SRE Philosophy

What is SRE, anyway? Jennifer Mace (Macey) gives us her definition of Site Reliability Engineer, discusses how to manage risk, and shares key questions to ask developers.

MP: Hello, and welcome to the first ever episode of the Google SRE Podcast, or as we affectionately refer to it, "The Prodcast." I'm your host today, MP, and here with me as co-host is Viv.

Viv: Hi, Viv here. Happy to be here. Thanks, MP.

MP: This series that we'll be releasing over the next few months has grown out of a long-time internal podcast series made by SREs for other engineers here at Google. And we decided we wanted to take that idea and create something new for the entire SRE and Dev communities. Over the course of these episodes, we'll be taking SRE concepts, many of which are covered in the 2016 SRE book, and chatting with domain experts here at Google to hear their opinions on these topics and get some new perspectives. And to start us off here with us today to talk about "What is SRE, anyway?" is Macey. Why don't you go ahead and introduce yourself?

Macey: Hi. So my name is Jennifer Mace, and everybody calls me Macey. And I am extremely happy to be here today and excited to get on with chatting a bit about what SRE means because who really knows? You know, at the end of the day, it could be anything. My official job title at Google is Shenaniganry Retribution Enforcer, in the grand tradition of SRE nonsense back titles. But more to the point, I am an SRE at Google, and I have been here for nine years. I joined fresh out of college. And I am, as I like to describe it, a mathematician who

tripped and fell into a computer. So I am very much a software engineer who has become an SRE and been one my whole career. I spent some time on the ad-serving SRE teams, where they have seven digits of queries per second, which is terrifying. I then spent a little while as the tech lead of GKE SRE, which is the Kubernetes Engine that Google runs for our customers. And nowadays, I run around trying to make Cloud more hospitable for Enterprise stuff.

Viv: You've been everywhere!

Macey: I've been everywhere. It's fine sitting here in Sunny Seattle, the last of Sunny Seattle.

MP: So one of the most, I think, well-popularized definitions of SRE that a lot of folks have seen is—I think I might be paraphrasing Ben Treynor with this one, is "SRE is what happens when you ask software engineers to take over an operations function."

Macey: Uh-huh. That's—

MP: But, like, that's not really useful when I go to talk to my friends or family. And that's not even necessarily that useful when I go to talk to other engineers. Like, how would you—right from the start—if you could redefine SRE from the start, how would you define it?

Macey: Let's see. Well, we're just another Dev team who has a different set of priorities, right, in the same way that a Security Engineering team isn't confusing to anybody. That's a team of engineers whose job is to make things secure. SRE is a team of engineers whose job it is to make sure things are up, whatever that might take. So it's less operations. I don't love the phrasing of "make software engineers do operations," 'cause that sounds like, you know, you misbehaved in high school, and now you have to clean the kitchens, go do operations until you've fixed your life. It really is another engineering discipline. It's just that our priorities are less about bringing new interesting features to our users and more

about making sure that the features out there work well and stay up and running. So I guess that's not a million miles different, is it?

MP: So I guess that sort o, begs the question, "What does it mean to engineer reliability?"

Macey: Oh, yeah. Well, engineering reliability can kinda go in a ton of different directions. And for me, that's part of why I find this discipline so fascinating, because the types of engineering that I might do in Ads, which is a high-traffic, low-latency, really fast-paced environment, might be around making sure that there are redundant servers so you can serve the traffic from anywhere in the world if you lose a data center. But the type of engineering that I would do in a Kubernetes environment, which is more about providing a single instance of Kubernetes to our users, can't use nearly the same tactics. So engineering reliability is about questions. It's about asking your developers and asking them to ask their customers, "What does reliability mean? What about your service matters if it's up and running? What would a user consider to be broken?" And then you serve that.

Viv: So when talking about SRE being a team that maintains the reliability of a service, is that one half of a team that also includes the developers, or what makes it different from the Dev team?

Macey: I think that there are a lot of different organizational models that a company can choose to field to kind of achieve the same ends. One of the things that Google really had was early on, a lot of services had a large number of developers that they wanted to kinda centralize and make efficiency improvements over the operations part. And so you might have five, ten, even more than that developer teams mapping back to a single SRE team. And when that happens, you really can't have the SREs be embedded in the Dev teams in quite the same way as you might when there's a one-to-one mapping.

Viv: That makes sense.

Macey: We do have some teams that are more embedded. I want to say that the internal SQL team works basically as a dual-purpose Developer and SRE team, for example.

Viv: I'm mainly curious because when I started as an SRE, which was about a year ago during the pandemic, I actually didn't really get to know more like, "Hey, this is an SRE role." More I was in the role of "Hey, here's what our team does. Would you like to join?" And without really knowing what SRE meant, I was like, "Yes, this sounds like fun. Sign me up." So I'm always curious about what these definitions mean and when people come and ask me because I'm just kind of like, "Here's what *I* do"…

Macey: Right.

Viv: Rather than what does SRE generally do.

Macey: I mean, me too. I was hired on a pure SWE role, and I was pitched three different teams at Google, and the SRE one sounded the most interesting. But otherwise, I would've been a developer on, I want to say, Database team, which, you know—no rudeness to databases, but I find this more fun.

MP: I think it's certainly more exciting.

Macey: Mm-hmm.

Viv: I almost feel bad that I think maybe some Dev folks never get to stumble into this, because it is definitely an interesting take on a lot of the same things.

Macey: Well, I mean, Google does offer a program so that developers can spend six months experiencing SRE and acting as an SRE, which I think is a really useful thing even if it's not necessarily something that the developer in question might want to do forever. I think it gives you different perspectives. One of the other things—one of the other ways that we talk about running systems as SREs is managing risk. I think, MP, you had some questions around that, right?

MP: Yeah, I think that was where I wanted to go next. I guess the next question I had is: we started talking about reliability, and then if you go and open up the SRE book to the third chapter, it's one of the first things that the book talks about…

Macey: There we go.

MP: …is risk.

Macey: Mm-hmm.

MP: So how does reliability map to risk?

Macey: So how does reliability map to risk? Well, we never, at Google, try to aim for 100% reliability, right? So let's start there. The first thing that you do when you're trying to measure the reliability of a system is you figure out some metric, what we call a service level indicator, and then you set a goal. What percentage of this metric do you want to be good? Usually, we'd aim like 95, 99%. Risk, then, is about spending that 1% budget, that 5% budget, of allowable mistakes. So a service that is happy to succeed 95% of the time has a much higher risk tolerance because you can spend 5% of your queries as errors, and that's completely fine and acceptable, right? Whereas if you have a service, like I once worked on one that was six nines of reliable, that's 99.9999%, that means that our monitoring system can't measure how reliable that is because it's more reliable than the monitoring. We did not have much risk allowance. We ran 20x over-provisioned. We ran 20 times as many servers as we needed 'cause we could not fail.

MP: That's like how many seconds of downtime per year does that one—

Macey: Oh, none. None. Like none. We don't actually advise that our developers ask us to do this, 'cause this is one of the funny things. Sometimes you get into a situation like you're saying, Viv. When you're on separate teams, you have to negotiate across that boundary a little bit more than when you're in the same team together. But sometimes you have a Developer team who says, "Well, my service is important, so we want a really high percentage of availability." And the

SRE turns around and says, "You want four nines? We can absolutely do that. You get to release once a month. No, you don't get that new feature." And the developer's like, "What? That's not what I wanted." "Well, then you didn't want that many nines," and they usually don't, right? So that kind of communication is also part of being an SRE.

MP: There was something else that I found really interesting when I was doing my research for this episode: the contrast drawn between stability and agility. The book—and it kind of talks about those as opposing forces. And I'm curious how you view those, given what you were just saying about how you get to release once a month.

Macey: Right. And honestly, if I had an SRE of my acquaintance who came to me and said, "We impose this on our developers," I'd be like, "Why?" Because they're not opposed, precisely; they interact with each other. A service that releases frequently generally has more stable releases because they're better at doing them, right? When you're releasing bi-weekly or daily when you have a Push On Green situation—which means that the automation will just do a release whenever your test is passing—then it's going to be a lot easier to push a fix really quickly or patch a security hole really quickly. So in some ways, agility is actually serving stability. But it's also true that most systems only break when something changes. And so when you're introducing change, you're introducing chaos. And so it really depends. It's almost a triangle, rather than a line between two things. And the point of the triangle is, you know, validation, testing, and canarying how well are you able to assert and prove that this particular change is safe and good? 'Cause if you can assert that really well, if you invest heavily in it, then you get to have both agility and stability.

MP: So the three legs of—the three points of—the triangle you would have would be validation, stability, and agility?

Macey: I guess, maybe. I don't know if it needs to be a triangle. I'm just making this up off the top of my head here. But I guess kind of how I think about it is if you invest heavily in making sure—in, like, having safety checks—then you get to be more agile and maintain a stable stability. Normally, in SRE…if you're thinking

as a physicist, right, the variable that you keep constant is the stability. That's 99%. I want that to always be 99%. I can vary the other two by dialing up the testing, which lets me increase my agility. But if I don't invest in testing and my infrastructure there doesn't keep up with changes, doesn't have good coverage on new features, then I can't be as agile and stay stable.

MP: That's actually a shift that I've seen in my own realm recently, where we're doing a lot more offline, hidden things where we're testing how the release that's going to make it to prod is going to perform days before it makes it to prod.

Macey: Right.

MP: But making sure that it looks identical or near-identical to what the live service is already doing.

Macey: Yeah. And the thing there that can be very difficult is that code and services and systems and applications can be more or less testable, depending on how they're architected from their very inception, right? And so that can get really tricky when you want to put operations as only the last thing on a checklist. This is why at Google, we really like to have SREs involved in design review early on before implementation even starts, to ask questions like, "How are you going to test that feature? Do you have a way to mock that data in production?" You know?

MP: Yeah. So the other thing that I am curious about is how does it differ when you have maybe an older service versus a newer service? You have that brand new ideal world where SREs are there from day one versus SREs kind of coming on to a legacy system that maybe has a little bit of cruft.

Macey: Right.

MP: Does the approach you would take as an SRE really change between those two situations?

Macey: I mean, simultaneously massively and not at all, which is deeply unhelpful of me to say. So let's break that out a little bit 'cause that's bad radio. It is very

different in what you will practically do, but philosophically quite similar. It's all about trade-offs, right? One of the important things to learn being a good SRE is not to rigidly insist on your solutions to problems, but to present problems and trade-offs and decisions to your developer partners and treat them as partners, with respect, and let them help make those decisions. So for a newer system, you might come in on a design and say, "Hmm, this seems like a monolithic architecture that's going to be really hard to test. Could you implement it in these microservices? Would that work?" And it's easy to do at the beginning. If you have a monolith in production already that's been running for six years, you might not go in straight away and say, "Can we break it out into microservices?" You might present that as an option if they have the staffing, but instead, you might say, "Our end goals are X, Y, and Z. Here's how we can get to them. I, as your SRE, cannot do everything at once. If you invest in making your service more reliable, then I can help invest in making it agile. But if you can't invest on your side, I'm going to spend all of my time doing the operations work, manually patching things. This isn't a punishment, but I just won't have the time to spend on the agility." So that's kind of the shift, right? It's all about presenting options and letting your partners have a hand in the decision.

Viv: So what I'm hearing here is an SRE acts almost like a consultant in those early stages and then follows up on the outcomes of whatever the developer decides to do. Does this depend on the team? How involved do SREs tend to be on the actual development of these things? For instance, like the testing, the frameworks?

Macey: So it super varies by team. At Google, we are a software engineer-heavy organization. I think it's about 60% software engineer SREs and about 40% systems engineer. So we do want to get our hands into the code base sometimes. Most of the time, that will be more like you're saying on the infrastructure side, on the release frameworks, or the testing frameworks, and helping the developers to use existing tooling. Sometimes that will involve getting deep into their actual code base, doing something, like I've been involved in implementing load shedding in the customer client code, for example. So SREs aren't just acting as consultants, but also sometimes as engineers who will come

in and work on a project inside of the Dev code base, depending on how much time you have.

MP: I have a little thought experiment. There's a new product coming up to launch, and you get to decide whether or not this launch goes forward. What are the questions you're going to be asking that team?

Macey: Well, thank you for the generous lead-in, MP, 'cause I have a little bit of a hobbyhorse, so I have my three questions of SRE, which is I want my developer to be able to answer. Before I take the pager, I want them to answer first: how does your thing break? Second: how do I know that it broke? And third: what do I do when I know that it broke? So this can mean any different sort of things, depending on how the service works, right? This could be how does your pipeline form malformed data? Or how do I get paged if the queries are broken? Anything, right? But that's the sort of things that I want my developers to be thinking about when they're launching, when they're saying, "Can I launch this thing?" Well, if you can answer these three questions to my satisfaction, then we'll have a chat.

Viv: I really like it. I feel like I should be asking—this should just be like a thing. Every time somebody asks me a question now and they're like, "Hey, we want to put this forward," I just want to be like, "Hey, see, you missed these questions here." This is the structure I'm missing.

Macey: Sometimes—a lot of the time in a well-organized system—the "How do I know that it broke?" should mean "It will make your SLO tank," right? If it's part of the main flow of the service, and if it breaking will cause your users to be unable to use the service, then it should show up in the SLO. That's the service level objective. But not every feature is in the main flow. Sometimes it's just a nice add-on, right? If they want to page an SRE about it, they need to be able to answer that.

Viv: I could also see it as working in the reverse way as a "How do I know it broke?" Like, "Maybe this should be in our SLOs," right?

Macey: Right.

Viv: "We didn't previously consider it, but now we want it as part of our thing so that SREs know what to do or when to do the do.

Macey: Exactly.

MP: The first question, "How does it break?" sounds like one that's kind of hard to answer exhaustively.

Macey: Mm-hmm. Yeah, I mean, this is—again, this is the sorts of things that you ask in conversation, 'cause, to me, being SRE is all about asking questions, being in conversation. So I would maybe give examples to the developer who owns the feature by asking, "What would it be like for this to be broken? What would a user experience? Okay, what would that look like on internal metrics?" Right? "What does broken mean for this feature? What would you as its developer consider to be not working enough that you would want an SRE to do something about that fact?" 'Cause we don't take 100% responsibility for every single aspect of a code base when we go on-call, right? Some services are what we call fully onboarded; some services are called assisted tier or partially onboarded, where we only make sure that the service is running but not that the business is going correctly. But even for things that are fully onboarded, you can't make sure that absolutely everything is functioning, right?

MP: There's a lot of business logic and a lot of services, and I know how all the parts [are] put together, but I don't really know what each part does.

Macey: Right? It's like how did, two weeks ago, this shift in the machine learning pipeline cause a change in that advertiser's profile, which caused a revenue shift in that budget system, which I have no idea [about]? It's not my job to know. None of those things are for me to debug as an SRE.

MP: What about external constraints with launches in SREs?

Macey: What kinds of external?

MP: Well, like SLAs in this case, I think, is what I would be interested in here. When there's external factors, not just the users, but maybe some—

Macey: Some kind of promise—some kind of promise that you've made? Yeah.

MP: Yeah. Like, does SRE really ever think about contractual agreements in SLAs, or do you just somehow try to avoid that?

Macey: Well, some of the stuff—there's a difference between things that the leads of an SRE team are thinking about and that every single on-call has to be aware of all the time, right? I would expect that the leads of a team have some awareness. So an SLA, a service-level agreement, is kind of a written obligation, almost, of the service owner to the users or the customers or the client, right? For example, I believe at one point, the GKE system had an SLA that said there would be no more than a certain percentage of downtime for any given Kubernetes instance. When a launch is being suggested or an outage is underway that threatens an SLA that is running up against, you might fail your contractual obligations, or internally, you might cause your customers to be broken, too, because you're breaching your SLA. That is the time when even the most junior SRE on-call starts having director authority. This is somewhat unofficial, right? But the SRE on-call gets to tell managers what to do, gets to force deploy engineers and say, "You need to drop everything. This is an emergency." Or, "You don't get to launch this thing because the risk profile butting up against this agreement we've made means that I don't think we can do this safely." And sometimes that means, you know, if a junior engineer ends up making that call, they'll probably bring in their tech lead and say, "Please support me. Back me up. Make sure I'm saying this right." And a big outage, you'll pull in your SRE colleagues as well. But there is a power in that relationship that SRE does have when they think something is in danger, and it's a power we have to be careful not to misuse. But it's important 'cause that's our job.

MP: The phrase, when I started learning to go on-call, that my manager liked to use was "You have the keys to the car."

Macey: Mm-hmm. "Don't drive it off a cliff."

Viv: So we've talked a little bit about the constraints for "How does it break?" Who makes those decisions? There's "How do I know?" and we said, you know, maybe service level objectives. I'm curious about "What do I do?" because as we've kind of briefly touched on a little bit, SRE is a very flexible role, right?

Macey: Right.

Viv: And I think maybe "What do I do?" can sometimes be, "Let's take our time and fix this," or it could be, "We need to drive this car really fast to avoid getting hit by an avalanche." So, what are your thoughts on "What do I do"?

Macey: Well, I'm sure that you guys probably have a later episode planned to go into a bit more depth on this, so I'll keep it fairly brief. But an SRE's first job is to figure out the impact when you're paged, right? Being on-call is not everything an SRE does, as I hope we've probably already touched on. But first, you figure out what the impact is, and then you try to mitigate that impact to the user as best as possible, and then you try to figure out exactly how it happened, in that order. So mitigate first.

MP: I'm always eager to hit the rollback button.

Macey: Hmm, yes, there's—and folks can find this—I have an article out with O'Reilly on generic mitigations. But basically, the idea is if you are an SRE with responsibility for a service, and that service matters to you, if you care about it, then you should probably have one or two things like a rollout that are actions you can perform when everything's broken and you don't know why. So in Ads, we used "drain to a different cell." If one cell is just acting weird, move all the traffic somewhere else. Figure it out later.

Viv: Yeah, I like it. Keep those 99.999s up.

Macey: Yeah, I mean, the job is to make the service work for the user. It can do that perfectly well from a different cell, hopefully.

MP: So is the worry about cascading failures?

Macey: Right. And cascading is its own form of entertainment. But that's work you have to do in advance.

MP: I had to deal with a bit of a failure loop the other week, and it was just so—just not even sure if it's in a failure loop or if there is something else going wrong.

Macey: Yeah. Yeah.

MP: I'm just like, "I can't drain if it is, 'cause then it's just gonna move the failure loop."

Macey: Well, one of the things is if you have these generic mitigations, you drain one cell, you data roll back a different cell, you binary roll back a third cell—whichever one of those fixes—you do it globally if you're in a bad situation. But in case our listeners—I don't know if "cascading failure" is quite a Google-specific term, but basically, when you have a bug or a problem that causes crashes in your servers and then you move it to a different cell and they all crash, and then you move it to a different cell and they all crash, nobody's having a good time.

MP: The outage that chases you.

Macey: Yes. I'm like, "Why? Go away. Nobody asked you." One time we had an entertaining Belgian who kept doing that to us, so it was very fun.

Viv: This is all making me a little nervous because I'm actually on-call right now. And now I—

Macey: Oh, no.

Viv: Oh, not right now, right now. I will be on-call in, like, half an hour. And now I'm worried that all these things we're briefly mentioning are going to happen.

Macey: It's fine if they do. It's all investment in your education.

Viv: Thank you. Thank you. I'm looking forward to draining a cell, rolling back a binary, and whatever the third one was.

Macey: Yes. It's great fun.

MP: So I think we'd be a little remiss in a conversation about "What is SRE?" to not mention this particular buzzword, which I think responding to pages technically counts as.

Macey: No. Well, okay, how do you—would you like to state us a buzzword, MP?

MP: Toil.

Macey: Toil. And everybody immediately makes devil horns. What would you or the book define toil as, MP? I say it [while] seizing the microphone.

MP: Well, I'm not great. I struggle to define it. And then the book just gives you a list of questions, and it's like…

Macey: Oh, God.

MP: "If you answer yes to more than one of these, maybe it's toil."

Macey: So I will give you, then, my definition of toil. Toil is boring or repetitive work that does not gain you a permanent improvement.

MP: I don't think responding to pages is boring.

Macey: Well, boring or repetitive.

MP: They can definitely be repetitive.

Macey: So I would say—I wouldn't put responding to pages as toil unless you're getting the same page or the same type of page repeatedly, and then it's toil. Because the first time you get a page, you're actually doing interesting

investigative work and you're learning things. And that is permanent improvement, right? You have gained something. Boring refactoring work that makes your code base better in the long run is not toil. It's just boring. The reason that SREs talk so much about toil is because toil is the environment of the Ops engineer, right? And so it's kind of in our prehistory. It's in our background. And we don't like it, and we try to get as far away from it as possible by doing things like automating our release process. Manually running a release of a binary is toil. Has not gained you anything. You haven't learned anything. You're just doing it. An SRE will sometimes do manual steps, if that's what's needed to maintain stability. But they will try as hard as they can to automate it or to improve the process so it's not needed. Because any work you do to remove toil basically gives your team back that many engineering hours of a skilled labor to put towards things which gain you permanent improvements. So that's kind of my philosophy towards toil and why it's bad. It's bad because it's a tax on your time and because people hate it, and thus, it's a tax on your morale, too.

Viv: So I guess you could even say that to the part where maybe you're getting pages that are recurring. Something becomes toil when you see it enough that there is a solution to automating or fixing it or putting it away.

Macey: Mm-hmm. SRE likes to write playbooks, but when you get to the point where you have a playbook that you get paged and you exactly follow steps one, two, three every time, write a robot. Write a cron job, you know? You write a script. We did this back in Ads. We ran a lot of C++ binaries, and sometimes they would get crashy or OOM. We had a restart server that when a particular server thought that it was getting unhealthy, if it thought it was getting stuck, it would send a little message to the restart server, and the restart server would come back, bop it over the head, and it would reboot so that SREs didn't have to go and reboot it. It's like, "Please resurrect me. Okay."

MP: I think my team runs something similar.

Macey: Yeah. Oh.

Viv: So is it possible if a team—say, toil appears—a wild toil appears, you fix it…

Macey: Fight item. Run.

Viv: Yeah. It goes away and, you know, repeat, repeat. Is it possible that SREs will eventually not have anything to do?

Macey: It hasn't happened yet. I think that the type of review work that we do, the type of consulting work we talked about earlier, all of that is always happening, and our developers are always introducing new features, creating new services. We're always getting new platforms, like the Kubernetes model, the Anthos model. All of that stuff is always coming. So there's always new stuff coming up, and that will always give us interesting things. And also, if you get to a point with a service where it's just too stable—it never pages you, it's boring—you can hand it back to your developers and say, "You've done a great job. This is a win. You have won."

MP: We can all dream.

Viv: I like it.

Macey: I've had that happen in the past.

MP: Really?

Macey: I had a hilarious situation where there was a geo backend that Ads needed to be way more reliable than it was. So we went to the geo team and were like, "We really need to know where the users are. Can you make us better at that?" And they were like, "No, we don't need that. We're fine at 90% reliable." Ads needed four nines. It was at one nine. So we were like, "We'll just be taking that service then. We took it for two years, we got it up to four nines, and we handed it back like, "Here you go."

Viv: That is pretty cool.

Macey: It can happen.

Viv: Wow, I love it.

MP: Well, on that note—

Macey: On the note of wrapping things up neatly and handing them back—

MP: Thank you so much for your time today, Macey.

Macey: It's fun. Thank you for hosting me.

MP: We will make sure, for all those interested, that the article on generic mitigations is in the episode notes.

Macey: Yep. There's another one kicking around about multi-single tenancy that is illustrated with cranes because I had a very enthusiastic artist who really wanted to draw birds for it, so we drew birds.

Viv: I was just about to ask, are they bird cranes or construction cranes?

Macey: I asked him this as well because the other one seemed less fun.

Viv: Like friendly construction cranes with birds working together.

Macey: There we go. It's SRE. We love our metaphors. Changing the wheel while the car is racing.

Viv: Well, it was really lovely to chat with you.

Macey: Awesome. See y'all later, and good luck with the new podcast.

MP: Oh, thank you so much.

Macey: Yay!