Maintaining an open source project while sustaining your sanity

Andrew Gaul

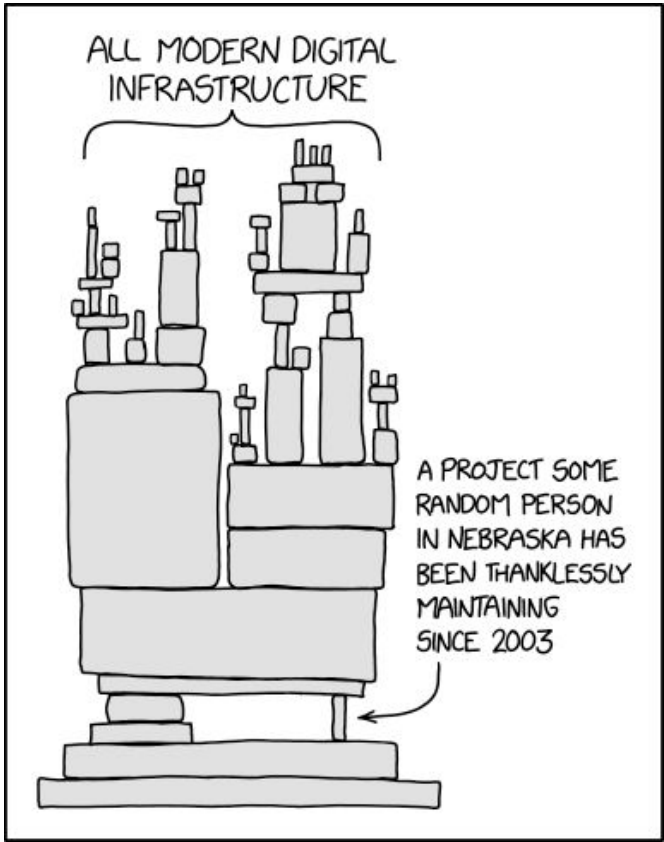
Open Source Summit Japan

15 December 2021

https://youtu.be/F-RAztCSOyc

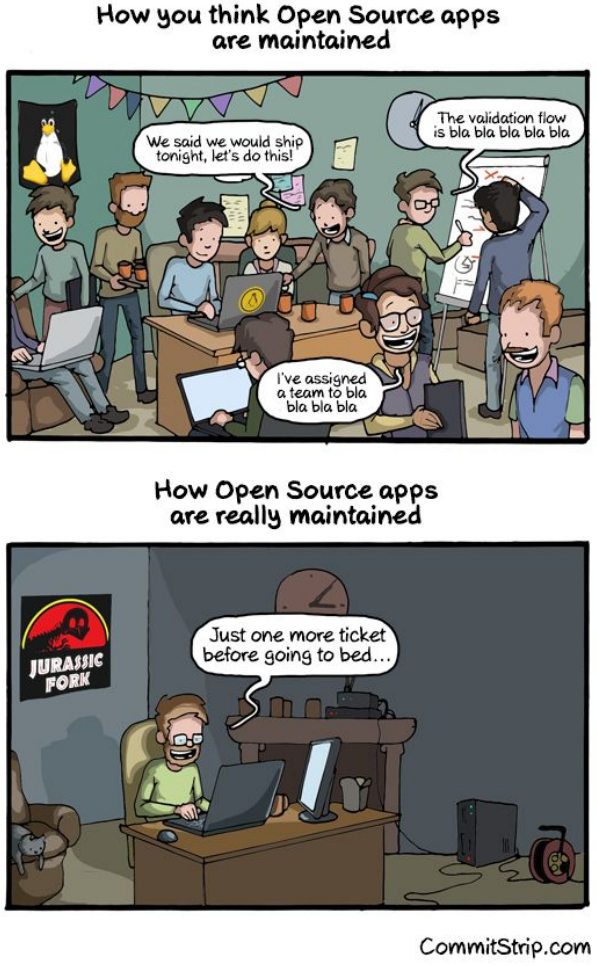
About me

- Maintainer of small- to medium-size projects
- Apache jclouds committer since 2012
  - Originally many maintainers but now only a few
  - 100s of contributors
- S3Proxy author, created in 2014
  - Only one maintainer
  - ~30 contributors
- s3fs contributor since 2015, committer since 2019
  - Two maintainers, neither are the original authors
  - ~100 contributors
- Live in Tokyo 🇯🇵



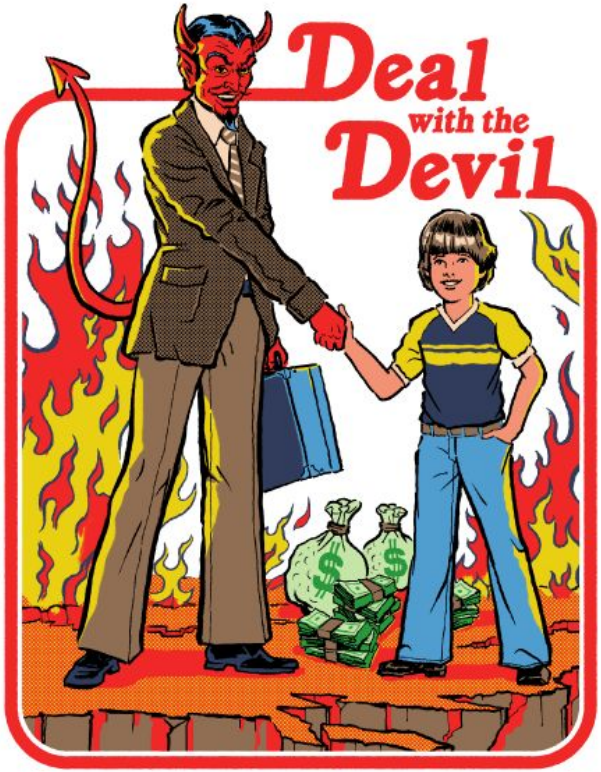
What is project maintainership?

- Working with users
- Triaging issues
- Cutting releases
- Scoping project
- Coordinating contributors
- Working with external projects
- Reviewing code
- Improving project quality
- Writing new code (sometimes)



Sustaining your sanity

- You will burn yourself out if you try to do everything by yourself
- Instead consider your project from different vantage points
- Try doing less but different work over longer periods of time
- This may allow you to sustain a project for years
- But you will have to get into management



Project maintainers wear many hats

- Product manager
  - Advocate for the user
- Engineering manager
  - Advocate for the team
- Technical lead
  - Advocate for the code
- This talk explores what you can do with few resources and lack of corporate sponsorship

Part 1: Thinking like a product manager

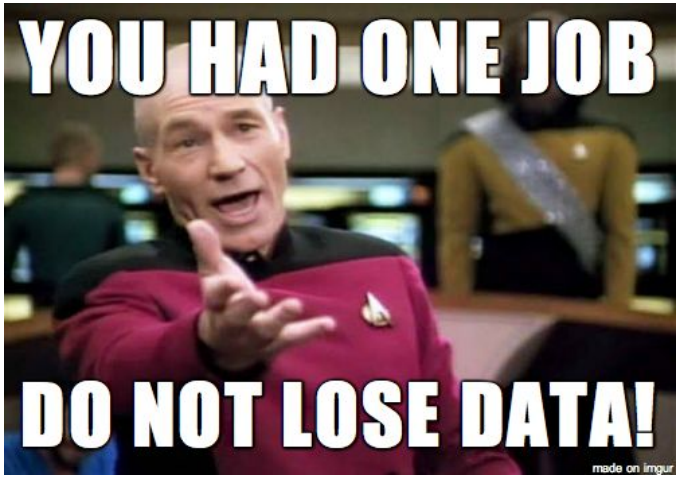
- Advocate for the user
- Evaluate the project against the ecosystem
  - Look at many sources: blogs, conferences, Hacker News, Reddit, Stack Overflow, Twitter
- Think about how users interact with project
  - Configuration, releases, packaging, documentation, backwards compatibility
- Prioritize new features and critical fixes
  - Immediate workarounds can be better than proper fixes in the future
- Look at “competing” projects

Issues

- Issues can be the highest-quality feedback from your users
  - Or a junkyard of vague, unresolved symptoms and abuse
- Periodic grooming can help you and your users understand the project
- Clarify, de-duplicate, and close issues
- Proactively use HELPWANTED and NEEDINFO labels
  - Avoids ambiguity and misunderstandings

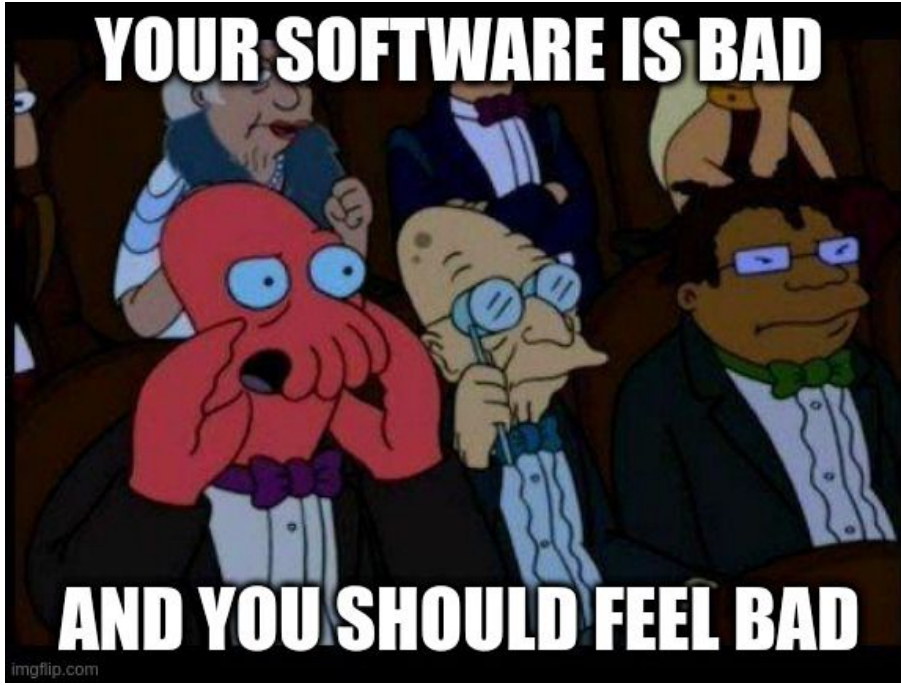
Case study: s3fs

- s3fs mounts S3 buckets as a filesystem
- Grooming revealed the most common issues
  - POSIX permissions conflicted with S3 interoperability
    - Required unintuitive configuration workaround
    - ~20 line PR fixed many issues simultaneously
  - Multiple symptoms of data corruption
    - Required years of iteration with users, improved testing, new tooling, and several bug fixes
- Hundreds of other issues
  - Categorized and ignored for years
  - Making slow progress addressing these



Working with users

- Most users are helpful or neutral
- Angry people cannot be helped -- ignore and report abuse if needed
- Entitled people misunderstand their relationship to the project -- can be educated
- Flip the script -- users should work for maintainers
    - Can users clarify issues, test proposed fixes, investigate workarounds, etc.?



Releases

- Most users consume release versions, not development branches
- Recommend regular, time-based schedule
  - Every six months or less
- Prepare to avoid buggy releases
  - Open a tracking issue and ask users to test
  - Slow down changes
- Beware of users reporting new bugs against old versions!
  - Understand why users prefer old releases, e.g., stale distribution packages, API breakage, regressions

Part 2: Thinking like an engineering manager

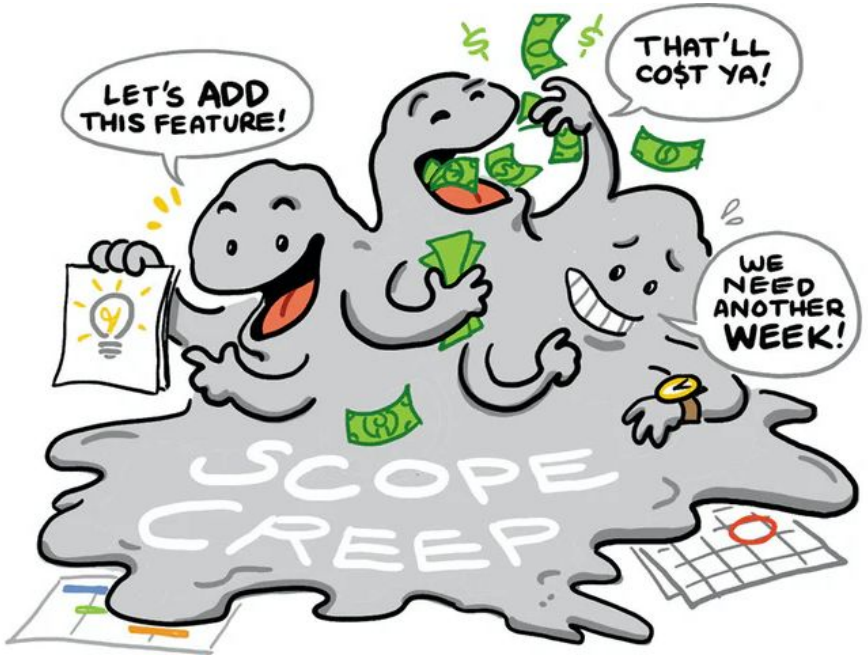
- Advocate for the team
- Managing project scope against goals
- Bring people and resources onto the project
- Coordination with external projects
- Dealing with forks

Managing the team

- Usually cannot tell contributors what to do
  - Instead @mention users to ask for help
- Any blockers to new contributors?
  - Toxic environment, build issues, technical debt, missing license, unclear code formatting
- Nudge issue reporters to become contributors
  - Suggesting how to fix an issue or where to start can overcome friction
- Outsource testing of PRs to issue reporters
- Dependencies are logically part of your code
  - Do you report issues upstream?

Project scope

- Anti-pattern: be everything to everyone
- Not all proposed changes need to be merged upstream
  - Even well-written ones!
- Will contribution increase your maintenance burden?
  - If so, will the contributor maintain the code?
- Project scope is easy to widen but difficult to narrow
  - Say “no” or “not yet” early and often



Case study: Apache jclouds

- jclouds is a Java-based cross-cloud abstraction
- Declining project activity despite stable user base and downloads
  - More work done by fewer maintainers!
- Scope was too large -- shrank project to compensate
  - Removed unmaintained code
  - Outsourced difficult features to separate projects (Karaf, CLI)
  - Dropped Java 6 and 7 compatibility
  - Upgraded or removed old dependencies (Guava, Guice)
  - Reducing number of repositories
- These changes should have been made years ago!

Dealing with forks

- Forking has several causes
  - Most benign is cherry-picking fixes until the next release
  - Some have local functionality that is inappropriate or not ready for mainline
  - A few have a major scope, license, or interpersonal disagreement
- Re-integrating forks can unify users and development effort
  - Look for opportunities to collaborate to reduce user pain
- Most forks die of neglect but it is worth understanding their motivations

Case study: s3fs

- s3fs mounts S3 buckets as a filesystem
  - Several forks and reimplementations due to inconsistent maintainership
- Aliyun and Tencent clouds offer S3-compatible cloud services
  - Forked ossfs and cosfs in 2016 to fix bugs and customize
  - Re-integrated in 2020 -- now some of their developers work with mainline
- Another cloud asked to add a non-S3 protocol
  - Told contributor “not yet” so mainline can focus on S3
  - Offered to merge partial changes that make their fork easier

Niche users

- Open source allows adapting to new use cases
- Some users are pioneering
  - Large-scale, Raspberry Pi, etc.
- But some users ask for esoteric things
  - RHEL 6, Java 7, AIX, or POWER compatibility
- You cannot do it all
  - Let niche users fork and periodically rebase

Part 3: Thinking like a technical lead

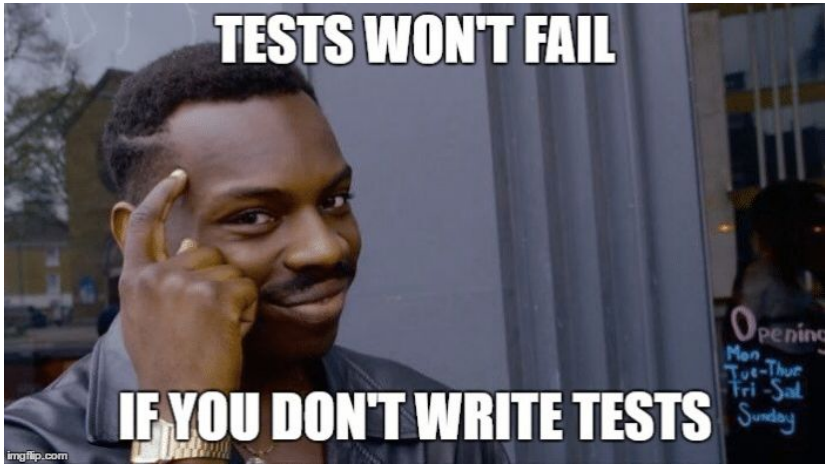
- Advocate for the technology
- Protect existing code
- Think about project evolution over the long term
- Make proactive technical investments
- Communicate with contributors

Protecting existing code

- Existing code is more important than new code
  - Cost of regressions is higher than you think
- Write tests and use continuous integration
- Code review helps but is labor intensive
  - Especially working with new contributors
- Break up large, risky changes into multiple smaller ones spread over time
- Evaluate how a proposed change interacts with current and future features

Case study: S3Proxy

- S3Proxy implements the S3 API with different backends, e.g., Azure, filesystem
- Adopted Ceph s3-tests project soon after project creation
  - Solid test coverage that improved over time
  - Reduced need to create custom tests
  - But it is a difficult project to work with that required a long-lived fork
- s3-tests have prevented many regressions from contributors
- Aligned project with the ecosystem



Evaluating technical risk

- Critically evaluate proposed changes
  - Does it improve the user experience?
  - Does it limit your contributors?
  - Does it reduce or increase the maintenance burden?
- Choose boring technology?
  - You have limited innovation tokens -- spend wisely
  - If you take all the risks one of them will burn you
- Think about project evolution
  - Will a new library still have maintainers next year?

Technical debt

- Any issue that increases development friction
- Missing, flaky, and slow tests
- Outdated or unnecessary dependencies
- Move from in-tree custom implementations to shared third-party libraries
- Push features out of your project into other projects
- Try to pay down (some) existing debt before taking on more debts

Case study: s3fs

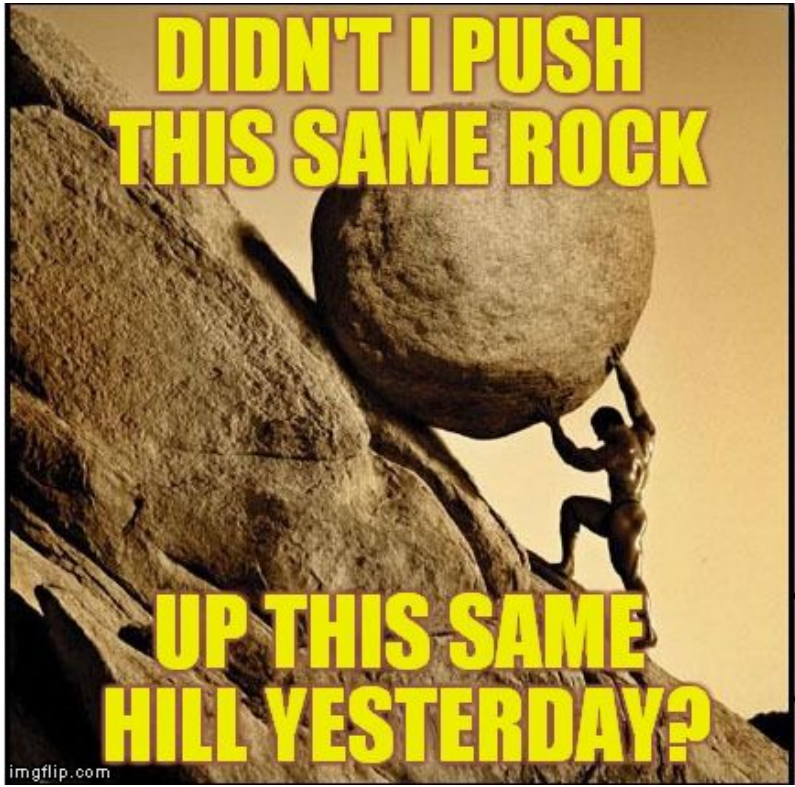
- s3fs mounts S3 buckets as a filesystem
- Implements a custom S3 client via libcurl
  - Legacy problem since s3fs predates third-party libraries
  - Historically a source of bugs
  - Frustration due to missing authentication mechanisms
- Transitioning to AWS SDK would address this debt but comes with trade-offs
  - Requires newer C++ compiler and refactoring
  - Periodically evaluate but always defer
- Maintainers keep paying small short-term costs instead of addressing long-term issue

Thinking out loud

- Internet makes some kinds of collaboration more difficult
- Important to foster a sense of community with contributors and users
  - Can you share your ideas on mailing lists, GitHub issues, or Twitter?
  - Can you ask others what they think?
- Bias towards over-communicating
  - Share work-in-progress commits
- Document design in issues, PRs, and wikis
  - Non-developers rarely read code comments

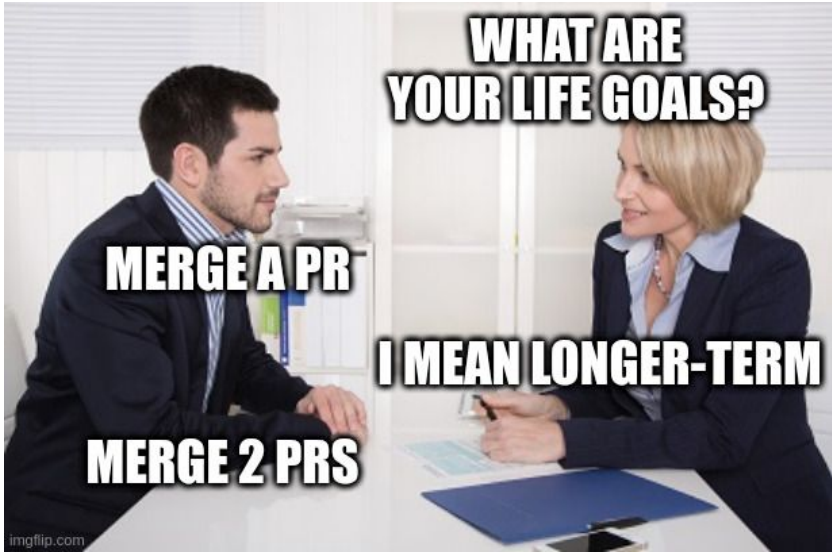
Part 4: Thinking about sustainability

- Project maintainership is a lot of work
- You may need to write less code to make time for these activities
- Aim to do fewer things over longer periods of time
- Sometimes the original author should step aside to let others maintain a project



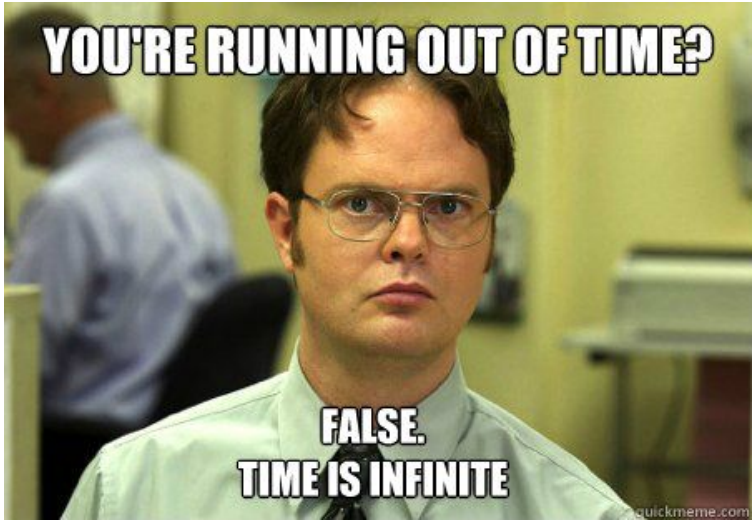
Thinking about the long term

- Where do you want your project to be in 1 year?
  - ...in 10 years?
- What kinds of changes would be truly impactful?
  - Easy to be reactive and fix random issues from loud users
  - Are there broad themes that you can make progress on?
- Your project may outlast your interest in it
  - Can you prepare for the next maintainer?



Thinking about maintainer commitments

- How much time do you want to dedicate to a project per month?
  - 1 hour? 10 hours? 100 hours?!
  - Setting a budget helps prioritize your work
- What quality-of-service you want to provide?
  - Consider batch-processing instead of interrupt-driven development
- Not everything needs to be done today
  - ...or maybe ever!



Conclusion

- Prioritize issues that are important to users
- Enable contributors and outsource tasks
- Make technology decisions that allow long-term evolution
- Say “no” and “not yet” more often
- Think about the long-term

Thank you!

Andrew Gaul
http://gaul.org/talks
Open Source Summit Japan
15 December 2021