Site Reliability Engineering

Balancing Risk and Velocity

Derek Remund
remund@google.com
https://www.linkedin.com/in/derekremund
DevOps: Still a thing?

(or dev ops, or dev-ops, or devops)
DevOps

Everyone’s (still) talking about it
DevOps

Everyone’s (still) talking about it

What does it mean to “do” DevOps? How do you know you’re doing it right? How do you know when you’re where you need to be?
Enter Site Reliability Engineering (SRE)

Getting some buzz since April 2016 (book release)
Enter Site Reliability Engineering (SRE)

Getting some buzz since April 2016 (book release)

Still not out-pacing DevOps
What is SRE?
“…what happens when a software engineer is tasked with what used to be called operations…”

- Ben Treynor Sloss, Google SRE
“...a specific implementation of DevOps with some idiosyncratic extensions...”

- The SRE Book
My Explanation

- An opinionated implementation of devops principles
- Site Reliability Engineering as a *practice* realigns dev, ops, product, and the business to achieve great customer outcomes
- Site Reliability Engineers as a *role* are developers working in an operational capacity to achieve reliability and performance goals for a service, with a user-centric focus
Principles

Site Reliability Engineering

- Hire great coders and let them leave when they want to
- Hire SREs and frontline devs form the same pool
- Approx. 5% of ops work goes to devs, plus all overflow
- Cap SRE operational load at 50% (ideally 30%)
- On-call teams have a minimum of 8 engineers in a locations
- Postmortems are blameless and focus on process and technology
- Require SLOs for each service and measure achievement
- Use error budgets as launch criteria
- Practice, Practice, Practice...and have fun

DevOps

6 Principles of DevOps – DASA - DevOps Agile Skills Association
https://www.devopsagileskills.org/dasa-devops-principles/


4 Principles of DevOps | Radify Blog
radify.io/blog/four-principles-of-devops/

Jul 2, 2014 - There are a lot of misconceptions about DevOps. Really, it’s a way of thinking. We explore 4 key principles that drive our operations.

The Three Ways: Core principles of devops | InfoWorld

Jul 22, 2016 - If devops has a foundation to build upon that foundation would probably be called The Three Ways. Originally developed by Gene Kim, these three principles underpin what devops is about. From here, many other principles are built but The Three Ways is what every other higher-level principle should be...

Adopting the 7 Principles of DevOps and Cloud Operations
https://logicalread.com/adopting-the-7-principles-of-devops/

These seven principles of DevOps bring benefits even to IT operation teams that operate separately from development teams.

8 DevOps Principles that will Improve your Speed to Market - Accenture

Sep 17, 2015 - I was recently asked about the principles that I follow for DevOps adoptions, so I thought I'd write down my list of principles and what they mean to me: Test early and often – This principle is probably the simplest one. Test as early as you can and as often as you can. In my ideal world, we find more and...
Principles

Site Reliability Engineering

- Hire great coders and let them leave when they want to
- Hire SREs and frontline devs form the same pool
- Approx. 5% of ops work goes to devs, plus all overflow
- Cap SRE operational load at 50% (ideally 30%)
- On-call teams have a minimum of 8 engineers in a location
- Postmortems are blameless and focus on process and technology
- Require SLOs for each service and measure achievement
- Use error budgets as launch criteria
- Practice, Practice, Practice...and have fun

DevOps

- Too Many Lists!!!

---

6 Principles of DevOps | DASA - DevOps Agile Skills Association
https://www.devosagileskills.com/blog/6-principles-of-devops/


4 Principles of DevOps | Radify Blog
radifyblog/4-principles-of-devops/

Jul 22, 2016 - There are a lot of misconceptions about DevOps. Really, it's a way of thinking. We explore 4 key principles that drive our operations.

The Three Ways: Core principles of devops | InfoWorld

Jul 22, 2016 - If devops has a foundation to build upon that foundation would probably be called The Three Ways. Originally developed by Gene Kim, these three principles underpin what devops is about. From here, many other principles are built but The Three Ways is what every other higher-level principle should be ...

Adopting the 7 Principles of DevOps and Cloud Operations
https://logicalread.com/adopting-the-7-principles-of-devops/

These seven principles of DevOps bring benefits even to IT operation teams that operate separately from development teams.

8 DevOps Principles that will Improve your Speed to Market - Accenture

Sep 17, 2015 - I was recently asked about the principles that I follow for DevOps adoptions, so I thought I’d write down my list of principles and what they mean to me: Test early and often – This principle is probably the simplest one. Test as early as you can and as often as you can. In my ideal world, we find more and ...
The Core

1. Define reliability in the context of the user: What does “good” mean?

2. Determine how reliable we need to be: How much “good” do we need?

3. Analyze and address risks to reliability: What do we fix when there isn’t enough “good”? 
What is “Reliability” then?

- Users are happy when we meet their implicit and explicit expectations

- Availability is the most common thing that comes to mind, but there are many possible measurements

- Expectations differ for each journey a user takes through the service or application
Service Level Indicators (SLIs)

- A measure of some aspect of the provided service, strongly correlated to user happiness
- Quantitative and carefully-defined
- Typically things like latency, throughput, availability, or correctness/quality
- Monitoring systems may (and should) capture a large number of metrics, but most will not become SLIs

\[
\text{SLI} : \left( \frac{\text{good events}}{\text{valid events}} \right) \times 100\%
\]
<table>
<thead>
<tr>
<th>SLI Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request/Response</td>
</tr>
<tr>
<td>● Availability</td>
</tr>
<tr>
<td>● Latency</td>
</tr>
<tr>
<td>● Quality</td>
</tr>
<tr>
<td>Persistence</td>
</tr>
<tr>
<td>● Throughput</td>
</tr>
<tr>
<td>● Latency</td>
</tr>
<tr>
<td>● Durability</td>
</tr>
<tr>
<td>Data Processing</td>
</tr>
<tr>
<td>● Coverage</td>
</tr>
<tr>
<td>● Correctness</td>
</tr>
<tr>
<td>● Freshness</td>
</tr>
<tr>
<td>● Throughput</td>
</tr>
</tbody>
</table>
A Fully-Qualified SLI

**User Journey:** Load Profile Page  
**SLI Type:** Latency  
**SLI Specification:** Proportion of requests for /home/{user} that delivered their payload in < 200ms  

**SLI Implementation:**
- Proportion of profile page requests served in < 200ms as measured from the ‘latency’ column of the server log
  - Pros: Measures data representative of majority of user requests  
  - Cons: Will miss requests that fail to reach the backend
- Proportion of profile page requests served in < 200ms as measured by black-box probers that execute javascript in a browser running in a virtual machine
  - Pros: This will catch errors when requests cannot reach the network  
  - May miss issues affecting only a subnet of users
Service Level Objectives

- Form the core of SRE, and are table stakes for running a service
- Define success by *measuring likely user happiness*
- Built from SLIs, whether simple or composite
A Word about SLAs

- SLAs are explicit contracts with users, usually with financial penalties attached.
- SLAs are constructed based on business and contractual needs; SREs may advise on SLAs, but do not define them.
- SLAs are not part of SRE, but if you have them and are practicing SRE, you should have associated SLOs to act as canaries in the SLA coal mine.

![Diagram showing objectives and agreements with user interaction](image-url)
Adding an SLO to our SLI

**User Journey:** Load Profile Page
**SLI Type:** Latency
**SLI Specification:** Proportion of requests for /home/{user} that delivered their payload in < 200ms

**SLI Implementation:**
- Proportion of profile page requests served in < 200ms as measured from the ‘latency’ column of the server log
  - Pros: Measures data representative of majority of user requests
  - Cons: Will miss requests that fail to reach the backend
- Proportion of profile page requests served in < 200ms as measured by black-box probers that execute javascript in a browser running in a virtual machine
  - Pros: This will catch errors when requests cannot reach the network
  - May miss issues affecting only a subnet of users

**SLO:** 99% of profile page requests in the rolling past 28 days served in < 200ms.
SLO Complexity

- Harder to define than you might expect
  - SLI interdependencies
  - SLIs that aren’t directly controllable
  - Service dependencies that cause stacking SLOs

- Evolution over time
  - Service requirements and cost realities change
  - Users grow to expect the performance you’ve delivered
  - Realities of risk, or your understanding of them, change
Error Budgets

- The complement (not inverse) of SLO attainment for a service
  - 99.9% SLO requirement = 0.1% Error Budget
- Measured over a defined period of time
- A tool to help stakeholders work together by minimizing politics and maximizing alignment: a shared definition of success
- As long as we are in budget, errors are tolerable or even useful
Error Budget Math

30 Day Error Budget for Availability:

99.9% SLO = 43.2 minutes of 100% downtime
99.99% SLO = 4.32 minutes of 100% downtime
99.999% SLO = 26 seconds of 100% downtime
Basic (naïve?) Error Budget Usage

- Product Management defines SLOs
- Actual achievement measured neutrally
- Remaining budget = goal minus reality
- As long as budget remains, so does our normal dev/ops cadence
Advanced Error Budgeting

- Alight budget with sprints
- Drive activity and/or alerting based on error budget consumption rates
- Keep a few “silver bullets” for business overrides
- Take risks to gain velocity (e.g. reduce UAT for a particular feature release)
- Run a portion of dev hours via Error Budgets, but a fixed percentage on regular feature dev cadence

Example Policy:

- Threshold 1: **Automated alerts** notify SRE of an at-risk SLO
- Threshold 2: SREs conclude they need help to defend SLO and **escalate to devs**
- Threshold 3: The 30-day error budget is exhausted and the root cause has not been found; **feature releases blocked**, dev team dedicates more resources
- Threshold 4: The 90-day error budget is exhausted and the root cause has not been found; SRE **escalates to executive leadership** to obtain more engineering time for reliability work
Risk Analysis

- Cataloging and estimating risks allows for validation of error budgets
- Look at MTTR, MTTD, and overall impact of DTEs, plus frequency
- Initial goal is order-of-magnitude correctness
- Stack rank risks and address as necessary to hit desired SLO targets

![Risk Analysis Diagram]

- User impact starts
- Human informed
- User impact ends

ETTD  ETTR

Time
Example Risk

**Risk:** Bad code push to canary production

**MTTD:** 3 minutes

**MTTR:** 2 minutes

**Impact:** 10% (push to 1/10 servers)

**MTBF:** 28 days (one every couple of sprints)
## Risk Catalog

<table>
<thead>
<tr>
<th>Risk</th>
<th>(in minutes)</th>
<th>(in users)</th>
<th>(in days)</th>
<th>incidents/year</th>
<th>bad mins/year</th>
</tr>
</thead>
<tbody>
<tr>
<td>annual fast detected major outage</td>
<td>5</td>
<td>60</td>
<td>100.00%</td>
<td>365</td>
<td>1.00</td>
</tr>
<tr>
<td>unlikely major risk, detected quickly</td>
<td>5</td>
<td>240</td>
<td>100.00%</td>
<td>730</td>
<td>0.50</td>
</tr>
<tr>
<td>frequent, small outages</td>
<td>10</td>
<td>10</td>
<td>100.00%</td>
<td>45</td>
<td>8.12</td>
</tr>
<tr>
<td>infrequent, small outages</td>
<td>10</td>
<td>20</td>
<td>30.00%</td>
<td>120</td>
<td>3.04</td>
</tr>
<tr>
<td>infrequent, large outage</td>
<td>10</td>
<td>30</td>
<td>100.00%</td>
<td>120</td>
<td>3.04</td>
</tr>
<tr>
<td>another, infrequent, large outage</td>
<td>10</td>
<td>30</td>
<td>100.00%</td>
<td>120</td>
<td>3.04</td>
</tr>
<tr>
<td>another, infrequent, small outage</td>
<td>10</td>
<td>10</td>
<td>50.00%</td>
<td>120</td>
<td>3.04</td>
</tr>
<tr>
<td>and another</td>
<td>5</td>
<td>20</td>
<td>100.00%</td>
<td>365</td>
<td>1.00</td>
</tr>
<tr>
<td>really short, frequent, outage for a 4-nines system</td>
<td>5</td>
<td>1</td>
<td>100.00%</td>
<td>90</td>
<td>4.06</td>
</tr>
<tr>
<td>really short, really frequent, outage for a 4-nines system</td>
<td>1</td>
<td>1</td>
<td>100.00%</td>
<td>30</td>
<td>12.18</td>
</tr>
<tr>
<td>Bad Canary rollout</td>
<td>5</td>
<td>5</td>
<td>10.00%</td>
<td>30</td>
<td>12.18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computed Stack Rank of Risks</td>
<td>bad mins/year</td>
<td>accept risk?</td>
<td>accept risk?</td>
<td>accept risk?</td>
<td>Colour Key</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>ETTR++ per incident (e.g., +5m due to lack of playbooks)</td>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td>red (risk cannot be accepted):</td>
</tr>
<tr>
<td>frequent, small outages</td>
<td>162</td>
<td></td>
<td></td>
<td></td>
<td>This risk is unacceptable, as it falls above the acceptable error budget for a single risk, and therefore, can have a major impact on your reliability in a single event.</td>
</tr>
<tr>
<td>unlikely major risk, detected quickly</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>infrequent, large outage</td>
<td>122</td>
<td></td>
<td>y</td>
<td></td>
<td>amber (risk should not be accepted):</td>
</tr>
<tr>
<td>another, infrequent, large outage</td>
<td>122</td>
<td></td>
<td></td>
<td></td>
<td>This risk should not be acceptable, as it’s a major consumer of your error budget and therefore, needs to be addressed. You may be able to accept some amber risks by addressing some less urgent (green) risks to buy back budget.</td>
</tr>
<tr>
<td>ETBF multiplier per incident (e.g., +10% due to lack of postn</td>
<td>74</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>annual fast detected major outage</td>
<td>65</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ETTD++ per incident (e.g., +30m due to operational overload</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>another, infrequent, small outage</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>infrequent, small outages</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and another</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>really short, frequent, outage for a 4-nines system</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td>green (risk could be accepted):</td>
</tr>
<tr>
<td>really short, really frequent, outage for a 4-nines system</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td>This is an acceptable risk. It is not a major consumer of your error budget and, in aggregate, does not cause your application to exceed the error budget. You don’t have to address green risks, but may wish to do so to give yourself more budget to cover unexpected risks, or to accept amber risks that are hard to mitigate or eliminate.</td>
</tr>
<tr>
<td>Bad canary rollout</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>blue (accepted risks)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This risk has been accepted to fit within your error budget. Accepting a risk means planning not to fix it and taking the outage and corresponding hit on the error budget. Be careful not to accept more than the size of your error budget!</td>
</tr>
</tbody>
</table>
Recap: The SLO loop

1. Stack rank critical user journeys
2. Select candidate SLIs - What makes users happy/unhappy?
3. Build aspirational SLOs
4. Perform risk analysis and gather data
5. Validate SLOs so they become achievable SLOs
6. Build and follow your Error Budget strategy
7. Review SLOs periodically
Recap: The SLO loop

1. Stack rank critical user journeys
2. Select candidate SLIs - What makes users happy/unhappy?
3. Build aspirational SLOs
4. Perform risk analysis and gather data
5. Validate SLOs so they become achievable SLOs
6. Build and follow your Error Budget strategy
7. Review SLOs periodically
8. Update LinkedIn listing SRE in your skillset
9. Collect mad recruiter connections
Next Steps: SRE Introspection

- Who is our user?
- What makes them happy?
- Can we measure it?
- Can we set some objectives?
- Can we measure them?
Next Steps: SRE Education

- SRE Books
  - https://landing.google.com/sre/books/
- Online Education
  - Coursera
- The Web
  - Google SRE blog
  - https://sre.xyz/