**Site Reliability Engineering**

SRE is defined as the protocols, practices and procedures that support a reliable product in production.

[SITE RELIABILITY ENGINEERING PRESENTATION](https://dta-images.discoverfinancial.com/articles/dojo_operate_site-reliability-engineering/static/DevOps_SRE.pptx)

**Overview**

**What is SRE?**

Site Reliability Engineering (SRE) is defined as the protocols, practices, and procedures that support a reliable product in production. This includes, but is not limited to, providing metrics and real time data about the product(s) in production. The main goal of SRE is to support highly reliable and scalable software systems.

SRE was coined by Ben Treynor, founder of Google's Site Reliability Team. SRE was designed to tackle the problem of massive silos in software development, there were many teams involved in an implementation of a product, which produced concentrated silos between the teams. This resulted in many headaches and misunderstandings as to what needed to be done to fix a known or unknown issue in production.

**Why SRE**

* bridges the gaps between development and operational execution by providing a set of standards for system reliability.
* develops the ability to measure, monitor, operate, and understand production systems in a resilient manner.
* sets measurable engineering goals mapped to service level objectives (SLOs) and enables monitoring and tracking of these quality of service (QoS) parameters.
* helps plan, identify, and deal with failures.
* thrives on automation and toil elimination.

**Read This First**

* [Testing Automation](https://dta.discoverfinancial.com/dojo_accept_test-automation)
* [CI](https://dta.discoverfinancial.com/dojo_build_continuous-integration)/[CD](https://dta.discoverfinancial.com/dojo_build_continuous-delivery)
* [DevSecOps](https://dta.discoverfinancial.com/dojo_release_devsecops)

**After Reading This, You Will Understand**

1. What is SRE, how is it different from DevOps?
2. What are the SRE principles?
3. What are Service Level Agreements (SLA(s)), Service Level Indicators (SLI(s)), and Service Level Objectives (SLO(s)) and how they drive SRE improvement.
4. Latency, Throughput, Saturation, and Errors.
5. Error Budgeting, and Operational Toil.
6. The need to eliminate the toil.

**SRE Fundamentals**

**How SRE Relates To DevOps**

Traditionally, IT teams consist of developers and operators. Developers are responsible for writing code for systems, and operators are responsible for ensuring that those systems operate reliably. Development and operations teams have conflicting priorities. Thus, DevOps emerged to help close gaps and break down silos between development and operations teams. DevOps is a philosophy, not a development methodology or technology. SRE is a practical way to implement DevOps philosophy, SRE consists of both technical and cultural practices.

**The Key Tenets Of SRE**

**Manage failures**

SRE thrives on errors and failures. SRE fosters the power of observability to maintain a reliable solution. Observability is the ability to monitor an application and quickly identify the root cause of a problem using observability engineering principles such as MELT (Metrics, Events, Logs, and Traces) and define metrics using the four golden signals (Latency, Traffic, Error Rate, and Saturation).

**Software reliability is linked to business objectives**

Most of the time, you hear SLA, SLO, and SLI. Business is concerned with Service Level Agreements (SLAs), which is the contractual agreement with the customer, whereas technologists are concerned with Service Level Indicators (SLIs), which are raw metrics. SLO sits in the middle between business and technologists. SLO is a target value of an SLI which falls within a defined threshold that meets the defined SLO. In a nutshell, SLIs drive SLOs which inform SLAs.

**Automation and Self-Healing**

Automation is the key to eliminate the toil. Automate the triage (troubleshooting), recovery (self-healing), ABCy (predictive ML models) and apply Chaos engineering to mature your system.

**Key Practices Of SRE**

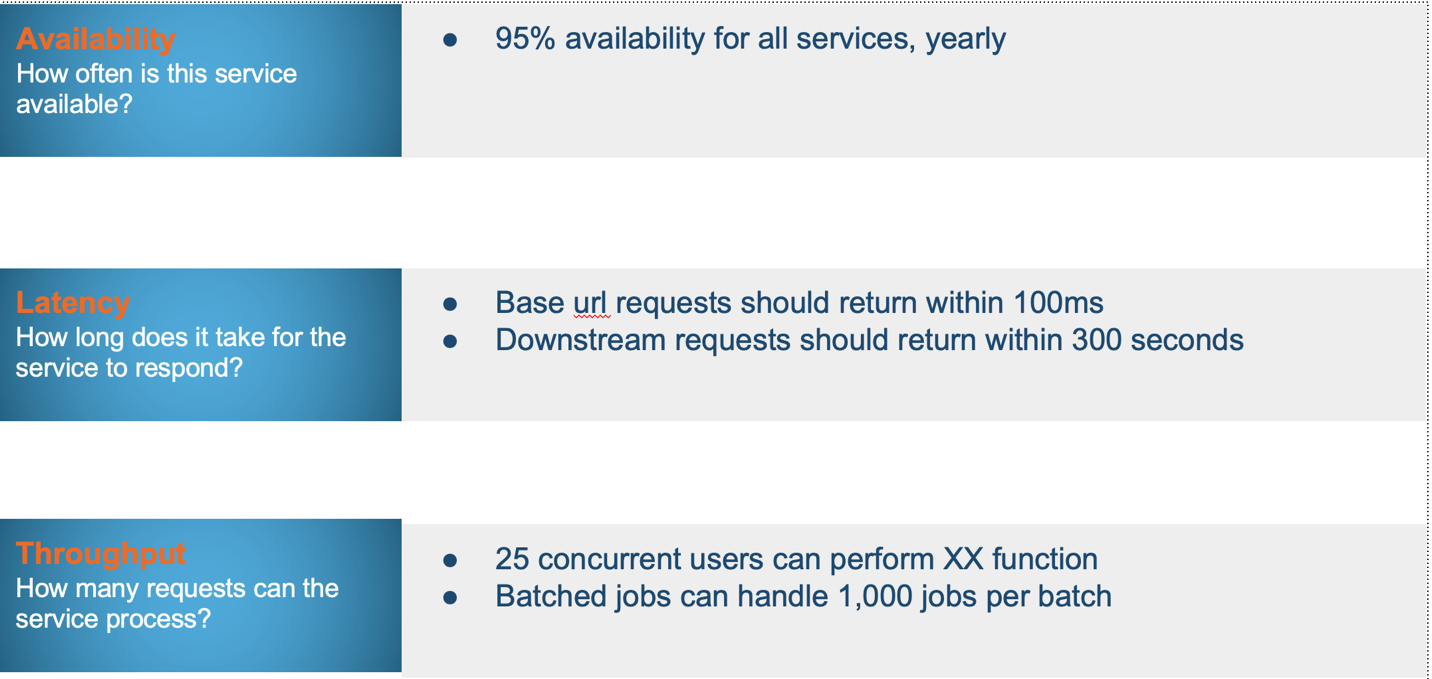
* SRE develops reusable best practices and backs critical technical infrastructure services to improve the production experience with a consistent focus on availability, latency, performance, and capacity.
* SRE's success is due in part to the development of procedures that teach people how to respond to emergency circumstances in a systematic and culturally appropriate manner.
* Monitoring and practical Alerting is fundamental to running a stable service.
* Being on-call is a critical duty that many operations and engineering teams must undertake in order to keep their services reliable and available.
* SREs are comfortable with failure. Failures are documented in blameless postmortems, which focus on systems and processes versus people. Blameless postmortems are detailed documentation of an incident or outage, its root cause, its impact, actions taken to resolve it, and follow-up actions to prevent its recurrence.
* 100% reliability is the wrong target because it slows the release of new features, which is what drives your business.
* Testing for reliability is the key. "If you haven't tried it, assume it's broken".
* SLOs and error budgets create shared responsibility and ownership between developers and SREs. The error budget is the difference between the target specified by the SLO and the service's actual availability for a given period. If you blew up your error budget, you should stop deploying new features, gracefully handling overload conditions, and focus on getting back into a healthy error budget.

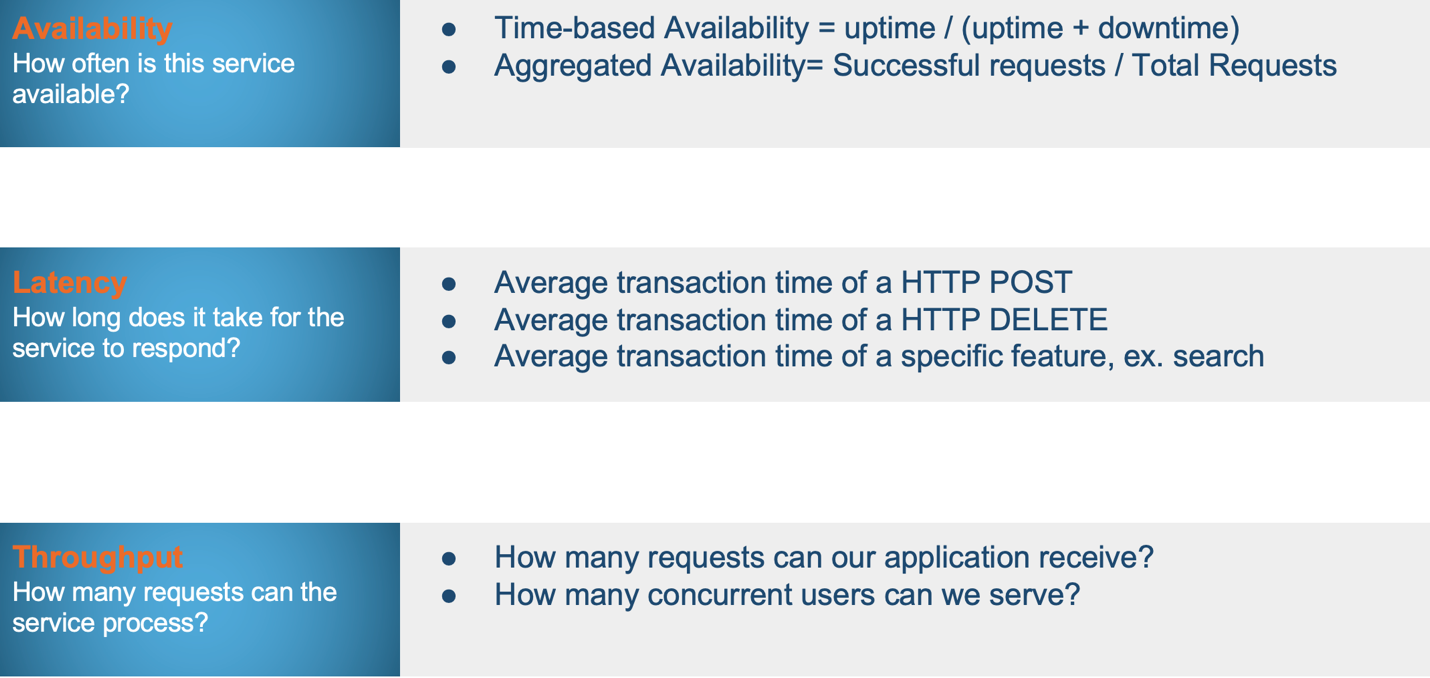
**Examples**

SLA,SLO, and SLI should not be arbitrary. They should be agreed upon by Customer, Product, and Engineering.

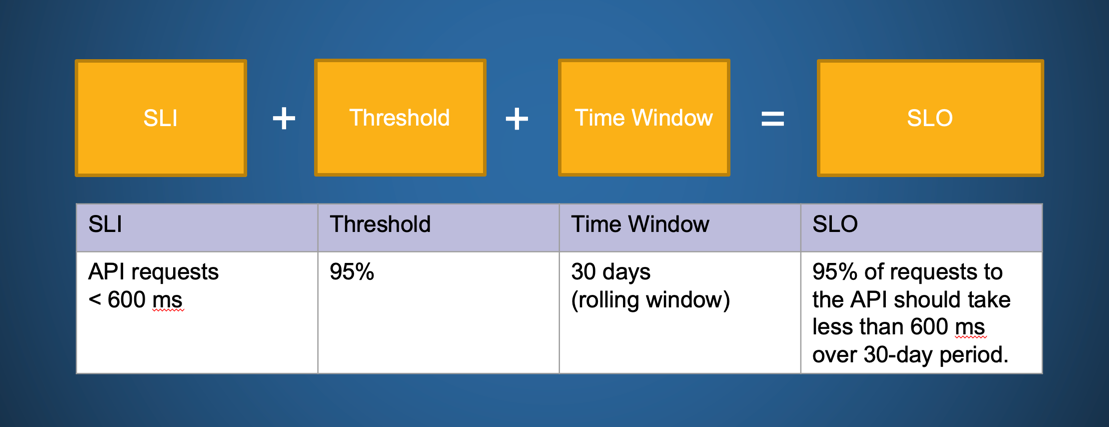
**SLA Example**

* [Google Cloud Functions Service Level Agreement](https://cloud.google.com/functions/sla)

**SLO Example** 

**SLI Example** 

**SLI vs. SLO**



**Observability**

Observability is the ability to monitor an application and quickly identify the root cause of a problem using observability engineering principles such as MELT (Metrics, Events, Logs, and Traces) and define metrics using the four golden signals (Latency, Traffic, Error Rate, and Saturation). SREs can use Observability to provide highly reliable applications and software at scale, with a focus on a real-time view of the performance of the digital assets. But **how does observability differ from monitoring**?

**Observability .vs Monitoring**

Monitoring alerts you when something is wrong. On the other hand, observability helps find what is wrong and why it happened. Observability can be achieved by expanding and extending monitoring processes to gain deeper insights into complex systems. Monitoring tools typically focus solely on data collection and, sometimes, generating alerts based on anomalies or pre-configured triggers. Observability combines data from various systems to provide meaningful context for each issue identified by monitoring data.

**The Three Pillars of Observability**

**Metrics:** Metrics are numerical indications of data that are frequently calculated over time (for example, the number of failed requests). These value counts are derived primarily from a system's performance and can come from various sources such as hosts, services, cloud platforms, and so on. Metrics also contain information about Service Level Indicators (SLIs), which shed light on memory and power usage. Because SLIs are frequently generated, they are efficient and dependable.

**Event Logs:** Logs are computer-generated structured, plain, or binary text. These are time-stamped records that provide a precise understanding of discrete events that occur at a specific time. Event logs help detect unpredictability in distributed system components.

**Traces:** Traces keep track of a request or transaction activity as it moves through an application. These are critical because they provide context for the previously mentioned data. Traces, for example, can help provide insightful information about which metric is more important or which log is more relevant to the specific transaction ID.

**Error Budget**

Error budget is an acceptable downtime a service can experience without contractual consequences.

According to Google's SRE handbook, an error budget is 1 minus the SLO of the service.

A 99.9% SLO service has a 0.1% error budget. If our service receives 1,000,000 requests in four weeks, a 99.9% availability SLO gives us a budget of 1,000 errors over that period.

Service Reliability Engineers uses Error Budget to balance service reliability with the pace of innovation. When prioritizing between feature development work and development work for stability, error budget plays a critical role on deciding where the efforts should be spent. When the error budget is within the threshold team can focus on enabling new feature work if not team should work on stability of the application.

| Availability | Downtime per day | Downtime per week | Downtime per year |
| --- | --- | --- | --- |
| 90% | 144 minutes | 16.8 hours | 36.5 days |
| 95% | 72 minutes | 8.4 hours | 15.25 days |
| 99% | 14.4 minutes | 1.68 hours | 3.65 days |
| 99.50% | 7.2 minutes | 0.84 hours | 1.825 days |
| 99.90% | 1.44 minutes | 0.168 hours | 0.365 days |
| 99.99% | 0.144 minutes | 0.0168 hours | 0.0365 days |

Let's assume our SLO is 80% and see how the error budget will be for the availabilities captured in the above table.

| Availability | SLO | Error Budget | Error budget per day |
| --- | --- | --- | --- |
| 90% | 80% | 10% | 2.4 hours |
| 95% | 80% | 15% | 3.6 hours |
| 99% | 80% | 19% | 4.56 hours |
| 99.50% | 80% | 19.5% | 4.68 hours |
| 99.90% | 80% | 19.9% | 4.77 hours |
| 99.99% | 80% | 20% | 4.79 hours |

Here is an example of negative error budget for the same SLO (80%) but the availability is 70%. When the error budget is negative team should work on stability of the application.

| Availability | SLO | Error Budget | Error budget per day |
| --- | --- | --- | --- |
| 70% | 80% | -10% | -2.4 hours |

**Apply SRE In Your Team**

**Uncover today**

* This is a collaborative effort between team members to assess the maturity level of SRE.
* Understand the existing tools and the level of scanning, monitoring, and alerting.
* Understand today's remediation process and discuss it.
* Identify points of failure in your systems from historical findings.
* Explore potential production problems.
* Explore your code resiliency.

**Build better tomorrow**

* Define SRE practices and foster SRE culture in the team.
* Discuss with customer, product, and engineering SLA, SLO, and SLI.
* Establish your monitoring expectations to meet SLO. Use tools like App Dynamic and Grafana for monitoring.
* Define alert levels and priorities: individual/system must act right now, or must act but not immediately.
* Make code resilient to failure by using resiliency patterns such as retry, circuit breaker, and bulkhead patterns.
* Automate SRE processes and keep improving.

**SRE Cheat Sheet**

[ABC-SRE Cheat Sheet](https://github.discoverfinancial.com/pages/chapter-org/SRE-Automation-Chapter/SRE/docs/Frameworks/site/SRE_Glossary/)

**Maturity Model**

Maturity Model helps you understand where your team stands when it comes to practicing SRE.

**Do You**

* have a team capable of producing audit trails and gain access to logging data for report generation?
* have this process streamlined and easy to follow
* use tools that are being used actively and managed by someone on the team.
* keep the tools up to date?
* identify an abnormal pattern of activity from a normal pattern?
* maintain a good relationship between operations team and the Development Team?

**What Does Great Look Like?**

Teams understand and can explain in great details what the followings terms are and how they are related to SRE:

* SLA, SLI, SLO, Error Budgeting
* The Four Golden Signals
* Give examples of tools and software suites that help automate some of the SRE practices

**Walk**

Monitoring and reporting are manually being performed.

**Run**

Monitoring and reporting are manually being performed. Scanning is performed regularly but does not report to the backlog. Latency, Throughput, Saturation, Yield, Availability, and Errors can be measured(manually or automatically).

**Fly**

Monitoring and reporting are automatically being performed. Scanning is performed regularly reported to the backlog. Latency, Throughput, Saturation, Yield, Availability, and Errors can be measured automatically.

**Additional Resources**

* [Site Reliability Engineering: How Google Runs Production Systems, by Betsy Beyer, Chris Jones, Jennifer Petoff, Niall Richard Murphy - O'Reilly Media, Incorporated, 2016](https://sre.google/sre-book/table-of-contents/)
* [The Site Reliability Workbook: Practical ways to implement SRE](https://sre.google/workbook/table-of-contents/)