

Module 2: DevOps, SRE, and Why They Exist

Now that you understand what this course will cover, let's get started.



In this module, you'll learn about the practice of DevOps, why Site Reliability Engineering, or SRE, came to exist, and who in an organization can and should practice them.

But first I want to tell you a story of an online retailer. Their application is similar to so many other online retailers. Customers browse the catalog, add items to the cart, then complete their purchase.



The operations team at this online retailer holds weekly reviews of key metrics.



At a recent review, the team noticed that the time it was taking between the customer clicking pay and the status coming back as confirmed was slowly increasing. They recognized this was not a critical issue but did need to be addressed. The team spent some time quietly working on addressing the latency.



Over time, on the business side of things, the product development kept pushing features. Can you guess what happened?



Well, developers started working overtime not only to keep up with the business requests of pushing out new features, but also to resolve the "small" latency bug that was identified earlier.

The product teams still weren't pleased with the pace of development. Basically, the IT teams put forth a heroic effort to not only satisfy the business, but also fix the bug—they suffered from burnout in order to meet the needs of both the business and reliability.



Later, when it came to light what the IT team was doing, the product teams agreed that if they had known about the latency bug, they definitely would have prioritized fixing that first before pushing new features. But because the business and IT didn't have shared standards for communication, this didn't happen.



So, what can you learn from this? Do you think there are ways that the business and IT can communicate better? How do you think you could achieve this in your organization?



Google thought a good place to start was changing the way we think about, measure, and incentivize reliability. We call this collection of principles and practices Site Reliability Engineering, or SRE.



Perhaps you've heard of the DevOps movement. It's a movement that, like SRE, strives to align principles, practices, and incentives across teams. **DevOps** and **SRE** have a lot in common and they are often discussed together. In fact, you may be wondering how they are different. We will explore some of the common ground between the two and how they complement one another in the next video.

What is DevOps?

So, what is DevOps? To understand what it is, we first need to understand why DevOps exists.



Traditionally, IT teams consist of developers and operators. Developers are responsible for writing code for systems, and operators are responsible for ensuring those systems operate reliably, so customers are happy.



Developers are expected to be **agile**, and are often pushed to write and deploy code as quickly as possible. Essentially, developers want to **work faster, innovating and succeeding (or failing) quickly.**



This resulted in developers throwing their code "over the wall" to the operators, who then had to deal with code that was written without much understanding of how it would run in production. Operators, who are expected to keep systems **stable**, would prefer to **work slower**, focusing on **reliability and consistency**.



Quite understandably, this way of working wasn't sustainable between these two groups. Their priorities caused tension between the two teams and were not necessarily aligned with the needs of the business. And so, as a way to knock down the wall and close the gap between developers and operators,



a culture and set of practices known as DevOps was born.

Let's take a look at how Google breaks down DevOps.

There are five key areas.



The first is to **reduce organizational silos**. You can increase and foster collaboration by breaking down barriers across teams.



Secondly, you need to **accept failure as normal**. Computers are inherently unreliable, so you can't expect perfect execution. And when you introduce humans into the system, you get even more imperfection. Things failing will inevitably become part of the process.



Thirdly, you'll want to **implement gradual change**. Small, incremental changes are easier to review. And in the event that a gradual change does release a bug in production, it allows you to reduce the time to recover, making it simple to roll back.



Fourthly, you need to **leverage tooling and automation**. Identifying manual work that you can then automate is key to helping your IT team work efficiently and focus on the tasks that matter.



And lastly, you'll want to **measure everything**. Measurement is a critical gauge for success. There's no way to tell if what you're doing is successful if you have no way to measure it.

It's important to understand that DevOps is a **philosophy**, versus a development methodology or technology.



While DevOps philosophy highlights critical ways for IT teams to operate, it doesn't give explicit guidance on *how* an organization should implement practices to be successful.

That's where SRE comes in.



So what is SRE?

SRE evolved at Google in the early 2000s, separately from DevOps. Back in 2003, Benjamin Treynor Sloss, currently a VP of Engineering at Google, was tasked with managing a team of engineers who were responsible for keeping Google's websites up and running.



You're probably wondering, "Wait... so a bunch of software engineers who write the code now also had to be responsible for running their production systems? But doesn't the ops team do that?"

Well, the answer, at least traditionally, is yes. However this team only had software engineers, so Ben had them spend some of their time on operations tasks in addition to development tasks, so they could better understand how their code ran in production.



This way of working is what led to the term Site Reliability Engineering, and the associated job role, where SREs, who are generally engineers, are responsible for operations.

Just as DevOps aims to close the gap between software development and software operations, this new SRE approach is a concrete way to solve the problems that the DevOps philosophy addresses.



Note that SRE is both a role and a practice. Not every organization that follows SRE principles necessarily must have engineers with the title "SRE". This course mostly focuses on the practices and principles themselves; things like titles and team structures are implementation details.



There are a number of SRE practices that align to Google's categorizations of DevOps. And you should not only implement SRE **technical** practices,



you'll also want to implement **cultural** practices. Without a culture to sustain them, it is not possible to maintain the practical aspects of SRE.



Let me explain where these fundamentals fit into the key areas of DevOps we discussed.

When it comes to reducing organizational silos, SREs **share ownership** of production with developers. Together, they define Service Level Objectives, or SLOs, and error budgets, sharing the responsibility of how they determine reliability and prioritize work. Culturally this promotes **shared vision and knowledge**, as well as a need for improved **collaboration and communications**.



Complex systems fail in interesting and complex ways. Accepting failure as a normal state is an important practice within SRE. A blameless post mortem is held after an incident to improve the understanding of the failure mode and to identify effective preventive actions to reduce the likelihood or impact of a similar incident. Learning from incidents in this manner requires a culture of **psychological safety and blamelessness**.



When implementing gradual change, SREs aim to **reduce the cost of failure** by rolling out changes to a small percentage of users before making them generally available. Culturally this promotes more **design thinking and prototyping**.



Next, in order to leverage tooling and automation, SREs focus on **toil automation**, reducing the amount of manual, repetitive work. Automating this year's job away can undoubtedly be met by resistance. That's why teams need to talk about and understand the **psychology of change** and how to address **resistance to change** within the team.



Finally, measure everything means that SREs work to **measure everything related to toil, reliability, and the health of their systems**. To foster these practices, organizations need a culture of **goal setting, transparency, and data-driven decision making.**



Before diving into each of these pillars as they relate to SRE, I want to acknowledge that sometimes people can see these practices aligning to different or many, of the pillars of DevOps. It is likewise important to acknowledge that the language and definitions across different organizations are less critical than driving towards the goals of your organization and the outcomes you are trying to deliver for your customers. In other words, we may disagree on the precise terminology, but many of the underlying principles are the same. The goal of SRE is to serve the business and the user, not the other way around.

The next three modules will dig a bit deeper into all of these SRE practices and cultural concepts.