Google SRE

Jamie Wilkinson
DOTC Melbourne 2018
What is Site Reliability Engineering?

Keep the site up—whatever it takes

- "Site" == google.com.
- Site unavailable? Our problem, whatever the reason.
Why Reliability?

It's the number one feature
Do you prefer Gmail 2016?
Or Gmail 500?
Reliability is easy to take for granted

- It’s the absence of errors
- By the time the service becomes unstable, a number of things have failed
- You need to work at reliability all the time
  - Not just when everything’s on fire
What is Site Reliability Engineering (SRE)?

- Solve production problems with software. It’s all just software.
- Work at a Large Scale
  - Many services, lots of data, many machines.
  - Not so many people. **People must scale sub-linearly to services.**
- Balance competing demands
  - Improve availability and reliability.
  - Improve efficiency.
  - Adopt new services
What is Site Reliability Engineering (SRE)?

- Not a purely operational, reactive team.
- Responsible for improving services along important dimensions:
  - Availability
  - Performance (both latency and efficiency)
  - Manageability
- Maximise rate at which new services, features, etc. can be delivered to users.
- And we undertake on-call responsibilities as well...
What do SREs do?

- Consult with our Development colleagues.
  - Design reviews for scale. Spot flawed design patterns.
  - Encourage the use of best practices and proven methodologies.
  - Collaborate on implementation.
- Develop software...
  - that directly benefits our users.
  - that improves the reliability of our services.
  - that improves the efficiency of our services.
  - that improves the manageability of our services.
What do SREs do?

- On-board new services
  - Consult with Development teams to bring services up to an SRE-supportable level.
  - Transition the service from Development-supported to SRE.
- On-call
  - Maintain the Service Level Objective (SLO).
  - Typically ~6w between shifts, various patterns.
  - Typically daylight hours only.
  - Fix issues whose resolution has not yet been automated.
  - Control changes to protect customer experience.
What do SREs do?

Use exactly the same tools and processes as Product developers
SRE Organizational Structure

- The SRE Organization is independent of business units/divisions.
- SRE teams are organized around a single service or a collection of related services or technologies:
  - Search
  - Gmail
  - Maps
  - Photos
  - Storage
  - .. etc
Services are jointly owned by product development and SRE.
Both can choose to extend or end SRE work on the service.
SREs are in short supply and can easily work on other services.
Staffing budget comes from the Product.
How is it different from Software Engineering?

- Typically not feature-driven workload.
- Typically not front-end related development.
- Typically greater choice in what you work on day-to-day.
Dev and Ops

Don't Dev and Ops always fight?

Dev wants to...
- ...roll out features fast
- ...and see them widely adopted

Ops wants...
- ...stability so they don't get paged
- ...on their shift
Development has its own lingo

- Development releases new products, but also:
  - Feature additions
  - Flag flips
  - UI changes
  - 20% experiments
And just to make it harder...

Information asymmetry is extreme.

The team which knows the least about the code has the strongest incentive to object to it launching.
Is conflict inevitable?

No!

- SRE doesn’t attempt to assess launch risk,
- or avoid all outages
- or set release policy.
Then what?

- Error budgets!
- But you first need a Service Level Objective
SLA, SLO, and SLI?

- **SLI → Indicator**: a measurement
  - response latency over 10 minutes
  - error rates over 10 minutes
- **SLO → Objective**: a goal
  - 99.9th percentile below 5ms
  - less than 1% errors
- **SLA → Agreement**: economic incentives
  - or we get paged
100% SLA
Google doesn't run at 100% SLA

- Impossible to achieve
- Very expensive
Error Budget

- 1 - SLO
- Example
  - SLO: 99.9%
  - Error budget: 100% - 99.9% = 0.1%
  - For a 1 billion query/month service
    - 1 million "errors" to “spend”
What do you spend your budget on?

- Change is #1 cause of outages
- Launches are big sources of change
- Solution: Spend error budget on launches!
The rule

- If service is within SLO, launch away.
  - Clearly product team is doing a good job.
- If service is not within SLO, roll it back.
  - Until you earn back enough error budget.
Two nice features of Error Budgets

1. Removes major source of conflict.
   a. It’s measureable, instead of guesswork

2. Product teams self-police because the individuals share the budget.

3. “Testing in Production”
You can throw people at a badly-functioning system and keep it alive via manual labor.

That job isn't fun.
But it’s sooo tempting?

“What I see is all there is.”

“What working on reliability slows feature dev”

➔ It’s another incentives problem.
➔ We have 6 fixes, and we do all of them.
Fix 1: Common Staffing Pool

- One more SRE = One less developer
- The more operations work...
  - ...the fewer features.
- Self-regulating systems win!
Fix 2: SRE hires programmers

- They speak the same language as product.
- They know what a computer can do.
- They get bored easily, so automation gets built.
Fix 3: 50% cap on Ops work

- If you succeed, work scales with traffic.
- Projects reduce work/traffic ratio.
- Leave enough time for project work
Fix 4: Keep Product in the rotation

- “What I see is all there is”
- Make sure the product team sees the product in action.
- Nothing builds consensus on bug priorities like a couple sleepless nights in a row.
Fix 5: Speaking of Dev and Ops work...

Excess operations load (tickets, oncall, etc) always gets assigned to the product team.
Fix 6: SRE Portability

- No requirement to stick with any project; in fact,
  - ...No requirement to stick with SRE
- Build it and they will come;
  - ...bust it, and they will leave.
- The threat is rarely executed, but it is powerful.
Death, taxes, and outages...

- SLO < 100% means that there will be outages.
  - This is fine.
- Two goals for each outage:
  - Minimize impact
  - Prevent recurrence
Minimize Damage

- Make the outage as short as possible:
  - No NOC - fast detection
  - Good diagnostic information - fast diagnosis
  - Practice, practice, practice - fast remediation
A word on practice...

“Operational readiness drills” aren’t cool.

You know what’s cool?

Wheel of Misfortune!

One of our most popular SRE events.
Wheel of Misfortune

Roll 1d20

- 1-4: Sharks with lasers
- 5-9: Sharknado
- 10-15: Sharkvalanch
- 16-20: Bear Cavalry
Prevent recurrence

1. Handle the event
2. Write the post-mortem
3. Reset

Corollary: max of 2 events per oncall shift

Corollary: minimum group size
Post-mortem philosophy

- Post-mortems are blameless
- Assume people are intelligent, well-intentioned
- Focus on process and technology

- Create a timeline
- Get all the facts
- Create bugs for all follow-up work
SRE Recap

- Reliability is your #1 Feature
- Hire programmers, and give them agency
- Have an SLO for the service, measure, use as launch criteria
- Common staffing pool; every SRE costs a Product Dev, and the converse
- Cap SRE operational load at 50%, by changing SLOs or escalation
- Oncall handles 1-2 events/shift, by changing SLOs or escalation
- Practice, practice, practice. Done right, it’s fun.
- Post-mortems focus on process and technology
O'Reilly Book

Site Reliability Engineering

How Google Runs Production Systems

Google's SRE Website

https://www.google.com/sre

- More resources
- Articles
- Videos