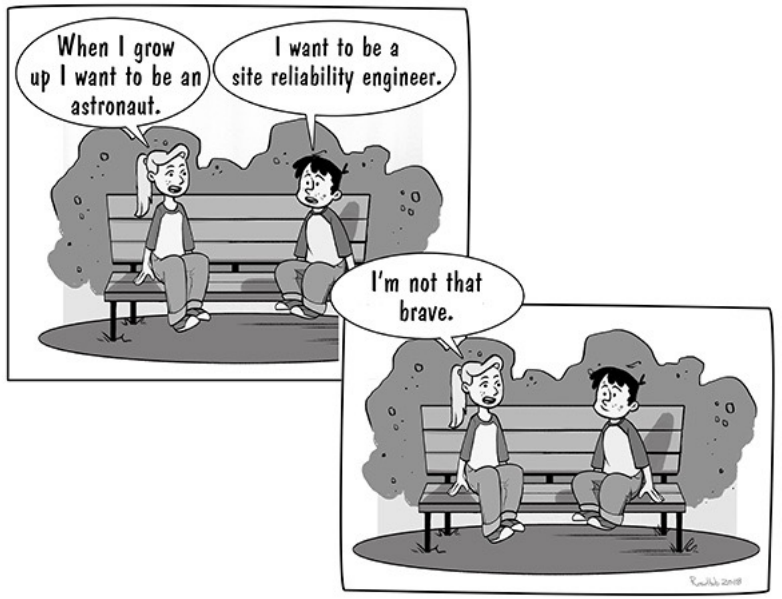
Site Reliability Engineering (SRE)

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University




EVERYTHING OKAY?
JUST WAITING FOR WHATEVER
IS ABOUT TO GO WRONG.

Cost of System

- Software engineering has this in common with having children: the labor *before* the birth is painful and difficult, but the labor *after* the birth is where you spend most of your effort.
- Software engineering as a discipline spends much more time talking about the first period as opposed to second, despite estimates that 40-90% of the total costs of a system are incurred after birth.
- The popular industry model that conceives of deployed, operational software as being "stabilized" in production, and therefore needing much less attention from software engineers, is wrong.

> *Hope is not a strategy."*

Traditional SRE saying

Traditional Systems Administrator Role

- This systems administrator, or sysadmin, approach involves assembling existing software components and deploying them to work together to produce a service.
- Sysadmins are then tasked with running the service and responding to events and updates as they occur.

The Conflict

- The development teams want to launch new features and see them adopted by users as quickly as possible.
- The operations teams want to make sure the service doesn't break while they are holding the pager. Because most outages are caused by some kind of change—a new configuration, a new feature launch, or a new type of user traffic
- the two teams' goals are fundamentally in tension.

What is Site Reliability Engineering?

- **Site reliability engineering (SRE)** is a discipline that incorporates aspects of software engineering and applies that to operations whose goals are to create ultra-scalable and highly reliable software systems.
- Ben Treynor, founder of Google's Site Reliability Team defines SRE as *"what happens when a software engineer is tasked with what used to be called operations."*

DevOps or SRE?

- The term “DevOps” emerged in industry in late 2008 and is still in a state of flux.
- Its core principles—involvement of the IT function in each phase of a system’s design and development, heavy reliance on automation versus human effort, the application of engineering practices and tools to operations tasks—are consistent with many of SRE’s principles and practices.
- One could view DevOps as a generalization of several core SRE principles to a wider range of organizations, management structures, and personnel.
- One could equivalently view SRE as a specific implementation of DevOps with some idiosyncratic extensions.

Tenets of SRE

In general, an SRE team for a service is responsible for the following:

- Availability
- Latency
- Performance
- Efficiency
- Change management
- Monitoring
- Emergency response
- Capacity planning

Eliminating Toil

> *If a human operator needs to touch your system during normal operations, you have a bug. The definition of normal changes as your systems grow.*
>
> Carla Geisser, Google SRE

Toil is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows.

Error Budget

- The error budget stems from the observation that 100% is the wrong reliability target for basically everything (pacemakers and anti-lock brakes being notable exceptions).
- 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.

Error Budget (contd.)

- If 100% is the wrong reliability target for a system, what, then, is the right reliability target for the system? This actually isn't a technical question at all—it's a product question, which should take the following considerations into account:
    1. What level of availability will the users be happy with, given how they use the product?
    2. What alternatives are available to users who are dissatisfied with the product's availability?
    3. What happens to users' usage of the product at different availability levels?
- The business or the product must establish the system's availability target. Once that target is established, the error budget is one minus the availability target. A service that's 99.99% available is 0.01% unavailable. That permitted 0.01% unavailability is the service's **error budget**.

Embracing Risk

- Reliability isn’t linear in cost. It can easily cost 100x more to get one additional increment of reliability.
  - Cost associated with redundant equipment.
  - Cost of building out features for reliability as opposed to “normal” features.
  - Goal: make systems reliable enough, but not too reliable!
- Example: if a user is on a smartphone with 99% reliability, they can’t tell the difference between 99.99% and 99.999% reliability

Measuring Service Risk

- For most services, the most straightforward way of representing risk tolerance is in terms of the acceptable level of unplanned downtime.
- Unplanned downtime is captured by the desired level of service availability, usually expressed in terms of the number of "nines" we would like to provide: **99.9%**, **99.99%**, or **99.999%** availability.
- Each additional nine corresponds to an order of magnitude improvement toward **100%** availability.

Time-based availability

- Using this formula over the period of a year, we can calculate the acceptable number of minutes of downtime to reach a given number of nines of availability.
- A time-based metric for availability is usually not meaningful when we are looking across globally distributed services.

$$\text{availability} = \frac{\text{uptime}}{(\text{uptime} + \text{downtime})}$$

Aggregate Availability

- Define availability in terms of the request success rate.
- For example, a system that serves 2.5M requests in a day with a daily availability target of 99.99% can serve up to 250 errors and still hit its target for that given day.
- *Not all requests are equal: failing a new user sign-up request is different from failing a request polling for new email in the background.*

$$\text{availability} = \frac{\text{successful requests}}{\text{total requests}}$$

Risk Tolerance of Services

- SREs work with product owners to translate business objectives into explicit objectives.

Service Level Objectives - Terminology

- Indicators - An SLI is a **service level indicator**—a carefully defined quantitative measure of some aspect of the level of service that is provided. Example, Consider request latency—how long it takes to return a response to a request—as a key SLI. Other common SLIs include the error rate, often expressed as a fraction of all requests received, and system throughput, typically measured in requests per second.
- Objectives - An SLO is a **service level objective**: a target value or range of values for a service level that is measured by an SLI. A natural structure for SLOs is thus **SLI ≤ target**, or **lower bound ≤ SLI ≤ upper bound**. For example, we might decide that we will return Shakespeare search results "quickly," adopting an SLO that our average search request latency should be less than 100 milliseconds.
- Agreements - SLAs are service level agreements: an explicit or implicit contract with your users that includes consequences of meeting (or missing) the SLOs they contain. The consequences are most easily recognized when they are financial—a rebate or a penalty—but they can take other forms. An easy way to tell the difference between an SLO and an SLA is to ask "what happens if the SLOs aren’t met?": if there is no explicit consequence, then you are almost certainly looking at an SLO.16

Collecting Indicators

- Many indicator metrics are most naturally gathered on the server side, using a monitoring system.
- Some indicators must be collected on client side.

> *May the queries flow, and the pager stay silent."*
>
> Traditional SRE blessing

Practical Alerting from Time-Series Data

- Monitoring, is fundamental to running a stable service.
- Monitoring enables service owners to make rational decisions about the impact of changes to the service, apply the scientific method to incident response, and of course ensure their reason for existence: to measure the service's alignment with business goals.

Emergency Response

- Things break; that's life.
- Few of us naturally respond well during an emergency.
- A proper response takes preparation and periodic, pertinent, hands-on training.
- The way you respond to a situation is almost as important as the remedy itself.

- BA038 777 Crash ATC Recording

Postmortem: Learning from Failure

- Learn from the Past. Don’t Repeat It.
- A postmortem is a written record of an incident, its impact, the actions taken to mitigate or resolve it, the root cause(s), and the follow-up actions to prevent the incident from recurring.
- Avoid Blame and Keep It Constructive
- Collaborate and Share Knowledge
- No Postmortem Left Unreviewed

Additional Resources

See Lecture Page