Getting Started with Site Reliability Engineering

Jennifer Petoff, Google Ireland
Twitter: @jennski
Hello
my name is

Jennifer Petoff
(aka Dr. J)
Google Ireland

12 years at Google

wide range of project management and training/education experience:

- University Programs
- DCLK Publisher Training Team
- AdWords Global Customer Service
- Site Reliability Engineering
Jennifer Petoff (aka Dr. J)
Google Ireland

Senior Program Manager in SRE for >5 years

- Lead and Global Program Manager for SRE EDU
- Co-editor of the original SRE Book
Jennifer Petoff (aka Dr. J)
Google Ireland

Fun Facts
- PhD in Chemistry
- Part-time Travel Blogger at Sidewalk Safari
Software engineering as a discipline focuses on designing and building rather than operating and maintaining, despite estimates that 40%\(^1\) to 90%\(^2\) of the total costs are incurred after launch.


Incentives aren't aligned.

Developers
Agility

Operators
Stability
DevOps is a set of practices, guidelines and culture designed to break down silos in IT development, operations, architecture, networking and security.

Site Reliability Engineering is a set of practices we've found to work, some beliefs that animate those practices, and a job role.
Reducing product lifecycle friction

- Agile solves this
- DevOps solves this
What is Site Reliability Engineering?

- Originated at Google in 2003
- Framework for operating large scale systems reliably
- "SRE is what happens when you ask a software engineer to design an operations function"
- Focus on running systems in production
Site Reliability Engineering Principles

1. SRE needs Service Level Objectives, with consequences.
2. SREs must have time to make tomorrow better than today.
3. SRE teams have the ability to regulate their workload.
4. Failure is an opportunity to improve.
Product lifecycle

Business Process

Concept → Business → Development → Operations → Market

Site Reliability Engineering solves this problem
But getting started can feel daunting...
Service Level Objectives
What is a Service Level Objective?

- Goal for how well the system should operate
- Tracks the customer experience
  - SLOs met = 😍😊😄 Customers
  - 😞 😢 😡 Customers = SLOs not met
Example SLOs

- 99.99% of HTTP requests per month succeed with 200 OK
- 90% of HTTP requests returned in under 300ms
- 99% of log entries processed in under 5 minutes.
But What About SLAs?

- Service Level Agreements = contractual guarantees
- SLAs met != 😍😊😊 Customers
What Next?

- You could implement SLOs today for your application, but this is only a foundation.
- You need consequences.
Error Budget Policy
How Reliable Do You Want To Be?
“Anything that can go wrong will go wrong.

Murphy's Law
Anything that can go wrong, will...

Finagle's Law of Dynamic Negatives
Anything that can go wrong, will...

...at the worst possible moment.

Finagle's Law of Dynamic Negatives
100% is the wrong reliability target for basically everything.

Benjamin Treynor Sloss
Vice President of 24x7 Engineering, Google
SRE is About Balance

Reliability
Engineering Time
Development Velocity
Cost
So we introduce a budget
Error Budgets

- Gap between perfect reliability and our SLO.
- This is a budget to be spent.
- Given an uptime SLO of 99.9%, after a 20 minute outage you still have 23 minutes of budget remaining for the month!
What you agree to do when the application exceeds its error budget.

This is not "pay $$$"

Must be something that will visibly improve reliability.
Error Budget Policy Examples

Until the application is again meeting its SLO and has some Error Budget:

- "No new feature launches allowed."
- "Sprint planning may only pull Postmortem Action Items from the backlog."
- "Software Development Team must meet with SRE Team daily to outline their improvements"
SRE Principle #1

SRE needs Service Level Objectives with Consequences.
SRE Principle #1

- Even without hiring a single SRE, you can have an Error Budget Policy.
- Lever you can use to keep your customers from experiencing pain and sadness.
- You can implement this today: measure, account and act.
Making Tomorrow Better Than Today
Making Tomorrow Better Than Today

- SLOs and Error Budgets are the first step.
- The next step is staffing an SRE role...
- ...endowed with real responsibility.
Your First SRE

- Defines and refines Service Level Objectives.
- Enacts the Error Budget Policy when necessary.
- Makes sure that the application meets the reliability expectations of its users.
Toil

- A **bounded** part of the role.
- Recommend less than 50% of the workload be operations.
Project Work

- Consulting on System Architecture and Design.
- Authoring and iterating on Monitoring.
- Automating repetitive work.
- Coordinating implementation of Postmortem Action Items
SREs have time to make tomorrow better than today.
SRE Principle #2

- An SRE’s job is not to suffer under operational load, but to make each day brighter.
- "Brighter" might mean different things: It depends on what your SREs find most useful to do.
- Less toil, more meaningful system improvements.
Shared Responsibility Model
Dumping all production services on an SRE team cannot work.
An overloaded team doesn’t have time to make tomorrow better than today.
Implementing a mechanism to give back-pressure to dev partners provides balance.
Regulating Workload

- Give 5% of the operational work to the developers
- Track SRE team project work.
  - Not completing projects? → something’s wrong.
- Analyse and on-board new systems only if they can be operated safely.
- If every problem has to be escalated to its developer: why is SRE carrying the pager?
Leadership Buy-in

Without leadership buy-in, SRE cannot work.
Leadership Buy-in

- When applications miss their SLOs and run out of Error Budget, it puts additional load on the SRE team:
  - Need to devote more company resources to addressing reliability concerns.
  - or: Loosen the SLO.
Reliability & Consistency Up Front

- Fixing a product after launch is always more expensive.
- SRE teams can and should consult up-front on designs:
  - Architecting resilient systems.
  - Maintaining consistency means fewer SREs can support more products.
Automation

Three places SRE teams can benefit from Automation:

1. To eliminate their toil - don't do things over and over!
2. To do capacity planning - auto-scaling instead of manual forecasting!
3. To fix issues automatically - if you can write the fix in a playbook, you can make the computer do it!
SRE Principle #3

SRE teams have the ability to regulate their workload.
SRE Principle #3

- Teams need to be able to prioritise and do the work.
- Each new system to maintain has a human cost.
- Must be able to push-back on unreliable practices and systems.
A Culture of Blamelessness
Recognize the Antipattern

“I'm extremely angry right now. People should lose their jobs if this was an error.

--Hawaii State Representative Matt Lopresti

(in reference to the 2018 Hawaii nuclear alert false alarm)

Source: “How Hawaii Could Have Sent A False Nuclear Alarm”, Wired, Lapowski, January 13, 2018
https://www.wired.com/story/hawaii-nuclear-missile-alert-false-explanation/
Embrace Failure

- by setting SLOs less than 100%
- by modeling blamelessness at all levels
- by stamping out blame wherever it is found
- by celebrating cases of “I made a mistake” that lead to outages being resolved faster.
Learn from Failure

- You’ve already paid the price in an outage.
- Write a blameless postmortem.
- Make postmortems widely available so others can learn too.
“Human” errors are really systems problems.
Keep Asking Why

- The root cause of an outage is never a person.
- Ask “why” for as many iterations as it takes to identify system-related causes.
- Prioritize system fixes that support people to make the right choices.
SRE Principle #4

Failure is an opportunity to improve.
SRE Principle #4

Failure is an opportunity to improve.

Not an excuse to brandish pitchforks
SRE Principle #4

- Failure happens, there is no way around it.
- Stop pointing fingers.
- Embrace failure to improve MTTD and MTTR.
- Proactively addressing failure \(\rightarrow\) more robust systems
Site Reliability Engineering Principles

1. SRE needs Service Level Objectives, with consequences.
2. SREs must have time to make tomorrow better than today.
3. SRE teams have the ability to regulate their workload.
4. Failure is an opportunity to improve.
<table>
<thead>
<tr>
<th>Reliability level</th>
<th>Allowed unreliability window</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>per year</td>
</tr>
<tr>
<td>90%</td>
<td>36.5 days</td>
</tr>
<tr>
<td>95%</td>
<td>18.25 days</td>
</tr>
<tr>
<td>99%</td>
<td>3.65 days</td>
</tr>
<tr>
<td>99.5%</td>
<td>1.83 days</td>
</tr>
<tr>
<td>99.9%</td>
<td>8.76 hours</td>
</tr>
<tr>
<td>99.95%</td>
<td>4.38 hours</td>
</tr>
<tr>
<td>99.99%</td>
<td>52.6 minutes</td>
</tr>
<tr>
<td>99.999%</td>
<td>5.26 minutes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reliability level</th>
<th>Allowed unreliability window</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>per year</td>
<td>per quarter</td>
</tr>
<tr>
<td>90%</td>
<td>36.5 days</td>
<td>9 days</td>
</tr>
<tr>
<td>95%</td>
<td>18.25 days</td>
<td>4.5 days</td>
</tr>
<tr>
<td>99%</td>
<td>3.65 days</td>
<td>21.6 hours</td>
</tr>
<tr>
<td>99.5%</td>
<td>1.83 days</td>
<td>10.8 hours</td>
</tr>
<tr>
<td>99.9%</td>
<td>8.76 hours</td>
<td>2.16 hours</td>
</tr>
<tr>
<td>99.95%</td>
<td>4.38 hours</td>
<td>1.08 hours</td>
</tr>
<tr>
<td>99.99%</td>
<td>52.6 minutes</td>
<td>12.96 minutes</td>
</tr>
<tr>
<td>99.999%</td>
<td>5.26 minutes</td>
<td>1.30 minutes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error Rate</th>
<th>Allowed duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>21.6 minutes</td>
</tr>
<tr>
<td>10%</td>
<td>3.6 hours</td>
</tr>
<tr>
<td>1%</td>
<td>36 hours</td>
</tr>
<tr>
<td>0.1%</td>
<td>15 days</td>
</tr>
<tr>
<td>&lt;0.05%</td>
<td>all month</td>
</tr>
</tbody>
</table>