Engineering Reliable Mobile Applications

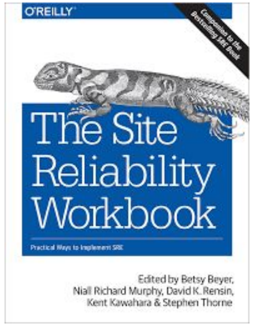
Conf42: SRE, Sep 2021

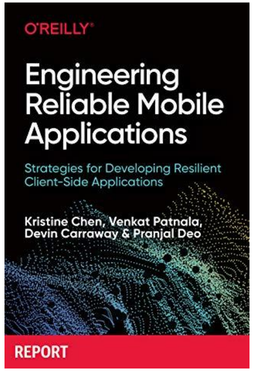
Pranjal Deo

SRE Program Manager, Client Infrastructure and Firebase

A little bit about me

- Site Reliability Engineering (SRE) Program Manager at Google
- External Engagements
  - Blameless Postmortem Chapter in the Site Reliability Workbook
    - Many talks!
  - Mobile reliability publication
    - This talk!
    - Many talks!
- Previous
  - Test automation / software engineering / DevOps at Brightidea Inc.
  - Electrical Engineer
  - Dance instructor
- Passions
  - Travel
  - Gastronomy


O'REILLY
The Site Reliability Workbook
Practical Ways to Implement SRE
Edited by Betsy Beyer, Niall Richard Murphy, David K. Rensin, Kent Kawahara & Stephen Thorne


O'REILLY
Engineering Reliable Mobile Applications
Strategies for Developing Resilient Client-Side Applications
Kristine Chen, Venkat Patnala, Devin Carraway & Pranjal Deo
REPORT

Agenda

- SRE for Mobile
- Challenges
    - Scale
    - Monitoring
    - Control
    - Change Management
- Strategies for developing resilient native mobile applications
- Case Studies: Google Doodle outage, Search app outage, Thundering Herd problem
- Key takeaways

Traditional SRE

- Availability
- Latency
- Efficiency
- Emergency response
- Change management
- Monitoring
- Capacity planning
- etc.

Site Reliability Engineering

1. SRE = Job role + mindset
2. *Hope is not a strategy*
3. Whole service lifecycle
4. Healthy services
5. Horizontal projects

Users <u>perceive</u> reliability of our services through the clients (devices).

What's the point of five 9s of server availability if your mobile application cannot access it?

SRE for Mobile

Focusing on the server-side does not entirely capture user experience anymore.

- Monitoring
- Rollouts
- Incident management & resolution
- Catch & fix/rollback issues in production fast
- Affect as few users as possible

1. Deliver code to users' devices
2. Make sure it works well
3. Things may only happen on a client
4. *Hope is not a mobile strategy either*

CHALLENGES

Challenge #1

Scale

- Billions of devices
- Thousands of device models
- Hundreds of applications
- Multiple versions of applications

Challenge #2

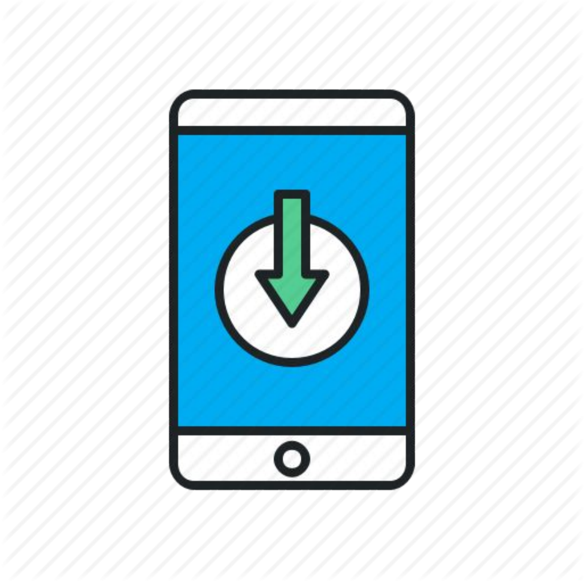
Monitoring

- Metrics have many dimensions because of scale
- Logging / monitoring has a tangible cost to the end user

Challenge #3

Control

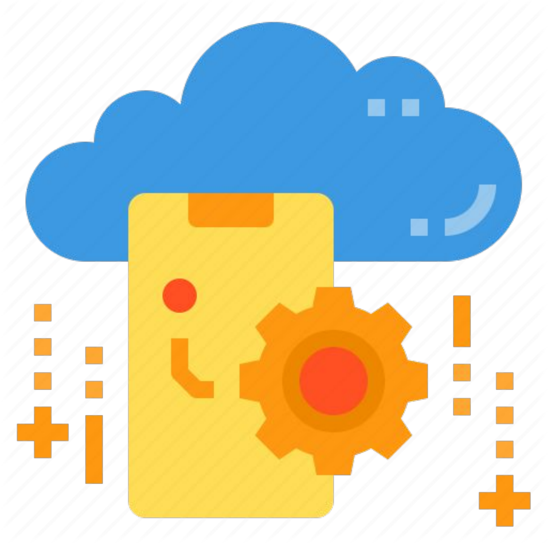
- Power lies with the user
- Upgrades come at a cost



Challenge #4

Change Management

- No rollbacks
- Power lies with the user
- This is very important!



CONCEPTS & STRATEGIES

App Availability

Examples of unavailability

- Tap icon, app about to load, then it immediately vanished
- Message saying “application has stopped” or “application not responding”
- App made no sign of responding to your tap
- Empty screen displayed
- Screen with old results, and you had to refresh
- Eventually abandoned by clicking the back button

Crash reports - Critical to monitor and triage.

Realtime Monitoring

- Reduce mean time to resolution (MTTR)
  - Faster problem detection, quicker investigation
- Get quick feedback on production fixes
- Typical server side fixes: Resolution time driven by humans
- Extra for Mobile: How fast can fixes be pushed to devices?
  - Polling oriented mobile experimentation and configuration
  - Uptake rate varies
  - Constrain view of error metrics to devices using your fix

Monitor metrics exposed by app internals

Run UI test probes for user journeys

Performance & Efficiency

- Mobile apps on a device share precious resources e.g. battery, network, storage, CPU, memory
- Particularly important for lower end devices
- Block launches that hamper user happiness

Change Management

- Problems found in production can be irrecoverable
- Take extra care when releasing client changes!
- Staged rollouts
  - Gradually gather production feedback
  - Diversify pool of users and devices
- Experimentation
  - Reduce bias caused by better network / devices
  - Release changes via experiments
  - A/B analysis over staged rollout
  - Randomized control and experiment groups
- Feature flags
  - Release code through binary releases and control user set via feature flags
  - Rollback shouldn't break the app
- Upgrade side effects and noise
  - Placebo binaries

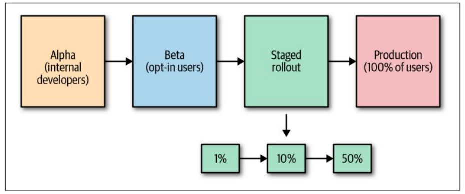


Figure 1-1. Release life cycle with a staged rollout in which 1% of users receive the new version, then 10% of users, then 50% of users, before releasing to all users

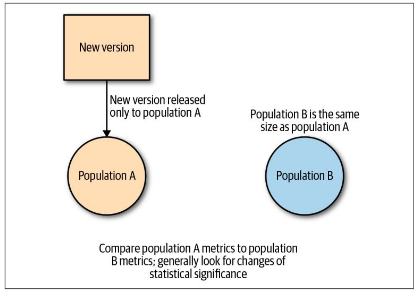

New version
New version released only to population A
Population B is the same size as population A
Population A
Population B
Compare population A metrics to population B metrics; generally look for changes of statistical significance

Figure 1-2. A/B experiment analysis for mobile change

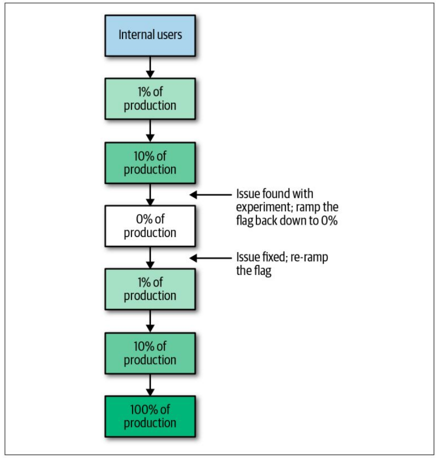
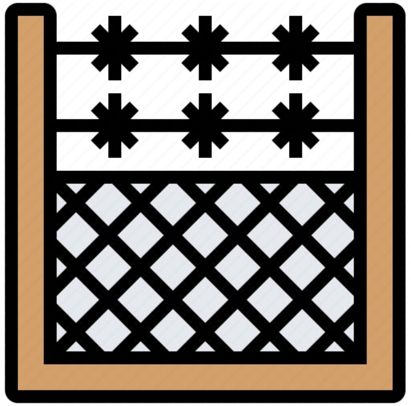


Figure 1-3. Example stages of a feature flag ramp: A feature flag's functionality is tested first on internal users before rolling out in stages to production users. If an issue is found on a subset of production users, it can be rolled back and the code is fixed before ramping the feature flag to 100%.

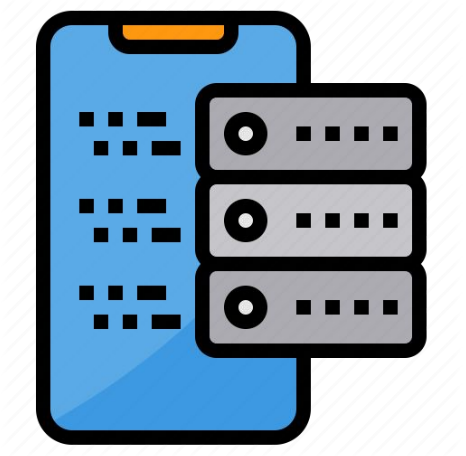
Support Horizons

- How many app versions can SRE meaningfully support?
- Older app version can never really go away
- Trade-off between reliability and business decisions



Server-Side Impact

- Client changes to apps impact servers
- Global events can suddenly overwhelm servers
- Client releases can cause unintended consequences



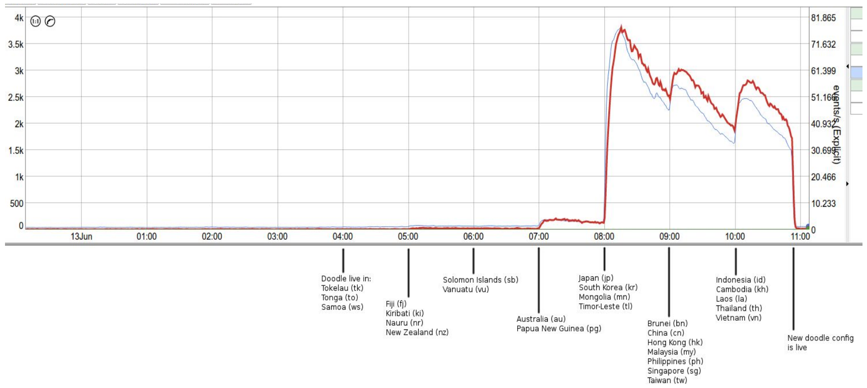
CASE STUDIES

#1

Android Google Search App (AGSA) Doodle Crashes

What happened?

- Bad Doodle configuration caused crashes in AGSA whenever user were shown a SERP (Search Engine Results Page)
- Triggered as doodle rolled out in each timezone
- Fix was submitted for this particular issue (both configuration and binary fix) but same issue happened again!
- Affected older versions without the fix

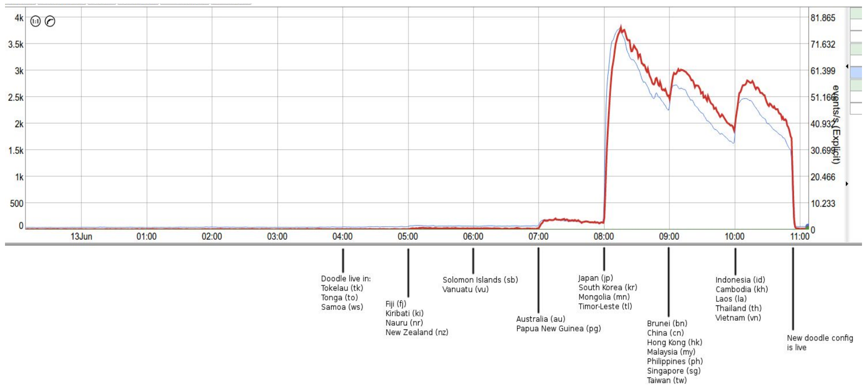


#1

Android Google Search App (AGSA) Doodle Crashes

Key Takeaways

- Client-only fixes may not fix everything (e.g. users may not update to the version with the fix); always include server-side fixes when possible
- Know your dependencies (especially if you have many feature teams contributing)

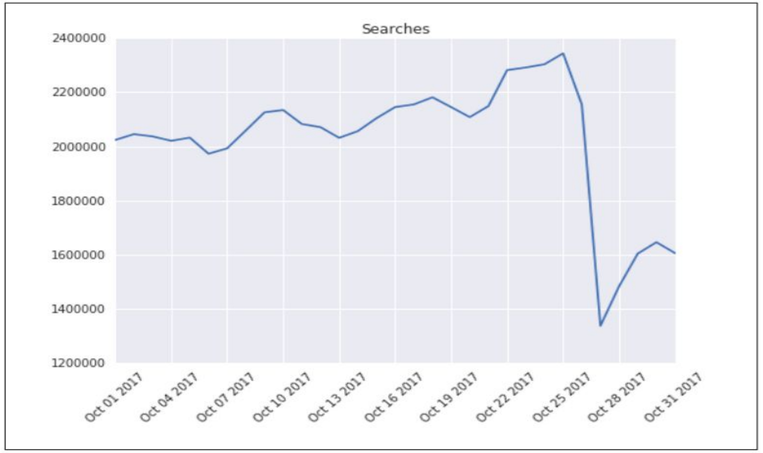


#2

Search broken for certain versions of AGSA

What happened?

- AGSA started crash looping on five older versions - a near miss of a massive outage
- A simple four character change to a config, caused a crash at app startup
- Unable to fetch the rolled back config before crashing
- Only recovery: notify users to upgrade or clear app data

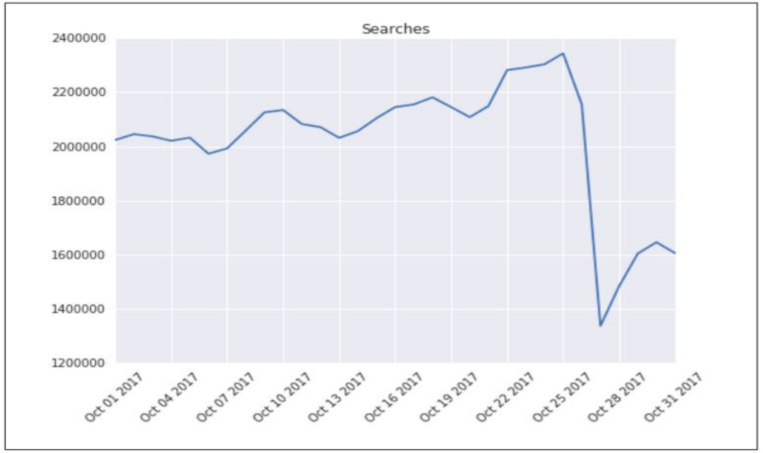


#2

Search broken for certain versions of AGSA

Key takeaways

- Lots of older app versions in the wild
- "Apply" before "Commit": always validate and exercise the new config before committing (i.e. caching)
- Expire regularly cached configuration in a reliable manner
- Detect and self-recover from crash loops
- Don't rely on recovery external to the app
- Sending notifications for manual recovery has limited utility
- Monitor crash recovery

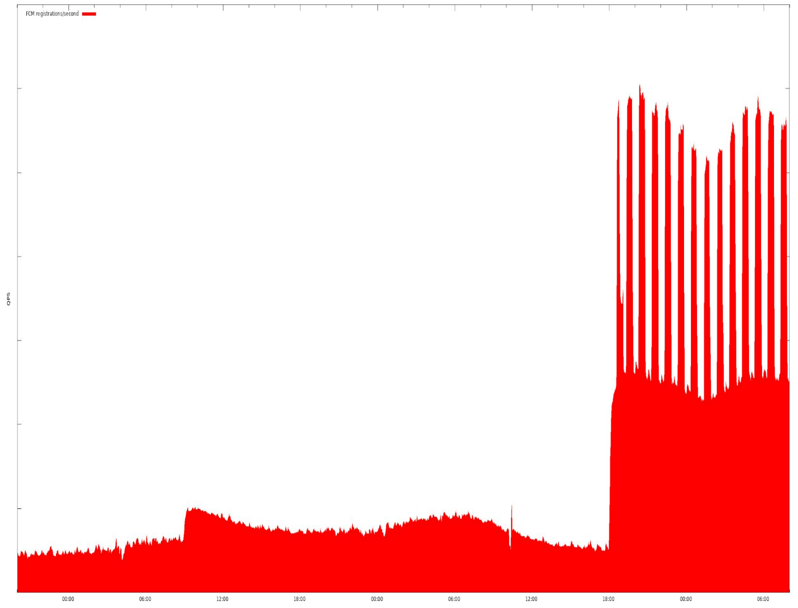


#3

Thundering Herd problem

What happened?

- A GMSCore (Google Play Services) update caused devices to register for Firebase Cloud Messaging (FCM) notifications at install time
- FCM is not scaled to support 2B devices updating at GMSCore's update rate, so it throttled all GMSCore registrations globally
- This could easily have been a global outage

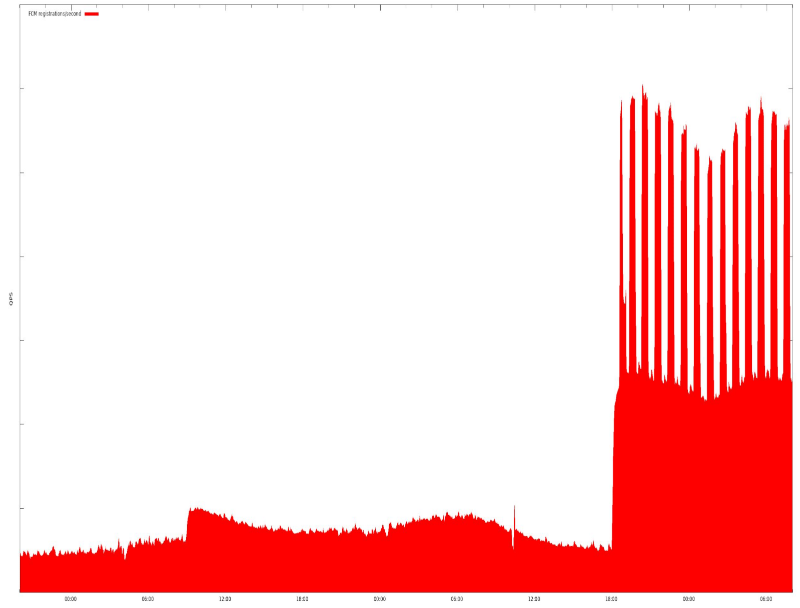


#3

Thundering Herd problem

Key Takeaways

- Don't make service calls during upgrades
- Server calls should be an app release qualification criteria
- App release rates are probably not well correlated with server capacity management



Hope is not a Mobile strategy

- Rollout changes in a controlled, metric driven way
- Monitor apps in production by measuring critical user interactions and key health metrics
- Prepare for app's impact on servers
- Create Incident management processes specific to client side
- Make client reliability a part of your mission!

THANK YOU!