Always Write Something

Author: Yann Tambouret

Claps: 234

Date: May 30

John Meichle
circulated the first Klaviyo RFC days after Black Friday, Cyber Monday (BFCM) 2017. He presented *RFC: Jenkins + AMI build pipeline*, which discussed automating away a manually triggered (though scripted) task. John co-opted the RFC pattern from a previous organization. Our people and systems were scaling quickly and John saw the need to be more formal about our technical planning and execution.

Since that first RFC, or *Request For Comments*, the process has become the standard Klaviyo engineers follow before developing projects of all sizes. Itâ€™s not at all a new concept, and the first Internet Standards RFC dates back to 1969. Many other contemporary engineering teams author RFC documents, for example, see Squarespaceâ€™s *The Power of â€œYes, Ifâ€�* and

Juan Pablo BuriticÃ¡
â€™s *A thorough team guide to RFCs*.

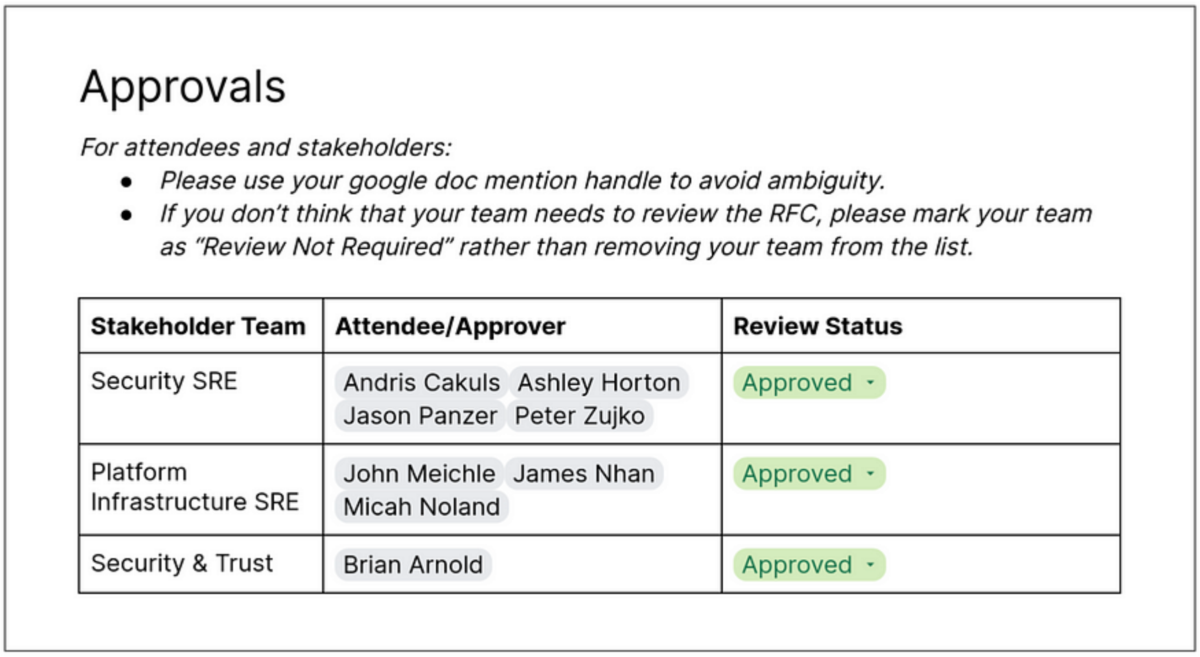
Early on, we evangelized the process with the motto *Always Write Something*. Weâ€™ve since authored and reviewed over 600 RFCs, covering everything from small data migrations to entirely new revenue generating systems.

In my 5+ years at Klaviyo, Iâ€™ve reviewed hundreds of RFCs and written over a dozen. Iâ€™m also a member of the Architecture Review Board (ARB), a group that stewards the RFC reviews. RFCs have become the foundation for our technical decision making. In this post Iâ€™ll share how RFCs work at Klaviyo with real examples.

What is an RFC and how does it work?

An RFC involves both writing your ideas down in a document and presenting them to your stakeholders for discussion. Communication is essential for engineers to work effectively with one another, especially when the organization grows beyond a few teammates. The RFC process encourages healthy communication. We want engineers with all levels of experience to have a positive and useful experience sharing ideas, and we want these ideas to be described and discussed in a consistent and high-quality way.

RFCs start as team discussions, which are then written up as technical specifications. When writing an RFC, authors use a standard template. The ARB maintains and iteratively improves the RFC templates (there are subtle derivatives of the one linked above, each focusing on domain specific details). The authoring teams then share these RFC documents with stakeholders, who are teams that will be impacted one way or another by the proposed changes. Stakeholders can be one of our SRE teams, e.g. Security Engineering, or they can be teams whose owned systems will interact with the proposed changes. For every RFC, the ARB is a required stakeholder. This helps ensure consistency and reliability across all teams.


Approvals

For attendees and stakeholders:

- *Please use your google doc mention handle to avoid ambiguity.*
- *If you don't think that your team needs to review the RFC, please mark your team as "Review Not Required" rather than removing your team from the list.*

Stakeholder Team	Attendee/Approver	Review Status
Security SRE	Andris Cakuls, Ashley Horton, Jason Panzer, Peter Zujko	Approved
Platform Infrastructure SRE	John Meichle, James Nhan, Micah Noland	Approved
Security & Trust	Brian Arnold	Approved

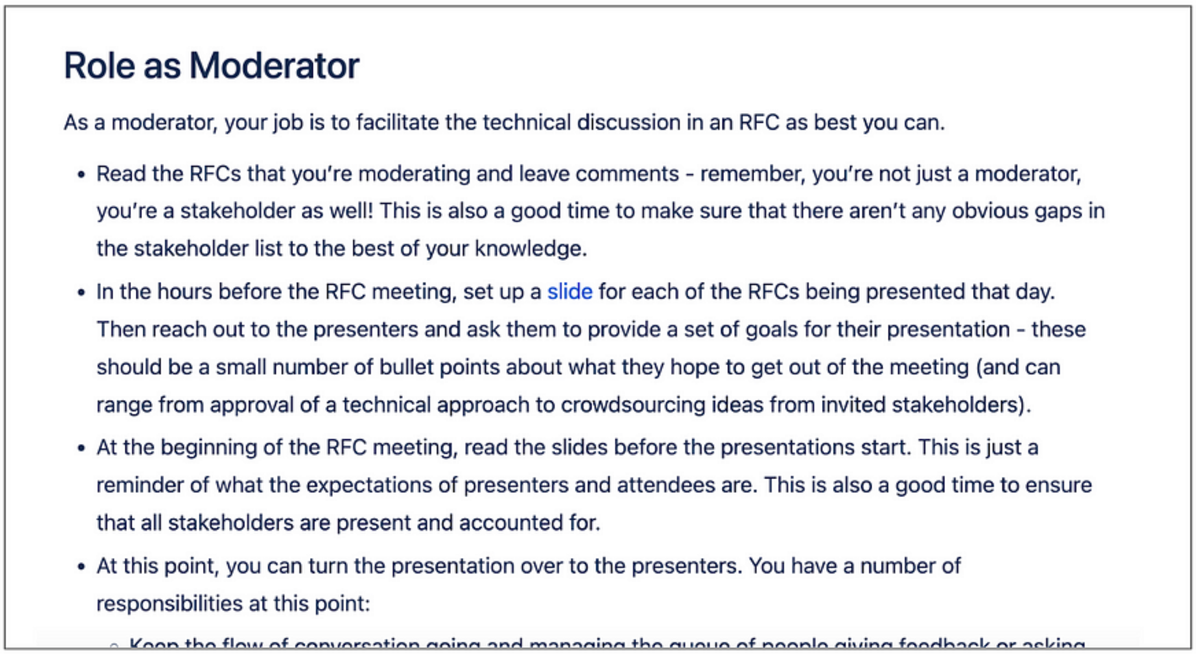
An example approvals section of the RFC, listing out stakeholders and their review status.

Discussion starts asynchronously in the form of comments on the RFC document and over Slack. Meanwhile, the author schedules a meeting to finalize the discussion. We reserve four one hour blocks per week to hold these discussions and authors can reserve 30 or 60 minutes â€" itâ€™s usually clear from the async comments how long it will take to resolve open points. Holding these blocks is a small but important process detail. Otherwise, if authors were left on their own to find blocks of time, weeks could be lost to scheduling.

These discussion sessions are open to everyone at Klaviyo. For that reason, RFCs are an excellent learning opportunity for new team members. Liam Kelly, a colleague and ARB member, put it this way:

> One thing that impressed me about Klaviyoâ€™s RFC process is the openness of it â€" i.e. everyone can view and comment on every RFC if they want, and anyone can drop in and observe any RFC review. Those things were a big help when I was onboarding as far as understanding the work of other teams.

An ARB member moderates the RFC discussion. Our goal is not just moderation, but also to prompt exploration of any under-addressed aspects of the proposal.



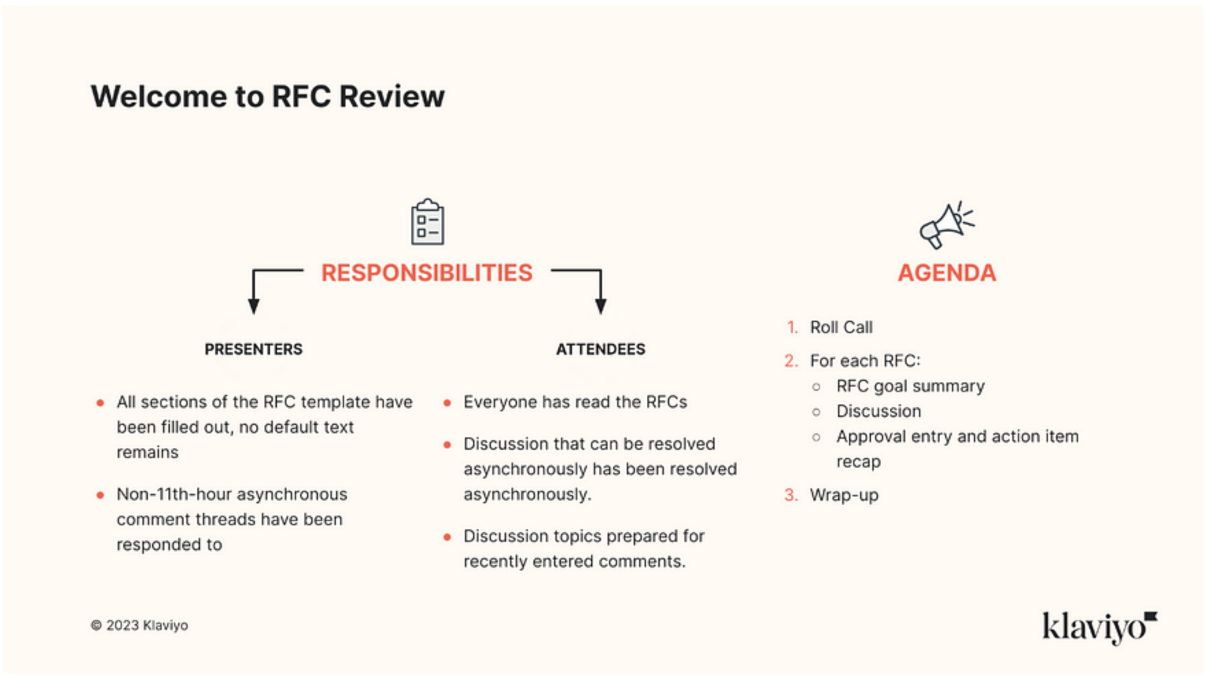
Role as Moderator

As a moderator, your job is to facilitate the technical discussion in an RFC as best you can.

- Read the RFCs that you're moderating and leave comments - remember, you're not just a moderator, you're a stakeholder as well! This is also a good time to make sure that there aren't any obvious gaps in the stakeholder list to the best of your knowledge.
- In the hours before the RFC meeting, set up a slide for each of the RFCs being presented that day. Then reach out to the presenters and ask them to provide a set of goals for their presentation - these should be a small number of bullet points about what they hope to get out of the meeting (and can range from approval of a technical approach to crowdsourcing ideas from invited stakeholders).
- At the beginning of the RFC meeting, read the slides before the presentations start. This is just a reminder of what the expectations of presenters and attendees are. This is also a good time to ensure that all stakeholders are present and accounted for.
- At this point, you can turn the presentation over to the presenters. You have a number of responsibilities at this point:
    - Keep the flow of conversation going and managing the queue of people giving feedback or asking

Page from our internal wiki describing the RFC moderation process.

Hereâ€™s the slide we show before each RFC discussion to remind everyone the rules of play.



Welcome to RFC Review

RESPONSIBILITIES

PRESENTERS

- All sections of the RFC template have been filled out, no default text remains
- Non-11th-hour asynchronous comment threads have been responded to

ATTENDEES

- Everyone has read the RFCs
- Discussion that can be resolved asynchronously has been resolved asynchronously.
- Discussion topics prepared for recently entered comments.

AGENDA

1. Roll Call
2. For each RFC:
    - RFC goal summary
    - Discussion
    - Approval entry and action item recap
3. Wrap-up

© 2023 Klaviyo

klaviyo

The â€œWelcome to RFC Reviewâ€� slide presented before each meeting.

(The slide is less fun than our initial effort: playing a Gilbert Godfreed impersonator [reading of the guidelines out loud]!)

Many RFC discussions resolve with all teams agreeing on a path forward. If not, thereâ€™s a process for iteration, and the outcome of the discussion might simply be a list of action items to address before re-review of the plan. The eventual goal is that stakeholders give approval and the RFC is ready for development. An alternative, and also positive outcome, is one where the process of writing and discussion makes us realize the plan is missing something that deserves more attention. This sometimes results in scrapping the approach or even the entire project.

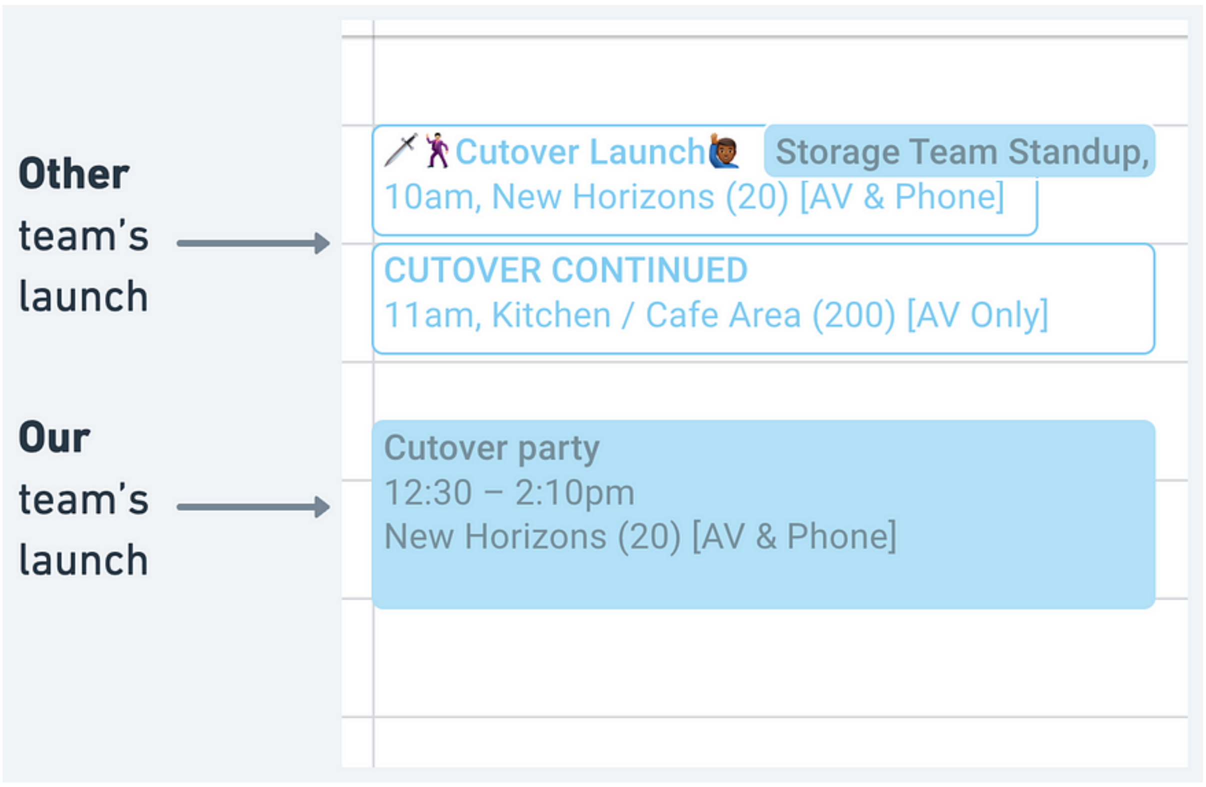
How long does this all take? Writing the initial draft varies widely. It could be that the author clearly understands the idea and writing it down takes half a day. Or it could be that in writing the RFC, the author is also writing prototype code, performance testing packages, etc., in which case it will take much longer. At any rate, once drafted, based on reviewer responsiveness, schedule congestion and the number of follow up action items, itâ€™s usually one to three weeks to approval.

The focus of the RFC document and discussion is purely technical. Itâ€™s *not* a contract among teams regarding timelines, commitments or resourcing.

A personal story

I believe most of my colleagues value the RFC process including the role of the ARB, but not everyone values it the first day they join. When I joined, Klaviyo neither used RFCs nor had an ARB to review technical plans. Thereâ€™s nothing like screwing up to develop conviction that formal socialization of ideas makes sense. That happened to me on October 30, 2018.

On that day, my team and I launched a massive rewrite to our event ingestion system. On that very same day, another team launched a whole new architecture for product catalogs.



My schedule on the day of the big cutover

Both releases went badly, and what made things even worse was that for ten hours we tried to debug issues in production. (We were a much younger company back then!) We had retros (oh there were retros for sure) and Iâ€™ve internally retroed about the actions leading up to and following that day ever since.

And after all those retros, it was clear to me and everyone else that more time on design, more time spent making sure we were solving the right technical problems, and more time making sure people on other teams knew what we were up to could have prevented months and months of pain and wasted effort.

In large part these events motivated the creation of the Architecture Review Board.

A request for permission?

Everyone on the ARB is an experienced engineer, and they bring that expertise to bear on RFC reviews. The ARB also works to indoctrinate our rapidly growing engineering team with the communication etiquette needed to support the RFC process at scale.

In my capacity as an ARB member, I go into each review, each interaction, serving as a supportive colleague. As a default reviewer on every RFC, the ARB has the authority to block a project. That might seem weird since we have a *request for comments* process, not a *request for permission*. Even though weâ€™ve rarely had an RFC blocked by the ARB and simultaneously supported by all other reviewers, weâ€™ve found that the authority plays a key role in validating the significance of the RFC process. Without that authority and ARBâ€™s continuous engagement, the quality of the discussions and the RFC process would suffer.

Itâ€™s not just the ARB that can block an RFC from proceeding. All of the stakeholders have the responsibility to prevent disastrous choices from impacting future systems. Their approval signifies that their systems, either as consumers of or sources to the RFCâ€™d system, will work well with the proposed changes. Authors must address questions of increased load, reliability, security and scaling (especially during peak holiday season) in order to gain approval.

What makes an RFC effective?

The RFC process formalizes high quality communication. RFC etiquette includes:

Know your responsibilities and own them.

Authors are responsible for sharing their RFC early enough for teams to review the proposal. They should also respond to all comments within a reasonable time â€” if youâ€™re going to ask someone to read your admittedly often dry document, they deserve the same respect and responsiveness in return.

Communicate early and often.

Always Write Something goes to the heart of this idea. Get in front of roadblocks by sharing ideas as early as possible. We encourage teams to RFC ideas before they are fully baked â€” the feedback theyâ€™ll receive could let them explore a direction they wouldnâ€™t otherwise consider and would be reluctant to later.

Assume everyone is operating with everyoneâ€™s best interest in mind.

Disagreements are common with RFCs, but they are never about personal issues, slights, offenses or politics.

Klassic Klaviyo Meme

If you have an opinion, make sure you explain it clearly and completely.

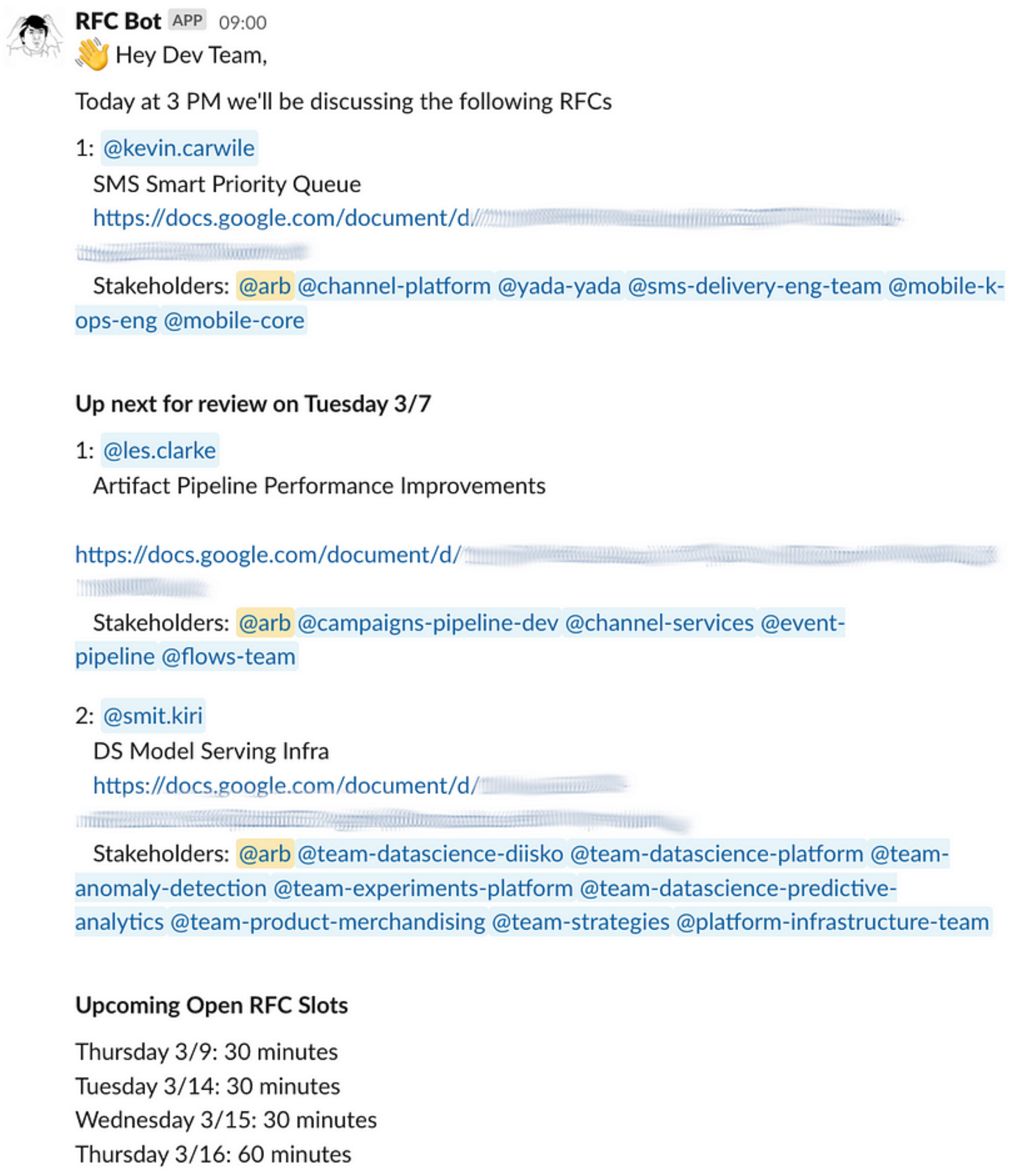
As a corollary, if you receive a comment, please return the courtesy with a reply.

Weâ€™re all working towards a better Klaviyo application.

Any one engineerâ€™s â€œownershipâ€� of code or service is really a temporary stewardship. The corporation owns the code and itâ€™s our job to make sure itâ€™s the highest quality possible.

Stick to a consistent routine.

The ARB developed and supported several strategies to foster this healthy etiquette. For example, I developed a lightweight Django app to manage scheduling and stakeholder alerting (on Slack). While itâ€™s not necessary that an ARB exists to manage the RFC process, it is essential that some individual or group own the process and make sure everyone remains committed. The additional overhead to create this app was well worth the efficiency gains weâ€™ve seen from establishing a routine. Though adhering to the process may feel like an administrative chore, doing so consistently makes managing the process easier in the long run.


RFC Bot APP 09:00
👋 Hey Dev Team,

Today at 3 PM we'll be discussing the following RFCs

1: @kevin.carwile
SMS Smart Priority Queue
https://docs.google.com/document/d/[redacted]
Stakeholders: @arb @channel-platform @yada-yada @sms-delivery-eng-team @mobile-k-ops-eng @mobile-core

Up next for review on Tuesday 3/7

1: @les.clarke
Artifact Pipeline Performance Improvements

https://docs.google.com/document/d/[redacted]
Stakeholders: @arb @campaigns-pipeline-dev @channel-services @event-pipeline @flows-team

2: @smit.kiri
DS Model Serving Infra
https://docs.google.com/document/d/[redacted]
Stakeholders: @arb @team-datascience-diisko @team-datascience-platform @team-anomaly-detection @team-experiments-platform @team-datascience-predictive-analytics @team-product-merchandising @team-strategies @platform-infrastructure-team

Upcoming Open RFC Slots

Thursday 3/9: 30 minutes
Tuesday 3/14: 30 minutes
Wednesday 3/15: 30 minutes
Thursday 3/16: 60 minutes

Screenshot from our RFC Bot in Slack

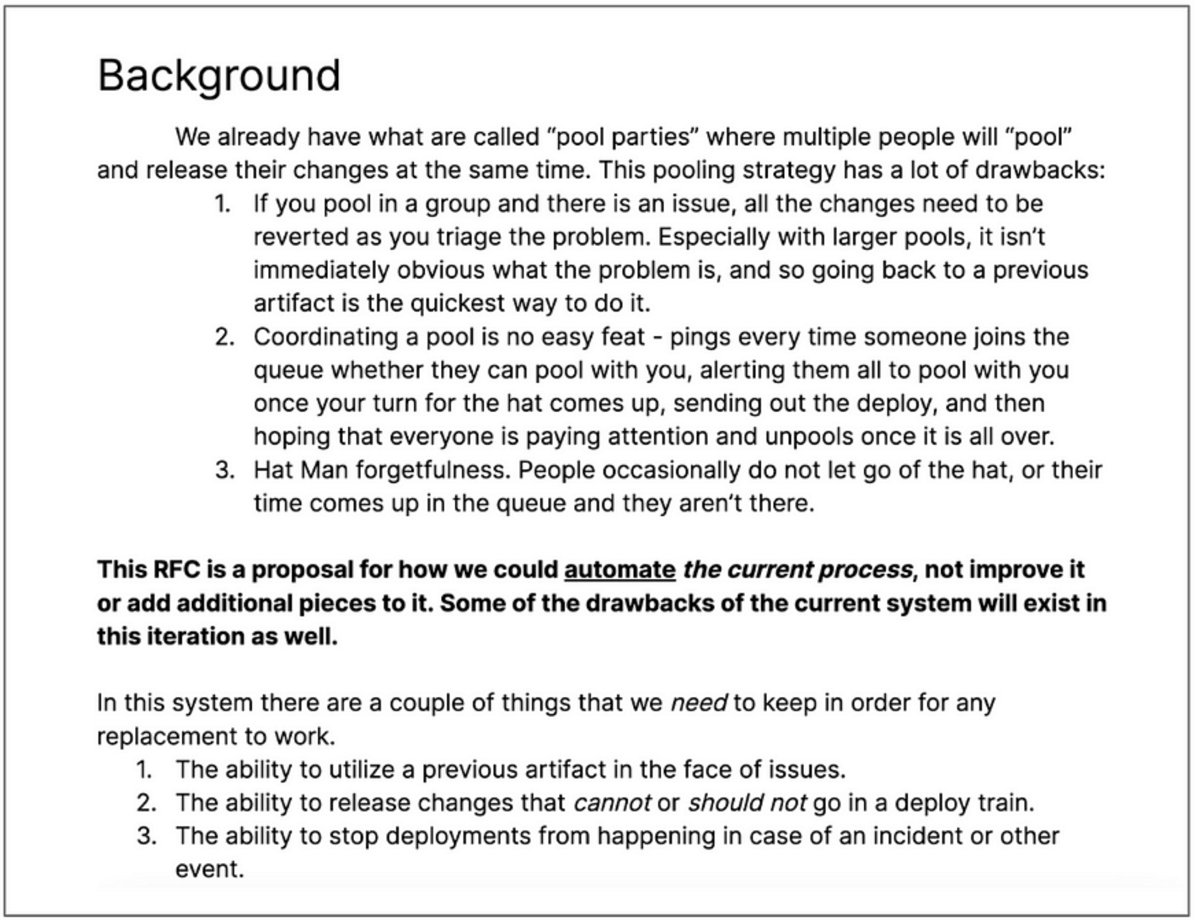
A real example

But of course none of that process will be of any help unless the underlying content is good. Here are a few examples from our 2021 *Deployment Trains* RFC authored by our Velocity SRE team. Iâ€™ve chosen that RFC because you can read about the system in our Thomas the Deployer blog post.

Here was the **problem statement**. Short, sweet, and clear:

> As the engineering organization grows, we continue to have more and more people who wish to deploy every day. We need a better way to deploy changes that doesnâ€™t add a significant amount of time for every ticket.

Here was the **background.** Itâ€™s clear and starts to draw lines around the scope of this RFC.



Background

We already have what are called "pool parties" where multiple people will "pool" and release their changes at the same time. This pooling strategy has a lot of drawbacks:

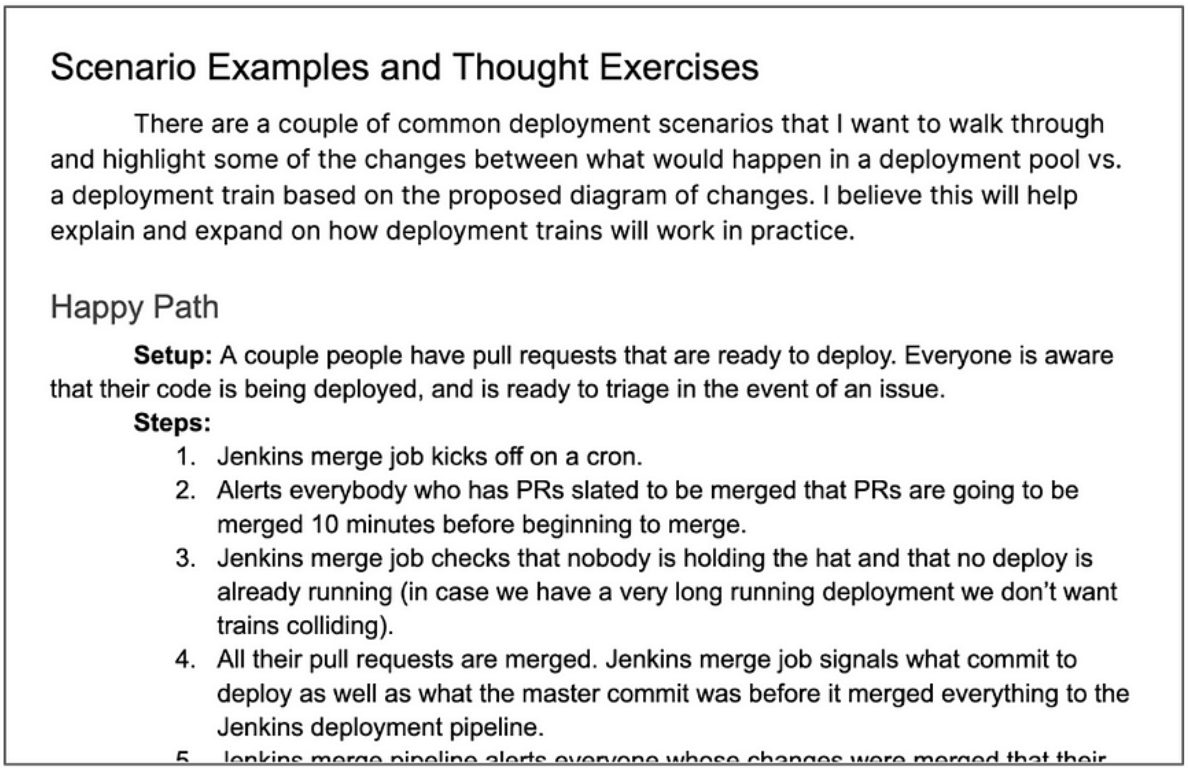
1. If you pool in a group and there is an issue, all the changes need to be reverted as you triage the problem. Especially with larger pools, it isn't immediately obvious what the problem is, and so going back to a previous artifact is the quickest way to do it.
2. Coordinating a pool is no easy feat - pings every time someone joins the queue whether they can pool with you, alerting them all to pool with you once your turn for the hat comes up, sending out the deploy, and then hoping that everyone is paying attention and unpools once it is all over.
3. Hat Man forgetfulness. People occasionally do not let go of the hat, or their time comes up in the queue and they aren't there.

This RFC is a proposal for how we could <u>automate</u> *the current process*, not improve it or add additional pieces to it. Some of the drawbacks of the current system will exist in this iteration as well.

In this system there are a couple of things that we *need* to keep in order for any replacement to work.

1. The ability to utilize a previous artifact in the face of issues.
2. The ability to release changes that *cannot* or *should not* go in a deploy train.
3. The ability to stop deployments from happening in case of an incident or other event.

The following snippet, from a few pages into the document, shows the author using a scenario so that reviewers can easily grasp whatâ€™s being proposed.



Scenario Examples and Thought Exercises

There are a couple of common deployment scenarios that I want to walk through and highlight some of the changes between what would happen in a deployment pool vs. a deployment train based on the proposed diagram of changes. I believe this will help explain and expand on how deployment trains will work in practice.

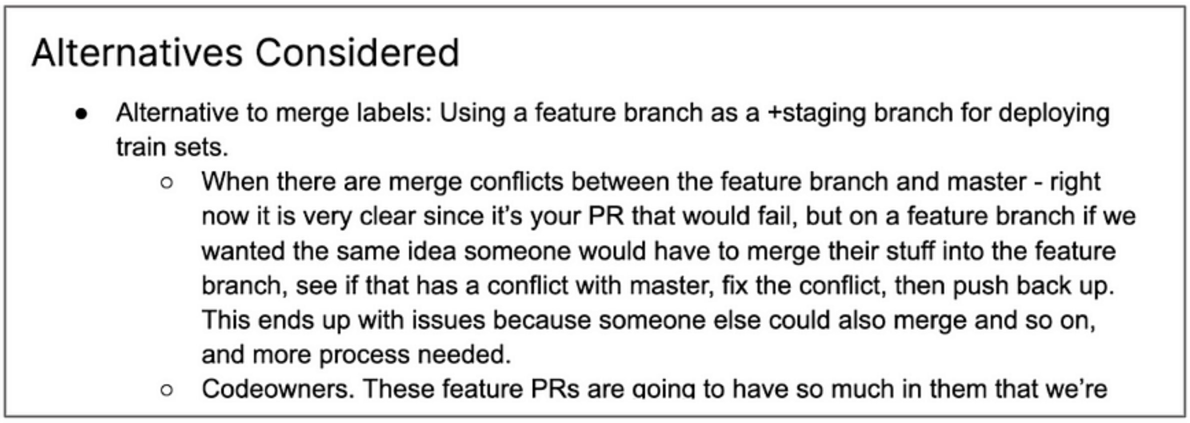
Happy Path

Setup: A couple people have pull requests that are ready to deploy. Everyone is aware that their code is being deployed, and is ready to triage in the event of an issue.

Steps:

1. Jenkins merge job kicks off on a cron.
2. Alerts everybody who has PRs slated to be merged that PRs are going to be merged 10 minutes before beginning to merge.
3. Jenkins merge job checks that nobody is holding the hat and that no deploy is already running (in case we have a very long running deployment we don't want trains colliding).
4. All their pull requests are merged. Jenkins merge job signals what commit to deploy as well as what the master commit was before it merged everything to the Jenkins deployment pipeline.
5. Jenkins merge pipeline alerts everyone whose changes were merged that their

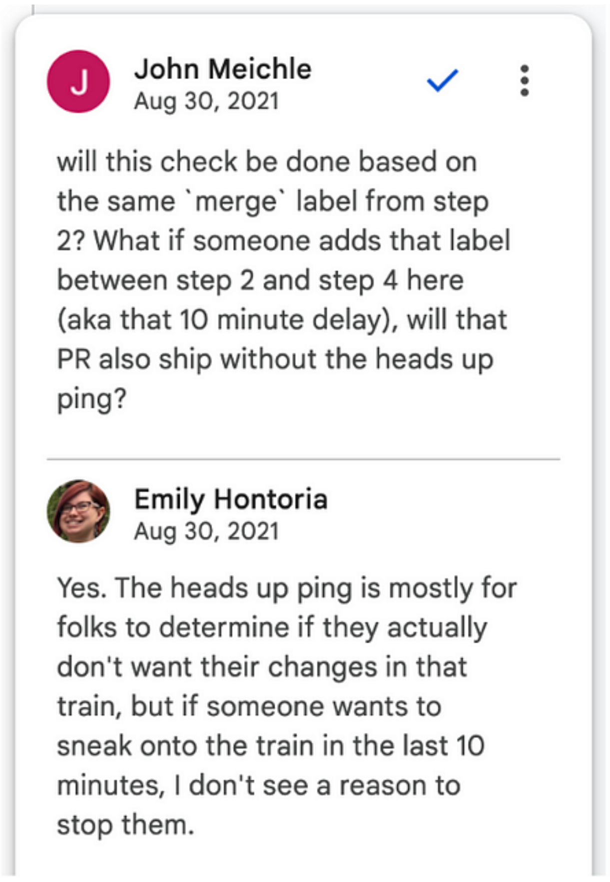
There are lots of ways to do deployment and anyone reading the RFC should be thinking not just about the micro details but if the overall approach makes sense. Thatâ€™s where itâ€™s useful to read about the **alternatives considered** by the authoring team. In this RFC there were six top-level bullets in this section, ranging from minor alternatives such as the one shown in the screenshot below to using commercial and open source tools that the team investigated



Alternatives Considered

- Alternative to merge labels: Using a feature branch as a +staging branch for deploying train sets.
    - When there are merge conflicts between the feature branch and master - right now it is very clear since it's your PR that would fail, but on a feature branch if we wanted the same idea someone would have to merge their stuff into the feature branch, see if that has a conflict with master, fix the conflict, then push back up. This ends up with issues because someone else could also merge and so on, and more process needed.
    - Codeowners. These feature PRs are going to have so much in them that we're

Finally, the comments and discussions are where the best content lives. Theyâ€™re invaluable in heading off confusion before any in-person discussions take place. Example:



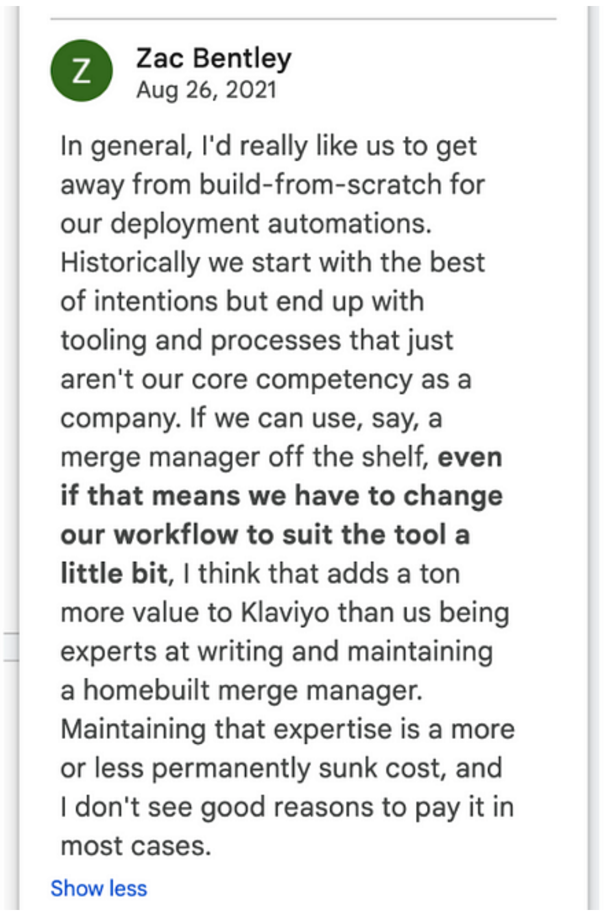
John Meichle
Aug 30, 2021

will this check be done based on the same `merge` label from step 2? What if someone adds that label between step 2 and step 4 here (aka that 10 minute delay), will that PR also ship without the heads up ping?

Emily Hontoria
Aug 30, 2021

Yes. The heads up ping is mostly for folks to determine if they actually don't want their changes in that train, but if someone wants to sneak onto the train in the last 10 minutes, I don't see a reason to stop them.

In the comments, experienced engineers drop gems and words of wisdom such as â€œdonâ€™t test in production.â€� Comments also drive exploration into important philosophical questions. Hereâ€™s an example of the RFC motivating a discussion of the build vs. buy question:



Zac Bentley
Aug 26, 2021

In general, I'd really like us to get away from build-from-scratch for our deployment automations. Historically we start with the best of intentions but end up with tooling and processes that just aren't our core competency as a company. If we can use, say, a merge manager off the shelf, **even if that means we have to change our workflow to suit the tool a little bit,** I think that adds a ton more value to Klaviyo than us being experts at writing and maintaining a homebuilt merge manager. Maintaining that expertise is a more or less permanently sunk cost, and I don't see good reasons to pay it in most cases.

This Deployment Train RFC had 24 reviewers. Keeping as much of the conversation asynchronous as possible helped the reviewers have a productive discussion in person. To organize in person discussions, we have a template section for **Open Questions**. This keeps everyone on task while holding a productive conversation, and it helps everyone involved achieve the goal of developing the strongest plan possible.

Conclusion

I always write something, because with each RFC, I gain more than I put into the process. Writing thoughts down makes those thoughts crisper and helps me plan. Plus, once written, I can solicit feedback at scale. And, most importantly, feedback sometimes reveals pitfalls that I would have otherwise learned about the hard way months later.

At the same time, socializing RFCs unlocks value for Klaviyo as a whole. My colleagues gain knowledge and insight, and since RFCs are open to everyone, people can learn and contribute even if theyâ€™re not involved in a particular system. RFCs are also a type of audit trail of decisions made. (How many times have you come across legacy code and asked â€œbut why?â€�)

A template and a written process arenâ€™t good enough. An individual or group must maintain quality, nurture the RFC culture, and own updating the process as the organization grows. At Klaviyo, thatâ€™s our ARB.

When I joined Klaviyo, I was surprised â€” especially compared to academia â€” at how open we were about information. We were encouraged to work transparently, there were dashboards with metrics all over the office, our calendars were all open, and lots of fascinating business information was shared at our weekly company-wide *By The Numbers* meetings. A few years ago, those ideas were formalized in our official company values, among them **collaborate radically**, **always learning**, **remarkable** work. Our RFC process reflects and reinforces these values. Over 600 RFCs after Johnâ€™s initial one, Iâ€™m proud that writing RFCs has become such an important part of our engineering culture.