SRE 101: Key Terms + Concepts

StackPulse

Contents

Introduction

If you do work related to site reliability engineering, incident response or even just plain-old DevOps, it's easy to feel like you are drowning in a sea of IT and dev-related terms and abbreviations. The IT world, in general, is riddled with acronyms and abbreviations (Wikipedia lists hundreds of them) that can be hard for the uninitiated to decipher; but the world of SRE (that's short for Site Reliability Engineering, FYI) and incident response lean especially heavily on them.

If you've ever found yourself struggling to figure out what a SRE or DevOps term stands for, or if you just want to bone up on SRE-related skills and concepts — this guide is for you. In the pages that follow, we walk through more than a dozen common terms and acronyms, then explain what they mean and where they fit as concepts within the SRE landscape.

Our hope is that, by working through this guide to SRE terminology, you'll sound a little smarter or at the least, feel a bit less out of the loop the next time you find yourself peppered with specialized jargon during an incident response or monitoring operation.

A/B Testing

A/B testing is an approach to software testing and deployment wherein an application feature is made available to one group of users -- the "A" group -- first, while remaining turned off for other users -- the "B" group.

By providing access to a new feature to only a subset of your users, you can monitor and evaluate how users respond to the feature, and how well it works, before fully deploying it. A/B testing helps ensure that a problematic feature update doesn't impact all of your users at once. It also makes it easier to roll back a failed feature update, because you only need to roll it back for certain users.

A/B testing is a technique most commonly associated with development and deployment more than SRE. However, because it offers a way of controlling and assessing potential disruptions to end-users that may be caused by a new software release, it's a valuable strategy for maintaining site reliability, too. If you work in SRE, it may be worth your while to get buy-in from your devs and IT team for A/B testing, rather than deployment strategies that push changes unto all of your end users at once.

See also ***canary release.***

APM

APM is an acronym that stands for two phrases that are seemingly similar, but actually have quite different meanings.

One phrase is Application Performance Monitoring. In this usage, APM refers to the process of collecting application performance metrics, like CPU and memory usage, then monitoring for signs of a problem.

The other phrase is Application Performance Management. This is a more holistic concept. It includes monitoring application performance metrics, but it extends to performance optimization as well. In other words, when you're responsible for Application Performance Management, your goal is not just to identify performance problems, but to find ways to improve performance over time by, for example, tweaking your deployment architecture or resource allocation.

(cont'd on next page)

Application Performance Management may also include tasks like managing autoscaling policies in order to ensure that applications always have enough resources available, while avoiding costly over-provisioning.

Canary Release

A canary release is similar to A/B testing in that it involves making a new application version available to certain users before others. The difference is that whereas A/B testing focuses on releasing certain features to some users first, a canary release makes an entire new application version available.

The canary release methodology helps you determine whether a new release is going to cause problems before you roll it out to your entire user base. Canary releases are also sometimes used as a means of providing early access to cutting-edge features to people who want them, such as power users who are willing to sacrifice stability for the latest and greatest features.

From an SRE standpoint, canary releases, like A/B testing, are a great way to mitigate the risk of instability even as your application delivery team rolls out new features quickly. Whether you choose to use A/B testing or canary releases depends largely on whether you have features that can be released discretely and individually (in which case A/B testing works well) or need to release collections of feature updates as a new application release (which is when canary testing makes sense).

*See also **A/B testing**.*

Chaos Engineering

Chaos engineering refers to a strategy where you experiment with production environments in order to ensure that they work as required. In a sense, it's the opposite of shift-left testing, which emphasizes performing tests prior to deploying software into production.

From an SRE standpoint, you'll ideally vet your releases both before and after they go into production. Pre-deployment testing will mitigate the risk of letting problems reach your end-users. But because you can't always catch every issue with those tests, chaos engineering is valuable as a means of finding issues that are lurking within production.

(cont'd on next page)

Chaos engineering also offers the benefit of helping to discover problems that are triggered by rare conditions that you may not otherwise emulate in your tests. In other words, by experimenting and "kicking the tires" of your production environment as you look for problems, you can catch issues that wouldn't show up in a normal monitoring workflow because the conditions necessary to trigger them wouldn't be present before chaos engineering happens.

Incident

In the context of SRE and incident management, an incident is any event that threatens the performance or availability of an application or service.

Incidents don't have to cause an actual disruption in order to qualify as incidents. Anything that poses the potential of becoming a disruption is an incident.

Nor do incidents need to pose the risk of total disruption to a service. An incident could instead involve a degradation of performance, such as an application taking longer than desired to respond to requests.

Incident Response

Incident response refers to the art and science of responding to incidents.

Incident response may seem straightforward, but it can be quite complex. Because most teams are inundated with a never-ending stream of alerts about potential incidents, effective incident response requires the ability to evaluate quickly which issues are real threats and which are false positives.

Likewise, teams must be able to assess the severity of incidents quickly so that they'll know which ones to prioritize when responding.

Assigning the right personnel with the right tools and expertise to handle each incident is critical, too.

MTBF

Mean Time Between Failures, or MTBF, is a measure of how frequently disruptions occur to a given application or system. It's typically calculated by dividing the total number of hours that a resource runs by the number of incidents that impacted the resource in that period.

MTBF is useful as a way of identifying which components of your environment are subject to recurring failures. A high MTBF rate is a sign that there are likely underlying issues that you should address in order to improve the reliability of an affected component.

Likewise, if you are seeing high MTBF rates for the same system but the individual incidents appear unrelated, you should dig deeper, because there's a good chance that the seemingly disparate disruptions are actually linked to the same root cause. By finding and fixing the root cause, you'll improve your MTBF rates.

MTTA

Mean Time to Acknowledge, or MTTA, measures how quickly your team begins responding to an issue. It's similar to MTTD, with the difference being that MTTD measures how fast the team noticies a problem, while MTTA measures how long it takes for the actual response to begin.

Effective incident analysis and triaging is critical for improving MTTA. By minimizing the time it takes your team to analyze each alert as it comes in, as well as helping your team to determine which priority level to assign to each incident, you can speed the rate at which response begins for critical incidents.

*See also **MTTD**.*

MTTD

MTTD, or Mean Time to Detection, measures how quickly your team identifies an incident.

(cont'd on next page)

Importantly, MTTD doesn't refer simply to how quickly an incident becomes evident within your monitoring or logging software. You need to calculate it based on when your team actually becomes aware of the incident, which may take some time, especially if you lack efficient alerting and notification tools.

In addition, optimizing MTTD requires tools that can identify false problems. By minimizing this distraction, you increase your team's ability to detect issues quickly.

*See also **MTTA**.*

MTTR

Mean Time to Recovery, or MTTR, refers to how long it takes to complete resolution of an incident. (MTTR is sometimes translated to Mean Time to Repair or Mean Time to Resolution, which refer to the same thing. However, MTTR shouldn't be confused as a stand-in for Mean Time to Respond, which is better described as MTTA)

MTTR is a measure of the overall effectiveness of your monitoring and incident response workflows. The faster you detect incidents and acknowledge them, the faster you'll achieve resolution.

The ability to communicate effectively about incidents, and to perform fast root-cause analysis, also informs MTTR.

Post-Mortem

A post-mortem (which is sometimes also known as an After Action Review, or AAR) is a formal assessment of an incident after response is complete. A post-mortem identifies the nature of the incident, which resources it affected, how severe it was, which steps the team took to respond, what the root cause was determined to be (if known) and -- most important -- which steps the team will take to prevent the same type of incident from happening again.

(cont'd on next page)

Usually, it's not realistic to perform a post-mortem for every incident. Minor incidents, or those that don't pose a critical threat, don't need a complete post-mortem. But major incidents that cause a critical disruption or came close to it, as well as incidents that had a complicated root cause or required an unusual response, are good candidates for post-mortems.

Toil

In the world of SRE, toil describes work that is manual, tedious, repetitive, unnecessary or otherwise of little value. Time spent sorting through monitoring data in order to analyze an incident when an automated analysis is available is one example of toil. Another is wasting time trying to implement a fix that doesn't resolve an incident, due to mistaken assumptions about the incident's root cause. A third is when two engineers are performing the same task at the same time because they have not communicated effectively as they proceed with incident response.

Toil is bad not just because it wastes resources and time when handling incidents, but it also decreases team morale. Automation, strong communication across your team and effective root-cause analysis are critical for reducing toil.

SLA

A Service Level Agreement, or SLA, specifies exactly how much uptime and (in some cases) which level of performance a service provider guarantees to its users. An SLA might promise 99.9 percent uptime, for example, or it could guarantee application response time of 200 milliseconds.

An SLA is just a contract. It's up to the SRE team to ensure that SLA guarantees are actually met.

A related acronym is SLO, which stands for Service Level Objective. An SLO is a quantifiable SLA metric that you aim to meet (such as "99.9 percent uptime)."

Similarly, SLI, or Service Level Indicator, measures your actual uptime or performance. An SLI of 99.8 percent uptime means you are falling short of your SLO, if your SLO is 99.9 percent uptime.

SRE

SRE stands for Site Reliability Engineering, which refers to the use of software-engineering principles -- such as designing systems to be resilient by default and building availability testing into the software design process -- to improve the availability and performance of IT resources.

The term also sometimes refers to the generic practice of optimizing availability and performance, without special emphasis on bringing software-engineering principles into the picture.

However you choose to think about the relationship between software engineering and SRE, what's most important is the idea of taking a systematic, consistent approach to software performance and reliability. The ultimate value of SRE lies in the way it turns monitoring and performance management into a distinct discipline, rather than just another task to be performed by IT engineers (which is how organizations historically approached monitoring and uptime).

The term was made popular by Google starting in the 2000s, but it is now widely used in a variety of organizations.

SRE may sometimes also refer to Site Reliability Engineer, a specific job role whose main responsibility is to oversee Site Reliability Engineering.

Conclusion

Whether you're an SRE or you just help out with incident response alongside other job duties, knowing the ins-and-outs of SRE jargon will help you to communicate with incident response teams as effectively as possible. Just as important, it provides crucial insight into the conceptual foundations of modern SRE, which are what make SRE the distinct discipline that it has become.

To learn more about how to put the SRE concepts defined above into practice, we invite you to Get a free trial of StackPulse to see what we have to offer. Stackpulse offers a complete, well-integrated solution for managing reliability — including automated alert triggers, playbooks, and documentation helpers.

StackPulse

StackPulse is a reliability platform for SREs, developers and on-call engineers. It enriches, analyzes and triages alerts, giving responders a head start on remediation with real-time insights. Remediation-as code playbooks standardize and automate fixes. Teams use StackPulse to reduce alert fatigue, decrease toil, bring down MTTR and meet SLOs.