Google

SRE at Google

Reliability for Planet-Scale Infrastructure

Cameron Tuckerman-Lee, Site Reliability Engineer
tuckerman@ --- cjt@google.com

Agenda

1

DevOps & SRE

What is the connection between operations and reliability?
How does SRE relate to DevOps?

2

Reducing Toil

How does approaching reliability work from an engineering perspective necessitate reducing toilsome work?

3

Data Driven Operations

How do SRE teams use Service Level Objectives (SLOs) to ensure a healthy relationship with their partner teams?

4

Best Practices

What are some of the best practices employed by SREs at Google that are broadly applicable across the industry?

DevOps & SRE

DevOps & SRE

DevOps

- A **philosophy and framework** for tearing down walls between dev/ops/sec/etc
- **Many forms**, some of which **combine the roles** of developer and operator.
- **Bottom-up**: *How do we empower **individuals** to work across traditional boundaries to improve **all facets** of a product?*

SRE

- A **job, organization, and a mindset** for managing and operating large systems.
- **One approach to DevOps** with a team that has **distinct skills and roles**.
- **Top-down**: *How do we create and hire into an **organization** that is empowered to ensure **reliability and uptime** of a product?*

Key Metrics of DevOps and SRE

DevOps	SRE
• Lead Time	• Latency
• Process Time	• Error Rate
• Percentage Complete / Accurate	• Throughput
	• Availability

Site Reliability Engineering is a **specialized approach to DevOps** focused on the reliability of Google's planet-scale infrastructure.

public class **SRE** implements **DevOps** {

SRE is a implementation of the DevOps interface†

- **Dedicated engineers** focusing on reliability and operations working together with development teams
- **Software and System Engineering** approaches to problem solving
- **Artificial scarcity** to encourage automation and sublinear team growth
- **Shared philosophy** and best practices across the company e.g. common tooling, SLOs, error budgets

}

† Fong-Jones, Liz. Vargo, Seth. What's the Difference Between DevOps and SRE? < https://youtu.be/uTEL8Ff1Zvk >

What is SRE?

Fundamentally, it's what happens when you ask a software engineer to design an operations function.

Ben Treynor, VP of Engineering

SRE Organization Structure

- Product SREs, embedded alongside dev teams
  e.g. Search, Photos
- Technical Infrastructure SREs, embedded alongside dev teams
  e.g. Bigtable, Spanner, Borg (Cluster Orchestration)
- Shared Platform SREs, similar to a dev team focused on SRE tooling
  e.g. Monitoring, Incident Management Tools, Rollouts

All three kinds of SRE teams are in the ***same organization****!*

SRE Organization Structure

By being in the same organization, teams embrace a common philosophical approach to reliability engineering. SREs...

- Work in the **intersection of software and systems engineering**
- Actively **foster diversity** ⇒ creativity and new ideas
- Are intentionally a **scarce resource** to ensure efficiency
- **Partner closely** with our dev teams

It also means we can encourage mobility within the SRE organization, ensuring novel ideas and improvements are shared throughout the company.

SRE Motto: Hope is not a strategy.

Eliminating Toil

What exactly is toil?

toil |tɔɪl| *n.*

1. Repetitive, manual work that is associated with some operation. Scales linearly with the size of a service and doesn’t provide lasting value.

Examples of Toil

- Manually reviewing monitoring and logging systems every day to make sure your system is still running
- Deploying software to many machines via **`scp`**
- Turning up a service in a new availability-zone or region requires copy-pasting configuration, manually migrating data
- Common tasks are driven by tickets: users request “thing 1” so they file a ticket asking an operator to run some script **`do_thing_1.sh`**

Why is toil bad?

- Systems that are very toilsome are **expensive**! They require scaling headcount as fast (or sometimes faster 😱) than the service.
- **Toil accumulates**—over time your team has less and less time to take on more important project work.
- Most incidents are a result of changes to running services. Toil **increases that risk of regressions** due to error-prone, manual changes.
- Last, but not least, **toil isn't exciting**! Excessive toil leads to unhappiness and burnout. Employees want to be challenged and create value.

Measure Toil!

Don’t just rely on your intuition; humans are notoriously bad at estimating how much time they spend on certain tasks. Just like you should profile your code before you work on performance, measure your toil before working on efficiency!

- You can often use the same tools you use for tracking project work, e.g. tickets/JIRA/etc
- Helps identify hot-spotting toil on certain individuals (prevent burnout)
- Identifies opportunities for high impact toil reduction which can result in reduced costs and increased team efficiency.

Patterns for Toil Reduction

- Use Third Party/Open Source where possible!
- Create standards for high risk actions like rollouts, e.g. standardize on k8s, Terraform, etc and build up patterns.
- Create a standard place for your team to "put automation" like a team CLI tool, Jenkins server, or Chef cookbook. Expose rpc/API endpoints/library so other teams can compose automation!
- **Where possible, don't accept toil in the first place!**

Data Driven Operations

The Problem of Reliability vs. Velocity

A) Developers and product teams want to release new features as fast as possible and aren't responsible for maintaining the service.

B) Operators want their services to be reliable/available for customers.

C) Most incidents/regressions are the result of changes to a system (e.g. new rollout, configuration change)

A + B + C = Operators decide **never** to release again!

Could we measure the riskiness of a given change? No, very hard in practice.

SLIs, SLOs, SLAs

Service Level Indicator	quantitative measure of how a service is operating/performing e.g. error rate
Service Level Objective	target value/range for a specific service SLI measured over some time period e.g. p99 latency in 5 min window < 200 ms
Service Level Agreement	contract to meet a specific SLO or provide the user of the service with compensation e.g. 99.9% availability in any month or ½ off

Setting SLOs

- Set SLOs for the SLIs that describe your **users' needs**
- Choose thresholds that:
  - If *just barely met* would mean your **users are still happy**!
  - Are possible for your team to meet!
- Use consistent SLIs, time windows, terminology across the company so users understand what they are getting.

Error Budgets

Now that our service has SLOs defined, SREs don't have to make judgement calls on whether releases are safe!

➔ Under Error Budget? New features are released!

➔ Over Error Budget? Only reliability/perf/monitoring improvements!

The decision **is not** personal, but instead is based on the **needs and expectations of the service's users**

Product teams can "spend" their remaining error budget on velocity.

Best Practices

SLO Alerts

Alerting on low level metrics (cpu/ram/networking) is noisy, and pager fatigue can lead to real problems (burnout, ignoring pages, etc).

We should alert when we are, or at risk of, breaking our promises to our users (i.e. spending our error budget).

➔ Spending your error budget slowly? First, wait and see if it passes. If it continues, then **Page SRE!**

➔ Spending your error budget quickly? **Page SRE!** You are at risk of violating your SLOs, and your users built systems that expect you to meet your promise!

Corollary: You shouldn't depend on a service's behavior if they don't have an SLO for it!

Spend Your Error Budgets!

100% uptime is explicitly **not** a goal!

- Users (internal and external) will rely on your past performance, even if you exceed your SLO (implied SLO).
- Users won't be able to handle errors/unavailability (implement failover, retry, degraded experience)
- What happens if there is a real outage?

At Google, teams will regularly **spend their extra error budgets** in controlled tests either as part of a company wide Disaster Recovery Testing (DiRT) exercise or in mini drills (dust).

Blameless Postmortems

At large scales of machines, LOC, or number of humans: **things break**!

- Engineers shouldn't fear punishment or retribution. They will feel unsafe in their jobs and avoid taking necessary, calculated risks†.
- Incidents help us enumerate opportunities to improve our systems

Instead of focusing on assigning blame or minimizing problems:

- Set clear criteria for incidents and requiring postmortems
- Prioritize understanding root cause (5 Whys)
- Create action plans to solve and follow through!

†: Allspaw, John. *Blameless PostMortems and a Just Culture*. Code as Craft.

Closing Thoughts

- Many of these practices (e.g. Error Budgets, Blameless Postmortems) have been adopted à la carte by companies big and small, on-prem and cloud. Start small and make incremental improvements!
- For more, the book *Site Reliability Engineering: How Google Runs Production Systems* is available in print from O'Reilly and an online version is available now at landing.google.com/sre
- If you have thoughts/questions/comments to share, let's chat!

References, Inspirations, & Good Reads

Allspaw, John. *Blameless PostMortems and a Just Culture*. Code as Craft.

Beyer, Betsy. Jones, Chris. Petroff, Jennifer. *Site Reliability Engineering: How Google Runs Production Systems.*

Fong-Jones, Liz. Vargo, Seth. *What's the Difference Between DevOps and SRE?*

Fowler, Susan. *Who's On Call?*

Goldfluss, Alice. *Scalable Meatfrastructure: Building Stable DevOps Teams. USENIX.*

Treynor, Ben. *Keys to SRE*. SREcon14.

谢谢