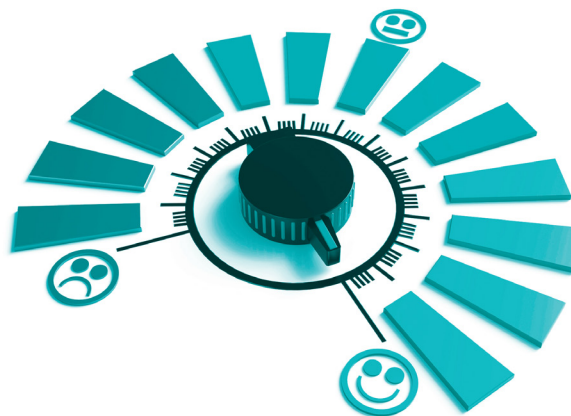
The SLO is a written statement describing what the entire team commits to in order to achieve a particular objective.

SLOs are both your safety net and your gas pedal.

Hopefully now you have a basic understanding of SLIs and SLOs. SLIs should be created based on metrics that indicate a reliable user experience. How do SREs pick SLO values? SLOs draw a line on what is acceptable or a threshold to stay at or above, consistently each month. It's the point of reliability that, if crossed, causes the end user to notice some type of system failure and likely translates to a negative experience. It's the acceptable level of reliability that you should always try to stay above.

Of course, the best way to make sure you don't fall below an SLO threshold is to aim beyond it. Another way to achieve this, especially when starting out, is don't aim for the moon or set the bar so high that you will likely fall below. Don't set it for 99.99% if you agree that 99.8% is acceptable. 99.99% uptime translates into about 5 minutes of unavailability per month, whereas 99.8% is about an hour and a half. Is an hour and a half spread over a month acceptable to users? Is it worth reducing further than that?

SLOs can also accelerate you. The error budget is a way to tell the team how they are doing as the month progresses. With charts, you can clearly see any degradation on that stated SLO. Therefore, it tells you how fast you can work in development by having the foresight to plan ahead. For example, you can trigger an alert if the system hits 80% of the error budget, telling you to prioritize fixes. SLOs are both your safety net and your gas pedal.



This sounds pretty tough

It can be. To guide you through, here's a list of the considerations you'll want to plan for as you begin tackling SLOs.

- ✓ **Getting the right metrics and event data you need.** You'll likely need to configure your monitoring/APM or Observability tools in order to feed all the necessary data into your SLI.
- ✓ **Figuring out user journeys.** Identify which metrics reflect what's important to users as you pick SLIs. This can be a project itself; you'll need to monitor, profile, and maybe survey. It'll likely be a collaborative effort between Engineering, Product, Customer Success, and others.
- ✓ **Agreeing on policies.** You'll want to align with Engineering about when it's right to reprioritize. Engineers want services to perform reliably for the end user too, so agreeing on an error budget from the get-go will hopefully mean smoother sailing through a set error budget.
- ✓ **Selling everyone on the time investment.** Building SLOs will take some time, and it will require revisions as you go on. Hopefully you can get everyone to agree that time spent planning equals time (and costs) saved in the long-run.
- ✓ **Reviewing and revising.** SLOs aren't just "set it and forget it". They'll always need to be adjusted as circumstances change. Start somewhere and iterate. By working on the SLO, you're essentially defining success and therefore helping to prioritize all future work.

The Holistic Side of SRE

Being familiar with DevOps, you see the value in cross-team collaboration. Those relationships empower SRE too! SLOs should take into account learnings from all stakeholders. When appropriate, invite other teams to collaborate on retrospectives and runbook updates. Promote the reliability mindset across the entire org!



The road to mastering SLOs

We just went over a long list of action items. Don't fret. On this page, we separated out what basic vs. advanced SLOs look like. That way, it's easy if you're just getting started. Even a basic SLO setup can supercharge your reliability and development velocity. Here's a guide to getting started.



Basic

- 1 Set up the data you want to monitor.** You likely have this in place already. Make sure your SLO tool receives all of the necessary data from your Observability and APM tools.
- 2 Build SLIs based on fundamental metrics.** The four golden signals: latency, errors, traffic, and saturation capture a lot of what you need to know about your service.
- 3 Set policies.** Decide what steps should be taken when the service falls short of an SLO. You can also set policies on how to approach process improvements over time.
- 4 Initiate review cycles.** Set a cadence for reviewing your SLOs. Periodically check to identify anything that is causing user pain. Maybe the SLOs are too stringent. Striking the right balance takes a bit of time at first, but finding the right SLO will save even more time in the long run.

Advanced

- 1 Build user journeys.** Research how customers use your service. Outline important details for the average use case. Allow this information to inform your SLIs (and respective SLOs).
- 2 Weigh steps based on impact.** Study which features are used more commonly and how critical they are to the overall user experience. Weigh them into your SLIs.

Take your time with these, and learn as you go. Every organization should have a unique set of SLOs and processes because teams vary, as do products and services.

Culture

What culture can do for you

While setting up a strong process is fundamental to implementing SRE, the culture built around SRE will make even more profound reliability improvements. If teams are resilient, they can better withstand any unexpected events that come at them, especially if it's 3am and they're on-call. An important source of resilience comes from a place of trust and clear expectation setting. Ultimately all teams strive for the same end-goal with shared outcomes. How you get there is the question. Let's examine what life looks like before vs. after teams adopt an SRE culture.

Before SRE	After SRE
Time is wasted pointing blame.	Work starts on resolving incidents right away.
People are blamed for incidents, and real causes are not addressed.	Systemic causes are identified and addressed, leading to enduring solutions.
Teams are paralyzed and risk-averse, wanting to avoid blame.	Teams feel psychologically safe to experiment and make mistakes, leading to greater agency and velocity.
Failure disappoints and creates resentment.	Failure is embraced as an opportunity to learn and grow.
Learning is siloed, causing redundant work and friction.	Teams share lessons freely and people are brought in to develop their understanding.
Reliability is an afterthought in development, causing a domino effect in operations.	Reliability is treated as critical from the beginning of all development work.

SRE culture can't be built overnight. It takes time to pull all teams in the same direction, especially if you're a large, distributed org. Culture takes shape as your org continues to follow established SRE procedures. They also take shape as your org cycles through the flow of tracking reliability insights, managing incidents, sharing retrospectives, and iterating on SLOs. SRE is equal parts process, systems, tools, and people, so getting everything and everyone aligned is no easy feat. If management supports the effort from top-down, it can speed things up. Let's look at how SRE practices support a resilient culture.

Be blameless

This one is obviously a big deal to Blameless, the company. When incidents occur — and they will — instead of trying to find a scapegoat, employ systems thinking and investigate potential contributing factors. Work together as a team to examine the system and its processes. There is always something to be learned and an area of the system to be improved.

Plant the seed. Do what you can to discourage others from assigning blame. This includes individuals blaming themselves for an issue. One way to avoid ruffling feathers as you start to build culture is to suggest that evaluating the system as a whole is more effective in helping the organization handle incidents even better in the future.

Watch it grow. Thorough retrospectives that offer suggestions for improving the system, processes, and tools in place help effect positive change. Once people see how helpful it is to identify underlying issues and roadblocks, pointing fingers begins to look like an excuse.

Harvest fruits. By establishing a blameless culture, people will start to feel confident that they won't be sanctioned for mistakes. It's important for teams to feel psychologically safe enough to raise issues, take risks, and be empowered in their choices.

Be holistic

Throughout the SRE lifecycle, it's helpful to lean on skills, ideas, and information from multiple sources. For example, during a post-incident review meeting, mature SRE teams invite a diverse group of stakeholders to contribute. Doing this, you'll come across great insights and even make interesting discoveries. When looking to improve your system, the possibilities should be kept wide open.

Plant the seed. For the next post-incident review meeting, invite stakeholders outside the usual group. Even when revising SLOs and adjusting SLIs, collaborate with teams like Product, Customer Success, and others.

Watch it grow. As insights from other teams feed into SLOs, the software development lifecycle (SDLC) becomes contextualized in more ways. This leads to more feedback and growth!

Harvest fruits. Working together across the organization promotes efficiency because everyone is working toward shared goals. People understand the responsibilities and challenges of those on other teams and start to identify where collaboration and support is needed.



Put reliability first

Reliability should be your number one feature. After all, it doesn't matter how new or cutting edge your service is if no one can use it! Reliability ought to be top of mind throughout the SDLC.

Plant the seed. Start gathering reliability insights for your services. Identify what's going well right now vs. what needs improvement. Draft requirements for what reliable code should look like.

Watch it grow. Investing in reliability yields greater and greater payoff as time goes on. Eventually, you'll see your org shift from a place of reactive incident management to proactive SRE. The results will speak for themselves.

Harvest fruits. Shifting from reactive to proactive, SRE teams might even begin to build runbooks during the development stage. Hope for the best, but prepare for the worst. When reliability becomes a shared responsibility, every stage in the SDLC works toward it.

Embrace risk

Incidents are bound to happen, even to the most advanced and well-established orgs. Pushing for 100% availability isn't plausible. Besides, customers likely can't tell the difference between 99.99% and 100% of requests returned at the expected speed. Why spend time, energy, and money where it won't be noticed? Instead, build out an error budget that encourages development teams to take risks.

Plant the seed. Establish SLOs to denote how reliable your service should be at a minimum. Build a plan for how you can accelerate development when it's safe to do so.

Watch it grow. The team will understand how risk can be measured against an error budget, and they won't be sanctioned (or blamed) for mistakes. Engineers will feel empowered to innovate.

Harvest fruits. Suddenly, taking risks doesn't exactly feel like taking risks, because you understand how updates impact system reliability. Engineers can make calculated decisions about the potential payoffs of new development.

Plot Your Maturity

SRE is both something you can see real benefits from on day one, and something that you'll grow into for years to come. Let's look at that journey in one picture.

	Starter SRE	Advanced SRE	Expert SRE
Team	Engineers take on SRE functions as they're able to.	Dedicated SREs focus entirely on SRE practices.	An established SRE team builds new practices together.
SLOs	Basic SLIs represent fundamental metrics tracking.	SLOs are built around specific user journeys on specific parts of the service.	Complex SLOs reflect user happiness service-wide.
Error budgets	Code freezes occur when SLOs exceed thresholds.	Development accelerates when the error budget is right on target.	Review cycles continually adjust SLOs and respective error budgets.
Runbooks	Teams record a collection of steps to use for common incidents.	Teams use documented processes that help tackle more complex issues.	Teams leverage automated runbooks that work through steps quickly and reliably.
Retrospectives	Teams do their best to document what they can as they resolve incidents.	Teams set up a system of writing retrospectives that include learnings.	Teams use a tool to automate the process of building retros and making improvements.
Culture	A blameless culture is achieved when resolving incidents.	People feel safe to make mistakes and take risks.	Failure is celebrated as an opportunity to grow, and everyone learns.

Summary



Following the steps in this eBook should help you feel comfortable talking about SRE and how it can benefit your org. If your team already follows the DevOps approach, then all it takes to get started is to repurpose existing systems, processes, and tools to give laser focus on reliability. Systems are never perfect, and neither are humans that build them. There's room for patience and grace, but there's also room for improvement, *always*. Remember to place end users at the forefront, make data-driven decisions, and promote systemic growth. Good luck!

Blameless drives reliability across the entire software lifecycle by operationalizing Site Reliability Engineering (SRE) practices. Teams share a unified context during incident response, efficiently communicate, and resolve faster. Detailed Retrospectives give teams a data-driven approach to learn and Service Level Objectives (SLOs) inform teams where to prioritize work and innovate at velocity. Customers include brands such as Procore, Under Armour, Citrix, Mercari, Fox, and Home Depot who embrace a blameless culture, team resilience, and greater reliability to protect their customers.

Blameless is a 2021 Gartner Cool Vendor recipient and is backed by Accel, Lightspeed, Decibel and Third Point Ventures. More info: www.blameless.com or [LinkedIn](#), [Twitter](#).

SRE and DevOps terminology can be confusing. Check out this [glossary](#) for clear definitions.