WHITEPAPER

APPLYING SRE

To Communications Applications

Hope is not a strategy; wish for the best, but prepare for the worst.

CONTENTS

INTRODUCTION

The term Site Reliability Engineering (SRE) was coined in the early aughts by Google. It is a set of practices and principles "...engineering teams can use to ensure that services stay reliable for users." In their own words, the ethos behind SRE can be summed up this way: "Hope is not a strategy; wish for the best, but prepare for the worst."

In our paper *Reducing Complexity In Communications Applications,* we covered the importance of building a flexible, ultra-scalable, highly reliable application and how the right CPaaS partner can help you reduce complexity and go serverless. SRE goes hand-in-hand with a well architected system. Through testing, defining realistic Service Level Objectives (SLOs) and Error Budgets, and deploying monitoring and alerting systems, your well-architected application can remain up-and-running even during something as unforeseeable as a natural disaster.

While there are a number of SRE resources available for you to take a deep dive (published by Google and many others), the purpose of this paper is to:

1. Provide you with a high-altitude view of how to embrace a robust, healthy SRE culture around a communications application.
2. Highlight the consequences of not embracing SRE in communications.
3. Equip you with the tools to choose the right partner(s) to assist with your SRE strategy.

SRE as Culture

At Voxology, we see SRE as a way of life. It's not merely the job of a group of individuals or a certain department... it is a company-wide philosophy, and it's embedded in one of our core values, evolution. SRE requires continuous monitoring of your state and iterating to make it better. When every employee sees providing a reliable customer experience as their job, the historic walls between Support, Development, and Operations can come down, empowering teams to work together to improve their systems and processes. There are a number of hallmarks of SRE in practice, including defining SLOs and monitoring metrics (SLIs), but ultimately SRE is about far more than just monitoring and alerting — it's about the businesses' relentless commitment to continuous improvement.

Do I Really Need SRE?

Do you *really* need clean water? Technically, you could drink questionable looking or smelling water and hope for the best. And, if you do, eventually, your body will report back on the cleanliness. Biological alarms would go off. Internal systems would fail. Meetings would be missed. Dysentery would claim members of your party, just like in the video game *Oregon Trail.* But hey, at least you'd know for sure that the water wasn't clean by then.

If you don't have a site reliability strategy in place, you are pretty much "testing the water" by relying on your customers (the body) to tell you when your services are down or running slow. Site reliability can be done that way, but it's a surefire way to lose customers and revenue.

As you'll see, treating your communications application like a living, breathing organism — with various types of illnesses and injuries that can (and will) arise — is a crucial aspect of running a successful and reliable communications application.

DEFINING UNRELIABILITY BUDGETS & SERVICE LEVEL OBJECTIVES

What is an Unreliability Budget?

"... a clear, objective metric that determines how unreliable the service is allowed to be within a single quarter."

In order to determine an acceptable Unreliability Budget for your application, you first need to define your quarterly Service Level Objectives (SLO) — for example:

1. Our average error rate should be less than 0.001%
2. Our average latency should be less than 200 milliseconds

By defining your SLOs you are able to decide what parts of your ecosystem, if any, need to be up and running 100% of the time. According to Google's SRE team, the optimal way to go about defining your SLOs is to think about them from the perspective of a user:

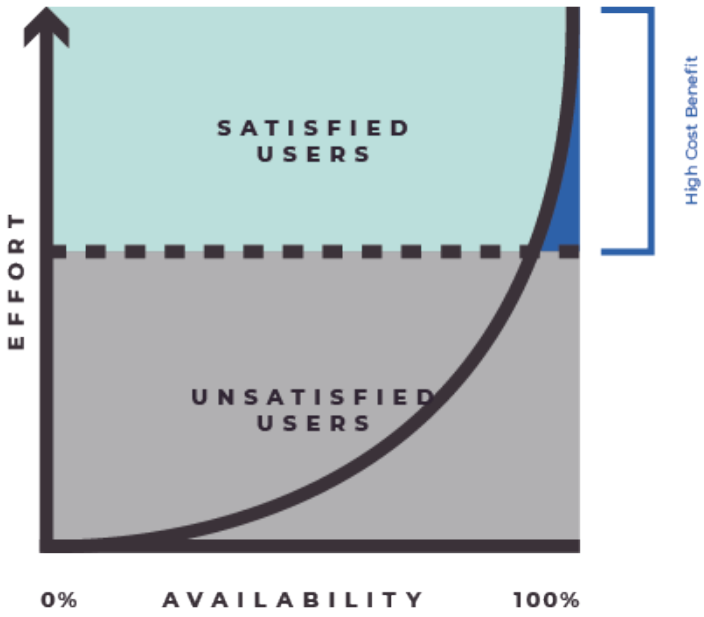
"In general, for any software service or system, 100% is not the right reliability target because no user can tell the difference between a system being 100% available and 99.999% available. There are many other systems in the path between user and service (their laptop, their home WiFi, their ISP, the power grid...) and those systems collectively are far less than 99.999% available. Thus, the marginal difference between 99.999% and 100% gets lost in the noise of other unavailability, and the user receives no benefit from the enormous effort required to add that last 0.001% of availability."

Questions to consider when defining SLOs:

What level of availability will the users be happy with, given how they use the product?

What alternatives are available to users who are dissatisfied with the product's availability?

What happens to users' usage of the product at different availability levels?



Per Jack Shirazi's article "The Cost of 100% Reliability", As a rule of thumb, each "9" you add to your 99% availability costs 10 times more to achieve. And, at some point, there will be increasingly limited perceived value to your end users.

Once your quarterly SLOs are defined, you then know how much wiggle room you have each quarter to deliver new updates that could, potentially, cause temporary service interruptions.

The delta between your Quarterly SLOs and the accrued down time is your unreliability budget. Because your SLOs have been defined and agreed upon by all relevant stakeholders, the best part about the unreliability budget is it removes personal feelings and office politics when it comes to deciding whether or not to push out new updates. It imposes a self-regulated system in which the Product Development team knows how much risk they can take at any given time when contemplating updates.

The next step in building a solid SRE culture is to figure out what to monitor.

WHAT TO MONITOR

To paraphrase an already ambiguous phrase, when it comes to errors, there are the known-knowns, and the known-unknowns.

Thankfully, through proactive and visual monitoring, it's possible to shed a bright light on these errors before they bring your whole application screeching to a halt — triggering a deluge of calls and emails from angry customers.

There are a handful of excellent open-source monitoring and visualization platforms available, such as Prometheus and Grafana, that allow you to capture and visualize time-series data, as well as **create custom dashboards, queries and alerts so that you can better see what's happening under the hood.**

There are also a number of businesses who specialize in telecom monitoring and alerting. If you are interested in this approach, let us know and we would be happy to make an introduction. These tools are critical to your site reliability success as they enable you to to take actionable steps in real time to remain within your error budget and meet your up-time SLOs.

Visualized data in the above Grafana dashboard: time series data monitoring latency, traffic, saturation, and errors (the "4 Golden Signals")

Four Golden Signals

According to Google's SRE team, the **four "Golden Signals"** to monitor are: ***latency, traffic, errors, and saturation.*** Comprehensive monitoring of these Golden Signals allows you to notify a human to step in and solve critical issues as they arise, thus reducing the length and impact of service disruption.

1. Latency

Latency is defined as the time delay between when a signal is produced at the source to when it is received at the destination. It is critical to delineate between the latency of a successful request vs. a failed request. Both are important, and you don't want one to factor into the measurement of the other. Examples of latency issues that impact a user's experience with your communications application are:

HTTP Callback Response Latency // How long it takes for your application to respond to callback requests from your communications provider. In the case of Programmable Voice, when you place (or receive) a call, your communications provider sends a callback request to your server to fetch the instructions for what you want to do with the call. When your server is slow to respond, it creates a poor experience for your users which can lead to callers hanging up or complaining about Post Dial Delay (PDD), silence, or error messages.

Audio Latency // How long it takes for the call media to reach the other end of the line. High latency manifests as a noticeable delay in audio. The maximum acceptable latency is 150ms. Anything greater than that, and your callers may end up talking over each other or wondering if the other person is even on the line anymore.

2. Traffic

Traffic is defined as the demand being placed on your system. In communications, it could be the amount of data or the number of messages traveling through a specific channel or circuit at a given time. The more connection points the data must travel through, and the sheer amount of data traveling through the channel, can increase delays in sending or retrieving information. Examples of traffic measurements in communications applications are:

HTTP Callbacks Per Second // How many callbacks your server is handling per second. NOTE: every call produces two (or more) callback requests to your server.

Calls Per Second (CPS) // How many call requests are being made per second.

Messages Per Second (MPS) // How many message requests are being made per second.

Comprehensive monitoring of these Golden Signals allows you to notify a human to step in and solve critical issues as they arise.

3. Errors

A no brainer, really. An error is what happens when an expected response to a request fails. Keep in mind that failures can be obvious, like HTTP 4XX/5XX, or not-so-obvious, like an HTTP 2XX with the wrong response body. Regardless, you must have a clear view of how many errors are occurring. The next questions are where and why? Knowing the *where* allows you to shut down the part of your distributed system that's experiencing problems. The *why* points your engineers towards the solution. Examples of error measurements in communications applications are:

SIP Code, 486 Busy // A higher than normal rate of busies can indicate a telecom issue.

SIP Code, 503 Service Unavailable // In SIP, a "503" means the carrier can't or won't attempt the call. High rates of 503s could mean it's time to switch to a backup carrier.

HTTP Code, 429 Too Many Requests // Your API vendor may be rate-limiting you; if so, this could be a sign that there is/was recently an issue with your vendor, a bug in your code, or an issue with your error retry and backoff logic.

4. Saturation

Saturation is defined as congestion in the service or how "full" it is. Many systems are impared long before they reach 100% utilization, so it is critical to measure the most constrained resources in the system — be it CPU, network bandwidth, memory utilization, etc. Many of these types of issues can be mitigated with proper load management and scaling. In communications, this can lead to a number of issues, including the following:

Packet Loss // The percentage of packets lost between transmission and the destination, which can cause a number of different issues that impact call quality.

Data Loss // Call/message logs or call recordings can go missing or be incomplete.

TO ALERT OR NOT ALERT

Now that we've covered the basics of what kinds of data to monitor, it's crucial to think strategically about how to handle the ensuing alerts. Two major concerns of SRE management are mitigating burnout and reducing the chances of desensitization.

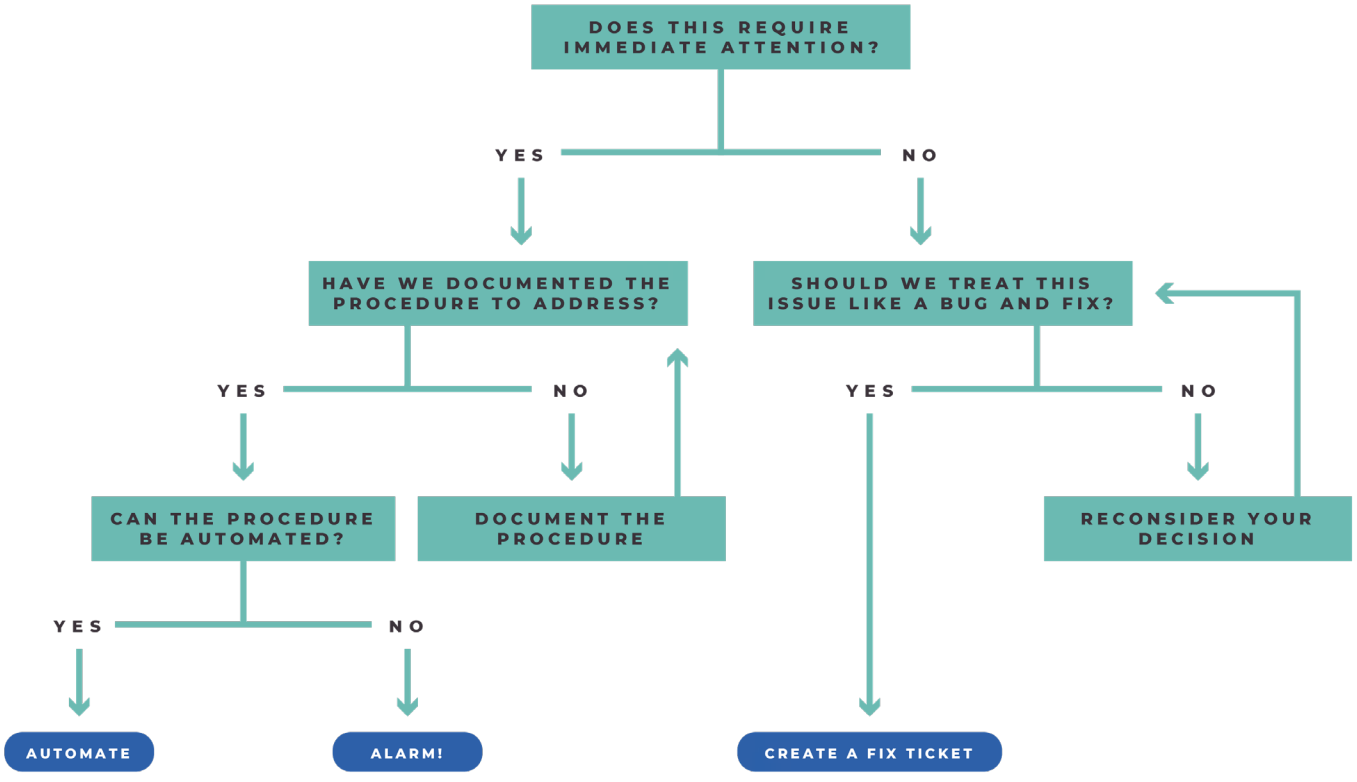
Alert (or alarm) fatigue is a massive problem faced by a wide variety of industries. Healthcare, aviation, and security — both physical and cyber — are three of the main sectors that have been forced to learn how to deal with an overwhelming amount of alarms, and the impact on their employees. IT and DevOps are no different.

Desensitization to alarms, or normalization, is what happens when you're over-exposed to something, like an alert.

Excessive alerts lead to employees who are prone to tolerate, normalize, and begin to ignore the alerts altogether.

Desensitization can be accelerated if many of the alerts generated are, in fact, false alarms. For instance, in the security industry, a survey found that 52% of alerts were false, while 64% were totally redundant. So, what to do about all these alerts? First off, we have to make sure the alerts are A. important, and B. handled in a systematic way. As the Google SRE book says:

"If it's important enough to bother a human, it should either ***require*** *immediate action (i.e., page) or be treated as a bug and entered into your bug-tracking system. Putting alerts into email and hoping that someone will read all of them and notice the important ones is the moral equivalent of piping them to* ***/dev/null:*** *they will eventually be ignored."*



Below are a few strategies we find helpful when it comes to reducing alert fatigue in SRE.

Important and Actionable Alerts // Create tiered alerts, since all alerts do not require immediate acknowledgement and resolution. If we reduce the amount of noise, the SRE team is able to remain focused on what actually matters now. Second, make sure that alerts are actionable. Vague alerts can lead to low productivity, and wear on the psyche of your team.

Reduce redundant alerts // A study found that for each reminder of the same alert, attention dropped 30%. More is not always better, and when it comes to alerts it's wise to find that balance between over-alerting and not alerting enough.

Unify information // Rather than having alerts originating from four or five different tools, which leads to extreme alert fatigue, consolidate the alerts into a single location — this will increase the likelihood that alerts are reviewed in a timely manner.

Don't be afraid to improve // Every month or so, check in. What has worked well? What could be done better? Listen to your team. Make changes. As long as everyone is working off the same standards (SLOs) and trying to achieve the same goals (maximum up-time), this process should be free of personal attacks.

CONSEQUENCES OF NOT IMPLEMENTING SRE

If your communications application goes unmonitored you are essentially relying on your customer base to inform you when problems arise. This can be embarrassing and, more importantly, result in a loss of users and revenue.

However, one of the benefits of working with a provider who has successfully implemented SRE is that they can help identify issues, even outside of their own systems. Below are a few recent issues that we were able to detect using the basic monitoring of the Four Golden Signals.

Latency Monitoring

An integral part of our own SRE culture at Voxology is for our "Platform Medics" to monitor alerts and alarms designed to highlight issues in our platform, with a downstream vendor, or even with a customer. One measurement we monitor is the latency of our callback requests to our customers endpoints. In one case in particular, a customer of ours (hosted in AWS) was under load. When things are functioning as they should be, their response to our callback requests would be nearly instantaneous. On this occasion, latency had increased dramatically which means people were answering phone calls and hearing nothing. The longer the latency, the more likely someone is to hang up. We alarmed on the abnormality and notified the customer immediately. It was revealed that they were not monitoring latency and were unaware of the issue. Thankfully, we were able to make some temporary adjustments on our end to help, as well as provide guidance for them to perma-fix.

Error Monitoring

Issues that impact your communications application can also arise outside of your own application -- for example you can probably recall a time when one of your providers (or your customer's providers) went down. In this scenario, our customer delivers thousands of inbound calls to their customer via a Voxology SIP Trunk. However, on this particular Saturday morning, their customer's destination was failing to receive the calls. When our alarms started metaphorically ringing, we contacted our customer and notified them of the problem before they, their customer, or their customer's service provider (who was failing) realized there was an issue. A single provider in the chain, who has embraced SRE, can help everyone.

CONCLUSION

Achieving optimal site and application reliability is possible by committing to a sound SRE culture. Choosing the right CPaaS provider can increase the capabilities and functionality of your communications infrastructure while enabling your overall SRE plan.

At Voxology, we are committed to working with our customers to help them go above and beyond their SLOs to deliver the best possible product to their customers. Don't wait for someone else to report a problem, define and implement a solid SRE plan and lead the way in site reliability engineering. Don't worry, we can help.

At Voxology, we are committed to working with our customers to help them go above and beyond their SLOs to deliver the best possible product to their customers.

WHAT'S NEXT?

1

Schedule a Call

Click here to schedule a 15 minute call with a Voxologist.

2

Collaborate With Our Team

Craft & execute a successful integration plan, embracing SRE culture.

3

Enjoy the Freedom of Voxology

Communications without the fuss.

SPEAK WITH A VOXOLOGIST