Aubrey Chipman &
Roberto Perez Alcolea

NETFLIX

Netflix's Journey to Confident Automated Changes

DPE SUMMIT

September 24-25, 2024 | The Midway | San Francisco

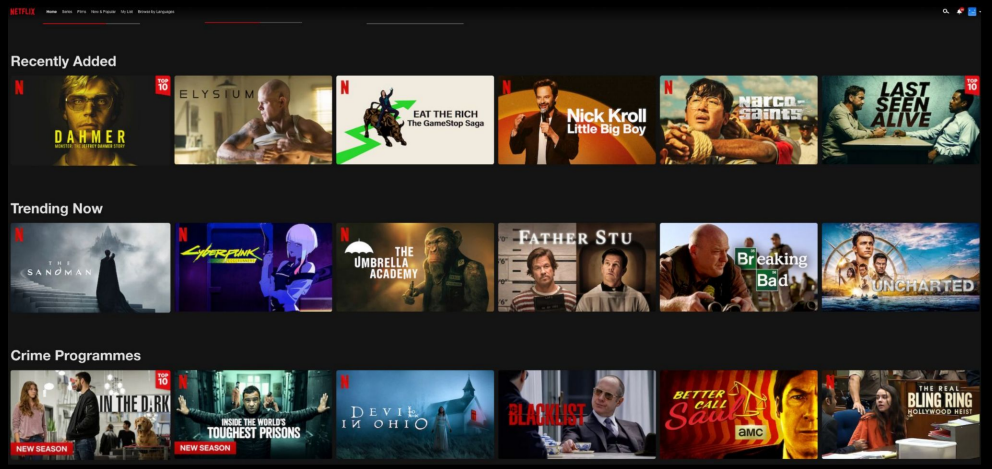
Brought to you by Gradle, Inc

Netflix engineering ecosystem

From...

Image from https://www.wired.com/story/why-are-2-million-people-still-getting-netflix-dvds-by-mail/

To...



NETFLIX

vs

NFL

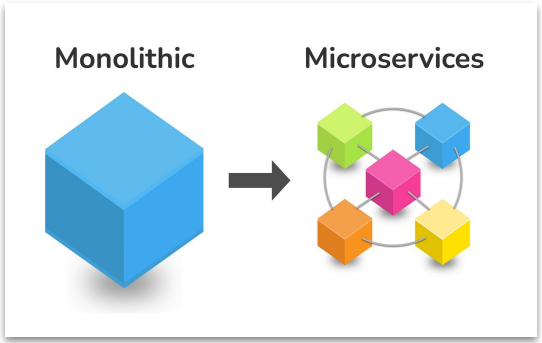
vs

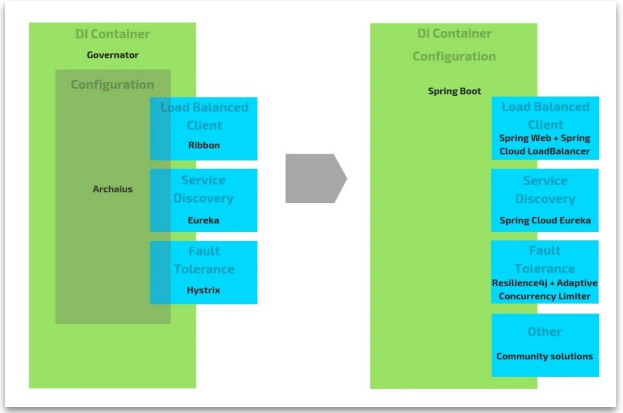
Images from https://about.netflix.com/

LIVE CHRISTMAS DAY

Let’s focus on JVM backend Landscape! How did we get here?

Architecture evolution

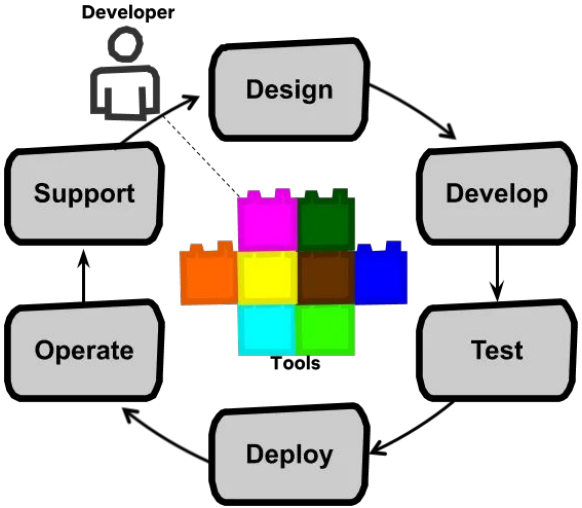


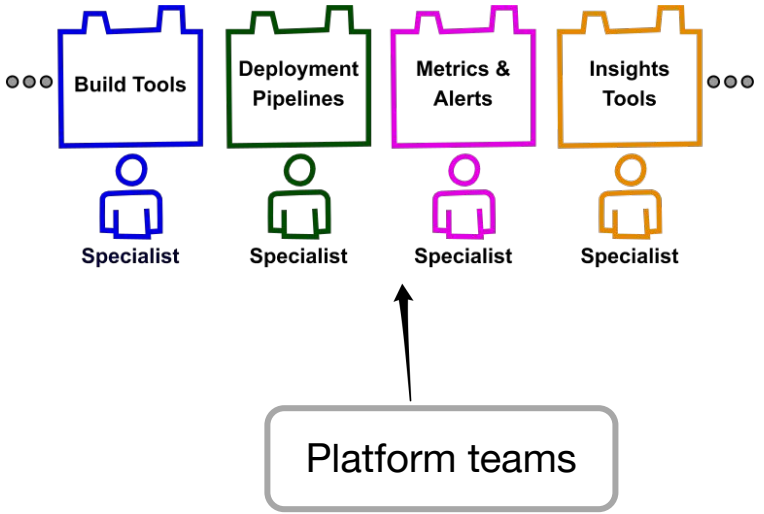


NETFLIX | OSS

Images from https://netflixtechblog.com/full-cycle-developers-at-netflix-a08c31f83249

Full cycle developers
(operate what you build)





Images from https://netflixtechblog.com/full-cycle-developers-at-netflix-a08c31f83249

We have the paved path

Image from https://media.wired.com/photos/5926e050cefba457b079b6b6/master/w_2400%2Cc_limit/GettyImages-166135659.jpg

JVM Backend landscape

- ~4k git repositories
- ~2.8k applications
- ~1.5k (internal) libraries

DEVELOCITY

NETFLIX | OSS

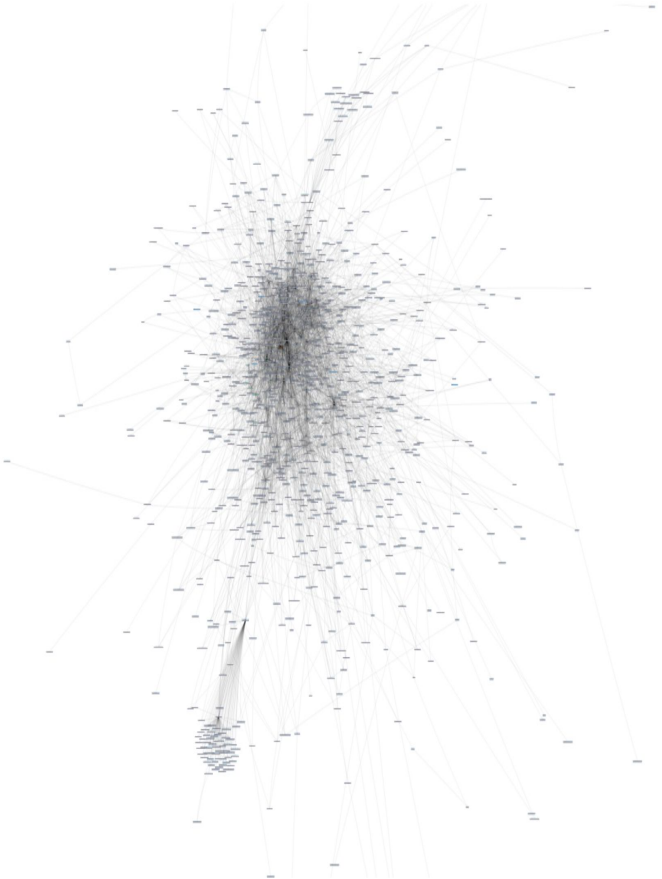
HOLLOW

Gradle

DGS

Images from various external products that we use internally

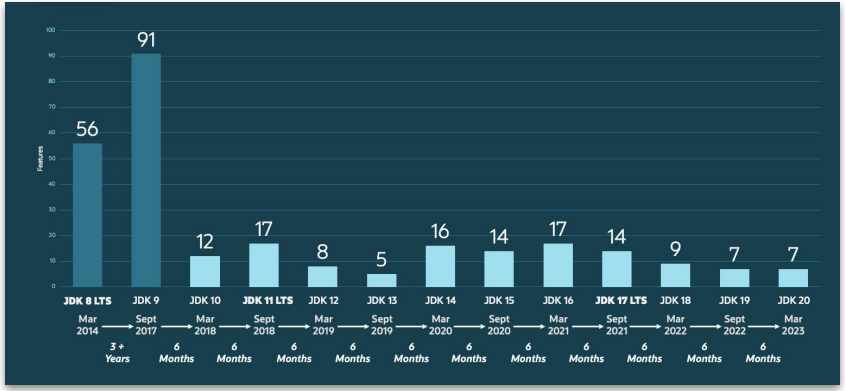
Eventually consistent distributed monolith

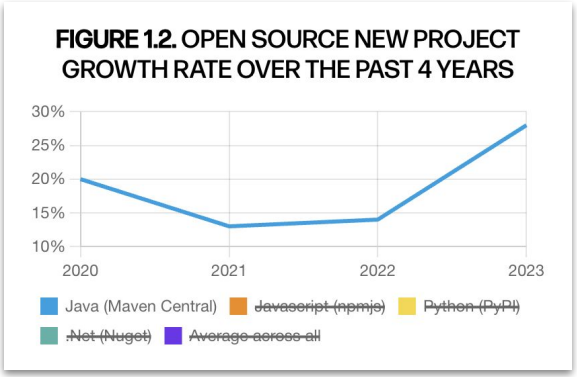


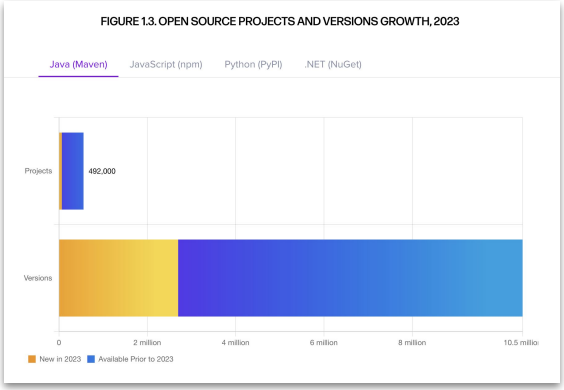
JVM and OSS libraries evolve faster

Spring Boot helps you to create Spring-powered, production-grade applications and services with absolute minimum fuss. It takes an opinionated view of the Spring platform so that new and existing users can quickly get to the bits they need.

Release	Released	OSS support	Commercial Support	Latest
3.3	3 months and 3 weeks ago (23 May 2024)	Ends in 8 months (23 May 2025)	Ends in 1 year and 11 months (23 Aug 2026)	3.3.3 (22 Aug 2024)
3.2	9 months and 4 weeks ago (23 Nov 2023)	Ends in 2 months and 1 week (23 Nov 2024)	Ends in 1 year and 5 months (23 Feb 2026)	3.2.9 (22 Aug 2024)
~~3.1~~	1 year and 4 months ago (18 May 2023)	Ended 4 months ago (18 May 2024)	Ends in 11 months (18 Aug 2025)	~~3.1.12 (23 May 2024)~~
~~3.0~~	1 year and 9 months ago (24 Nov 2022)	Ended 9 months ago (24 Nov 2023)	Ends in 5 months (24 Feb 2025)	~~3.0.13 (23 Nov 2023)~~
~~2.7~~	2 years and 4 months ago (19 May 2022)	Ended 9 months ago (24 Nov 2023)	Ends in 11 months (24 Aug 2025)	~~2.7.18 (23 Nov 2023)~~
~~2.6~~	2 years and 10 months ago (17 Nov 2021)	Ended 1 year and 9 months ago (24 Nov 2022)	Ended 6 months and 3 weeks ago (24 Feb 2024)	~~2.6.15 (18 May 2023)~~



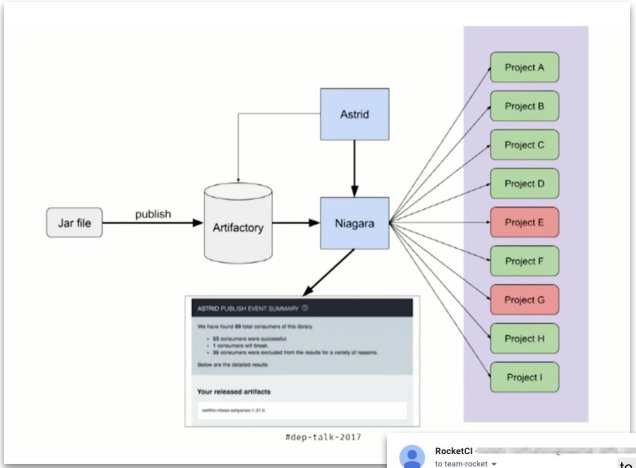


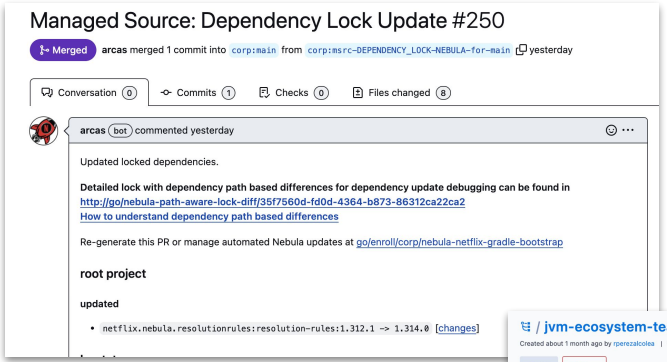


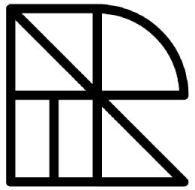
Images from https://www.sonatype.com/state-of-the-software-supply-chain/introduction and https://inside.java/2023/03/21/the-arrival-of-java-20/

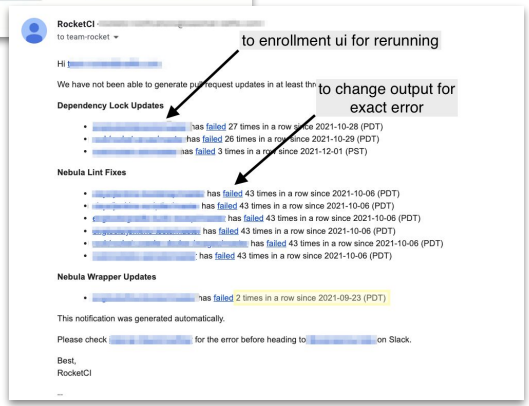
We had to invest on delivering change at scale

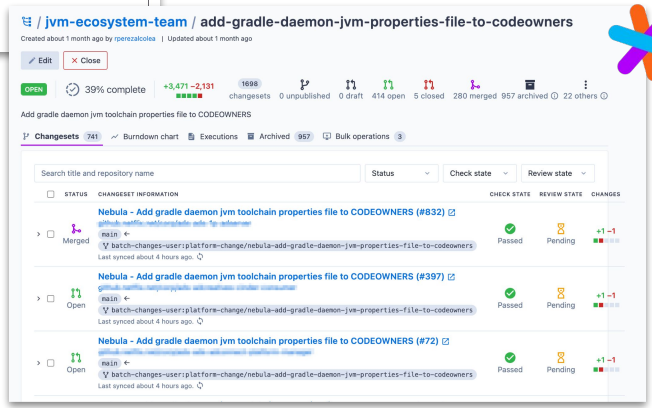
Examples of Investments





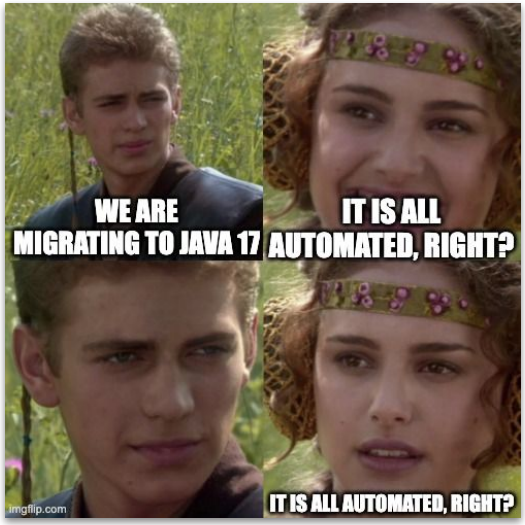






Images from various external products that we use internally, from https://docs.openrewrite.org/, Sourcegraph https://medium.com/@rohanguptatx/sourcegraph-the-future-of-big-code-1de9adbeca36 , and Sourcegraph Combi

And now everything is perfect and we can modernize our stack without engineers interaction!



Turns out delivering code changes is hard!

Making code changes at scale

Changing code

> *Changing code is hard.*
>
> *Making code changes across tens of thousands of git repositories is even harder.*
>
> *Netflix has been able to make progress to addressing this issue, but we need to take this further.*

From a recent Netflix job posting

A response

> *What the heck are you doing that requires changing code in so many repositories? It doesn't sound hard so much as weird. The only scenario that makes sense is if you are changing third-party code. Maybe auto-generating and applying security patches? That could be interesting but the hard part would be getting permission to apply the patches.*

A person's reply to a recent Netflix job posting

Technology is not the only aspect for a successful code change

🔎 Let’s look at a real world example 🔎

Updating application JDKs at scale

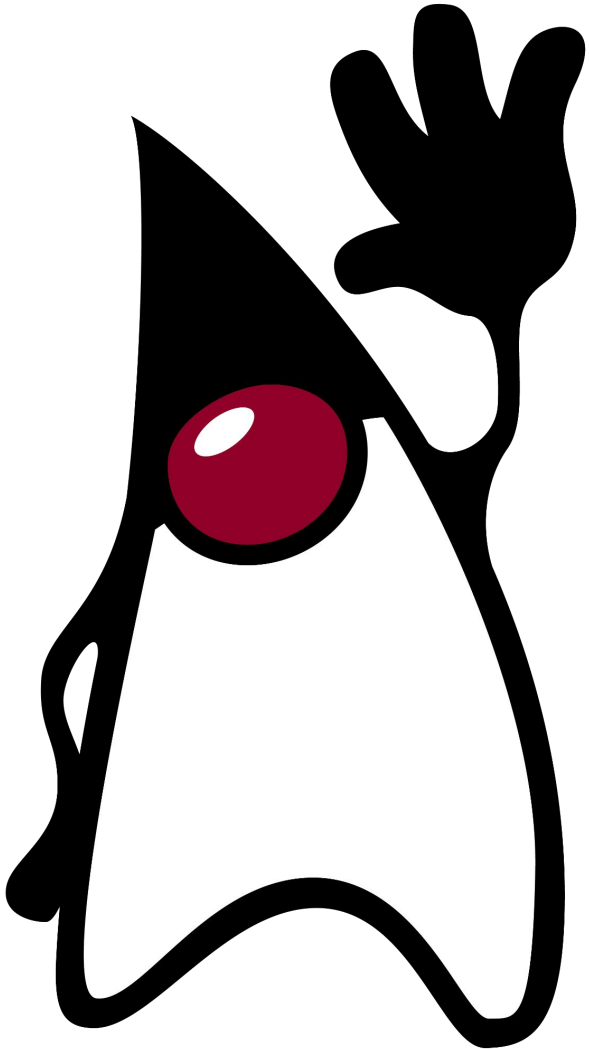


Image from https://wiki.openjdk.org/display/duke/Gallery

A very small code change 

```
java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(17)
    }
}
```

to upgrade application runtime JDK

Image from internal documentation on JDK upgrades and from https://book.git-scm.com/downloads/logos

And we have an internal campaign tool to plan and follow up in this migrations 

Campaign plan

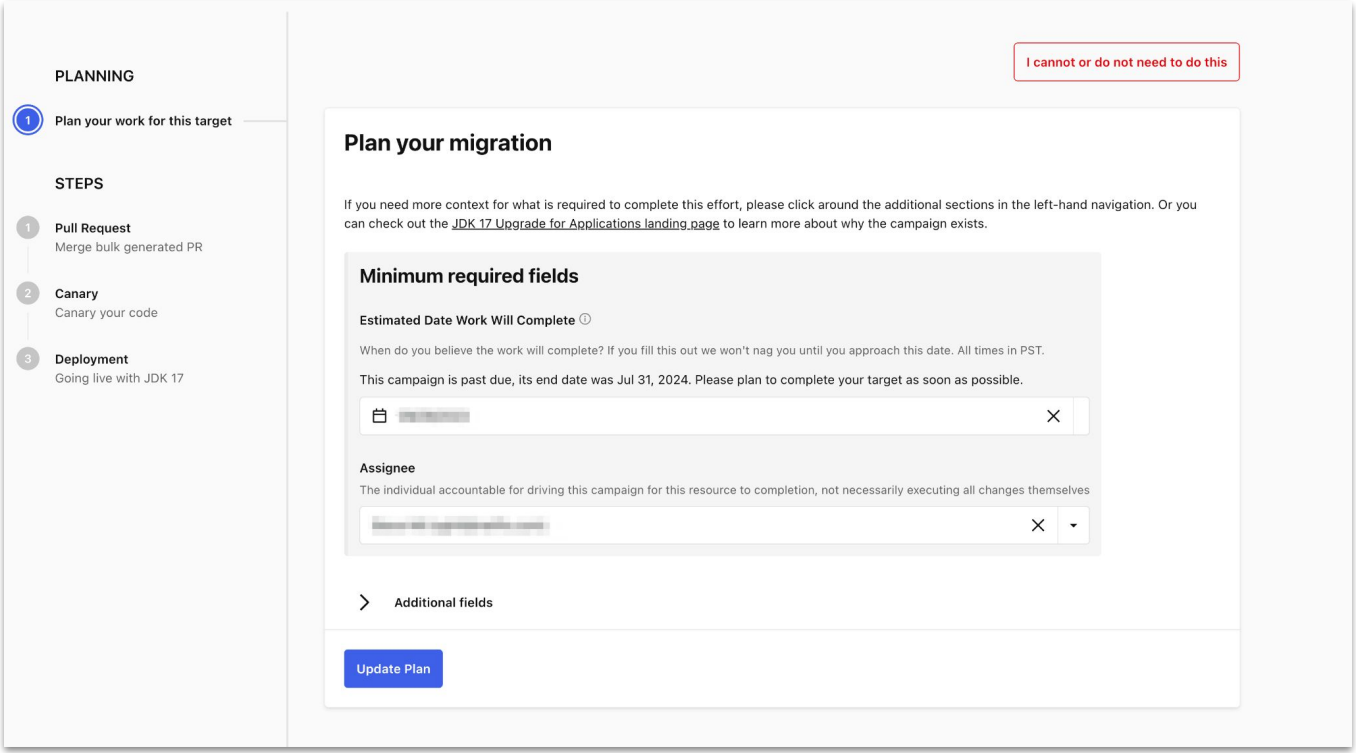
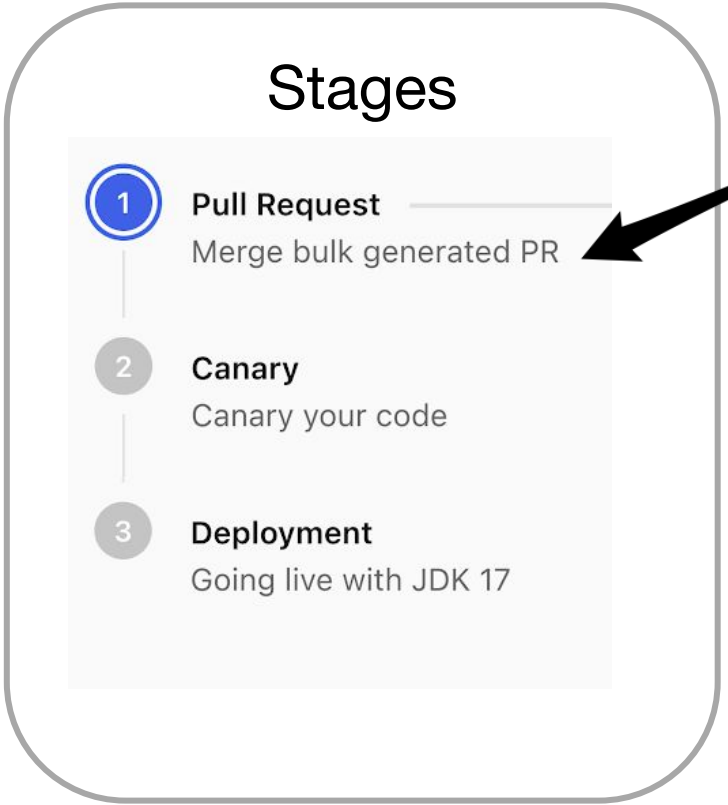


Image from internal campaign UI

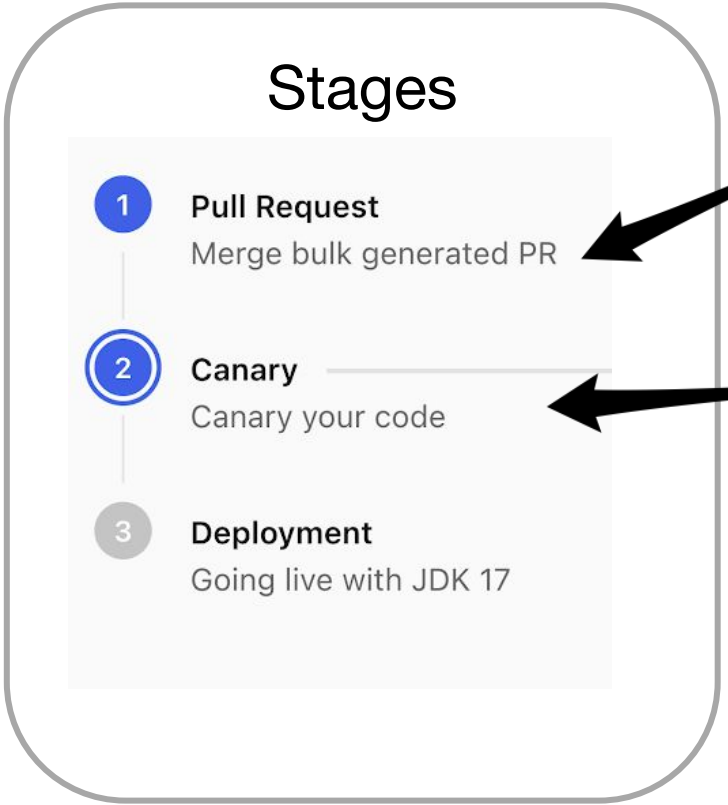
Campaign stages



1. Code change as SCM pull request
2. Run any validation configured for this repository when receiving pull requests

Image from internal campaign UI

Campaign stages

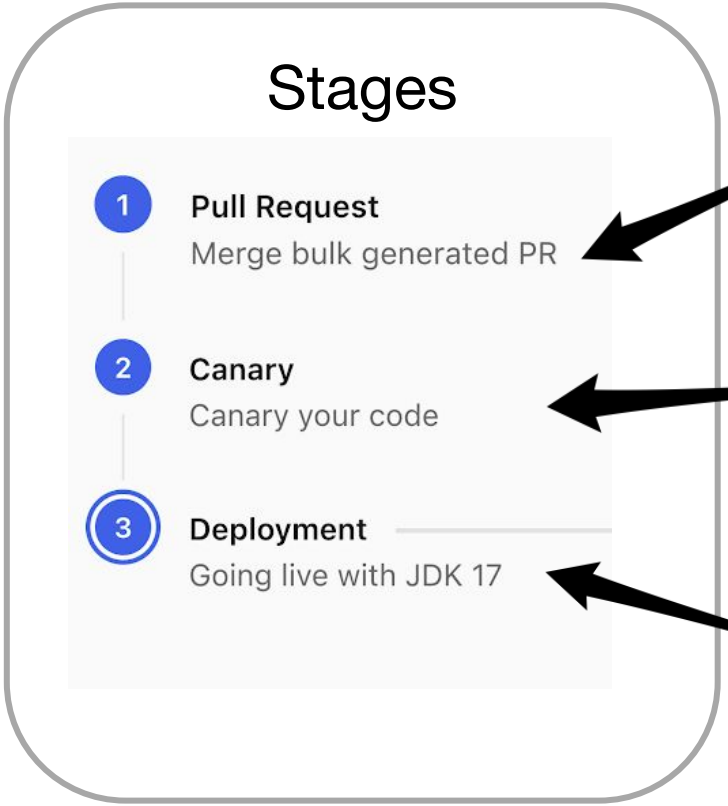


1. Code change as SCM pull request
2. Run any validation configured for this repository when receiving pull requests

3. Deploy application to take some live traffic

Image from internal campaign UI

Campaign stages



1. Code change as SCM pull request
2. Run any validation configured for this repository when receiving pull requests

3. Deploy application to take some live traffic

4. Fully deploy application

Image from internal campaign UI

Very easy…
right?

Unfortunately, IT IS NOT 😔… let’s look at some learnings

Existing verification might no be sufficient

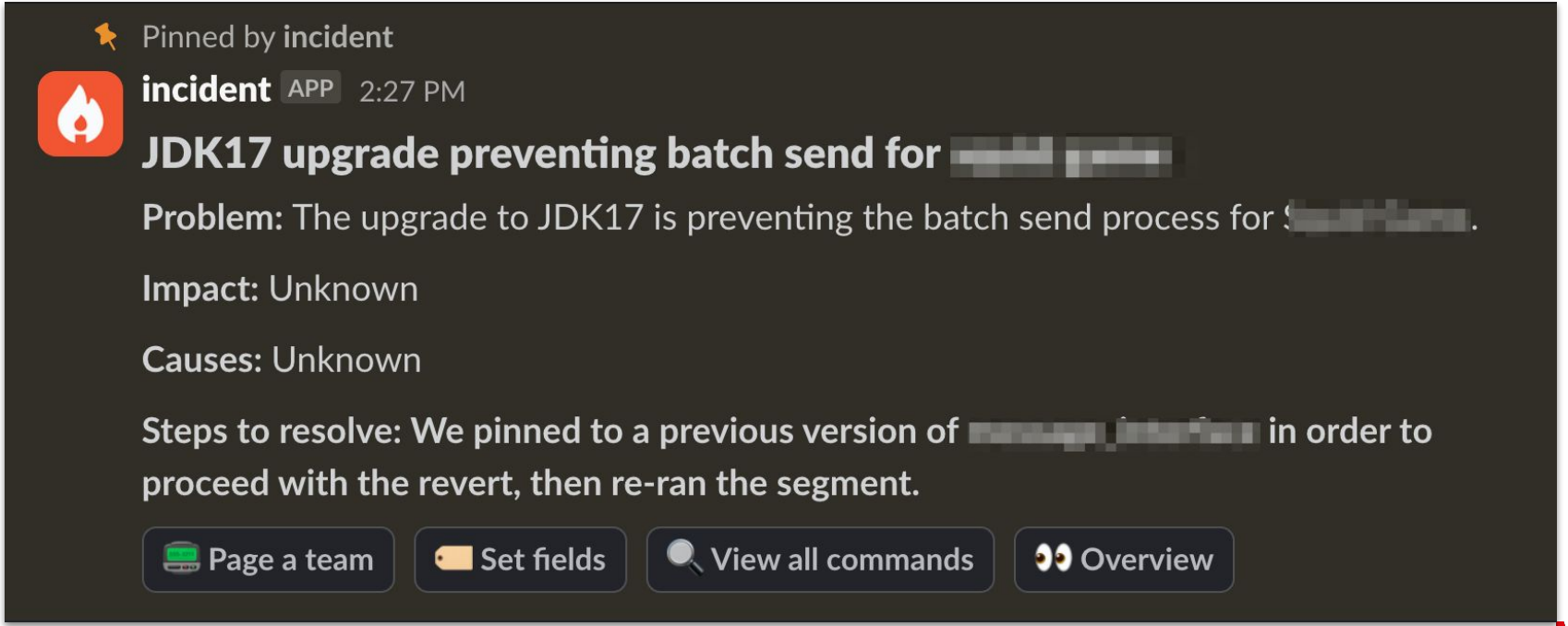
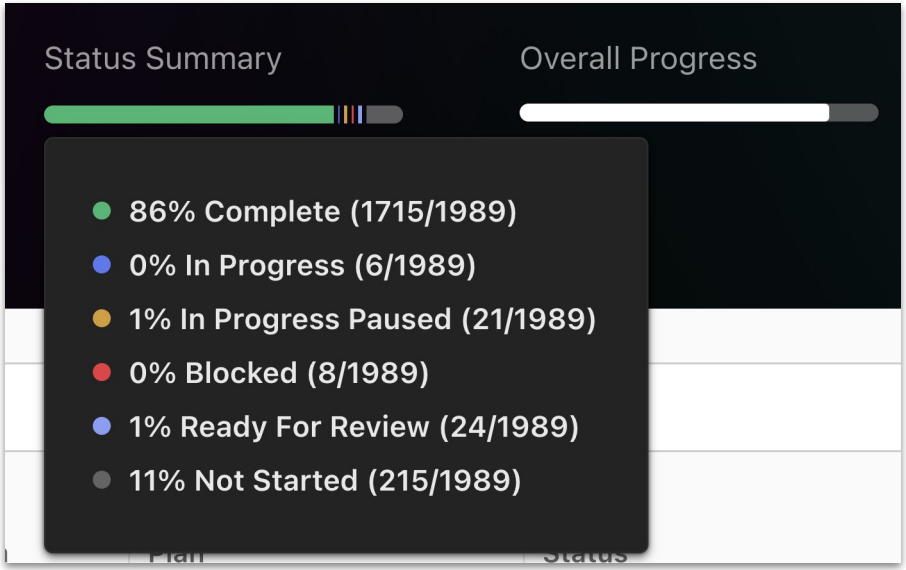


Image from an internal incident notification from the Incident.io Slack app

And human-watched or human-implemented migrations take time

Image from https://lordicon.com/icons/wired/lineal/45-clock-time

The campaign targets (size)



Thousands of targeted applications and each one of them has 1 or more running instances

Image from internal documentation on JDK upgrades

Requires owner engagement

1. Teams have their own priorities

1. Providing automation is great but they still wanted to have control of the change

1. Requires scaling for providing great support and customer experience

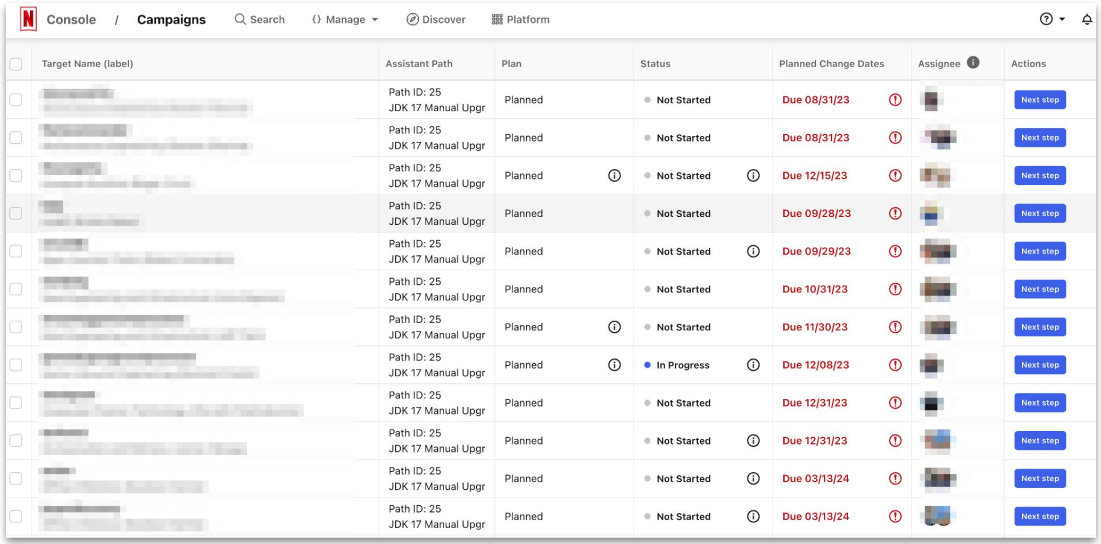


Image from internal campaign UI

And this is just our JDK 17 campaign…

What about more and simultaneous campaigns? 🤔

Image from https://dribbble.com/shots/1815379-Flip-Counter: Flip Counter by Federico Vaccarezza

This could be the campaigns for an engineer

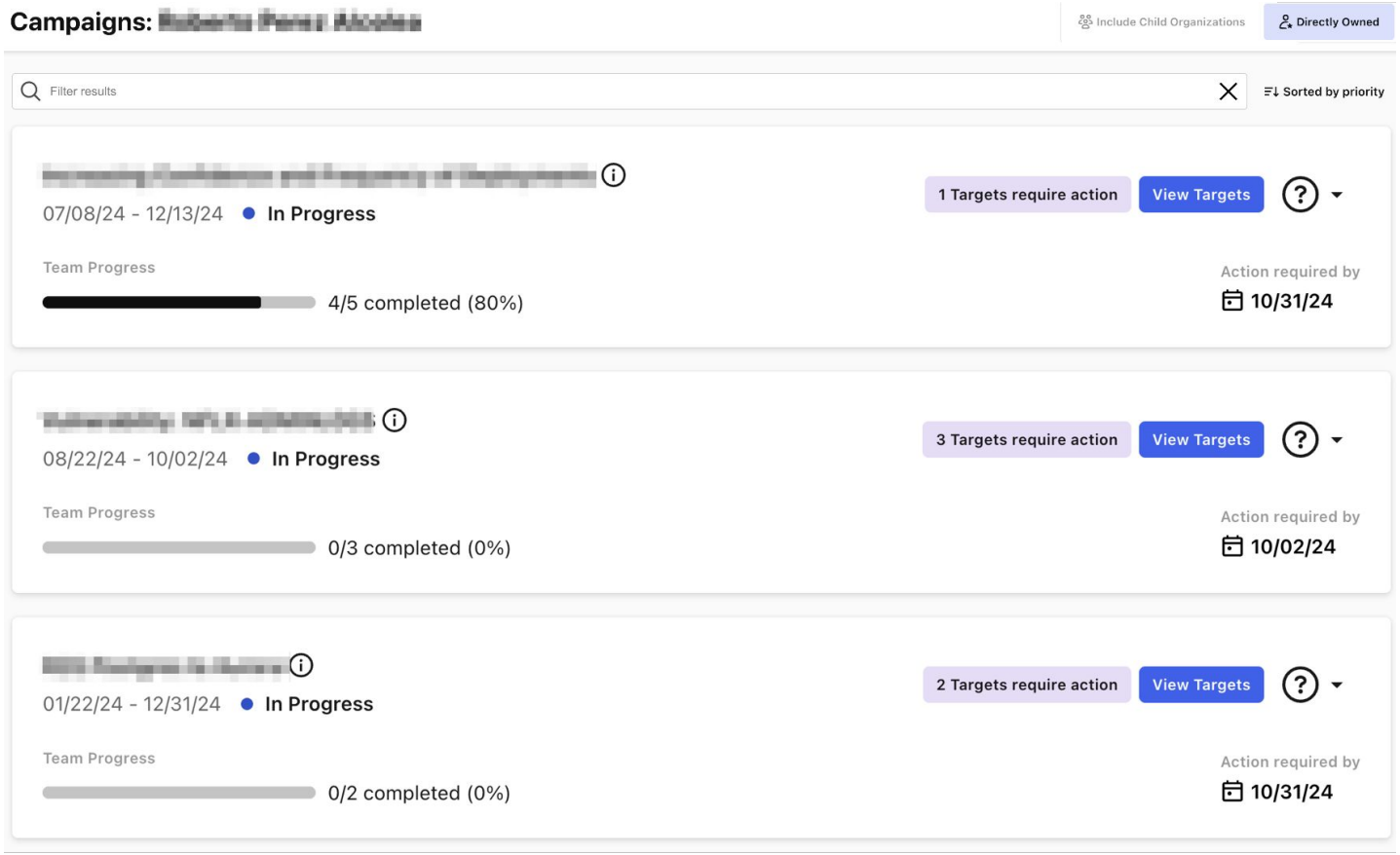


Image from internal campaign UI

Observations: engagement decreases

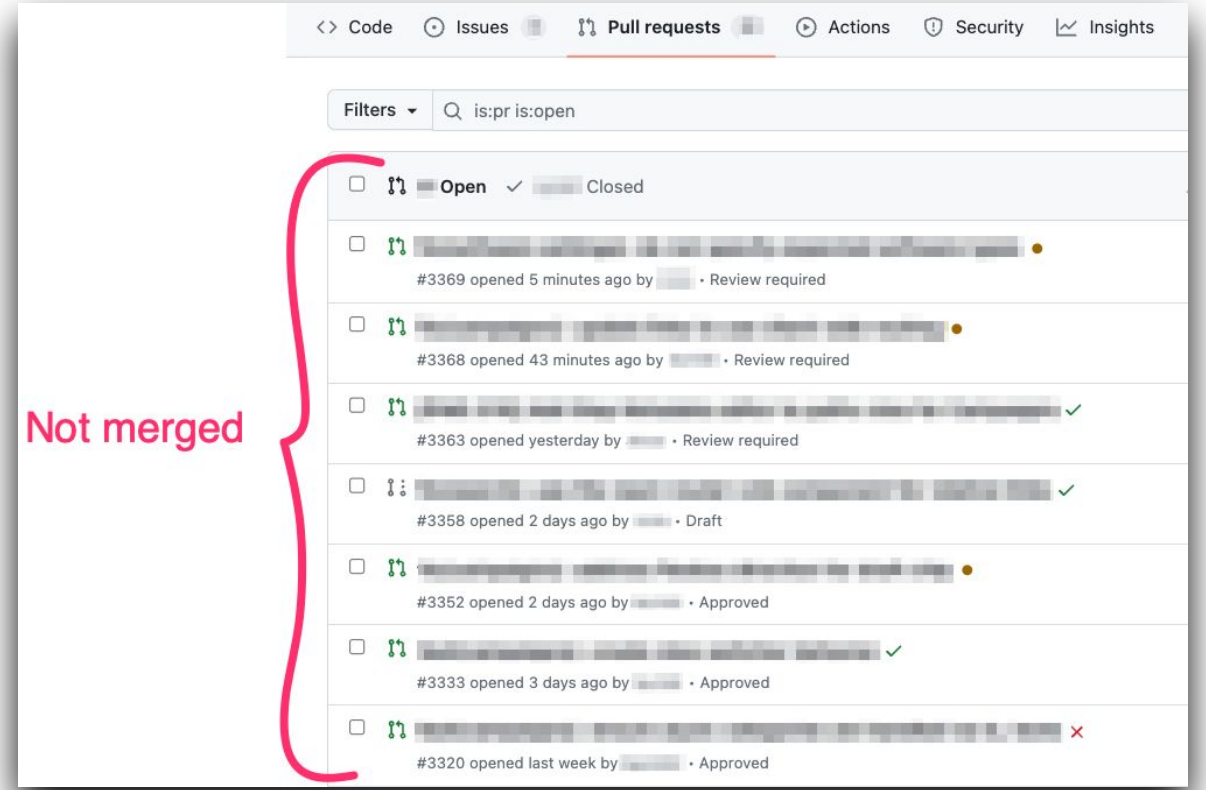


Image from screenshots within SCM tool

Observations: Deployments are not promoted

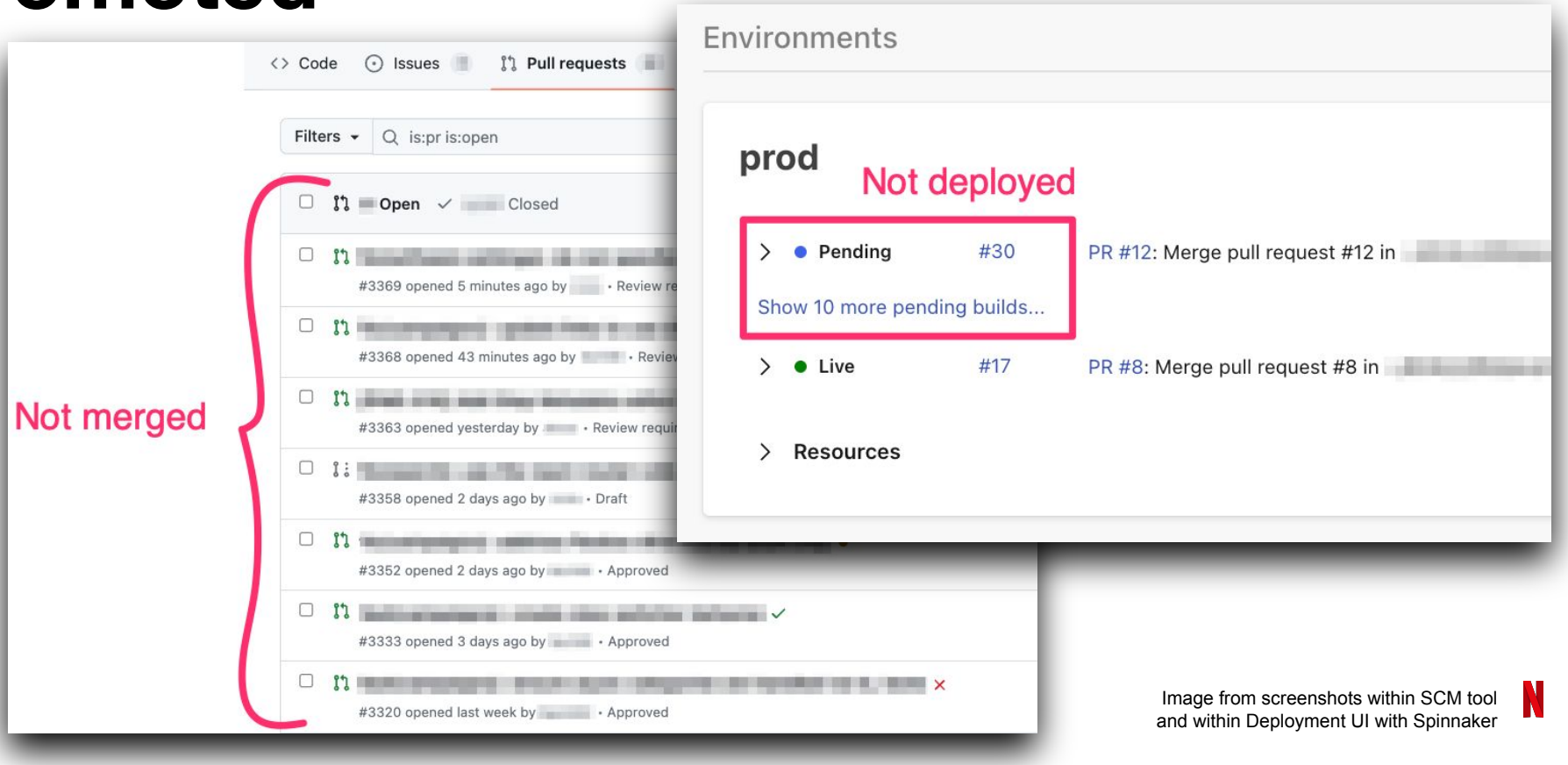


Image from screenshots within SCM tool and within Deployment UI with Spinnaker

Is that enough?

Image generated with ChatGPT

Is that enough?

It is NOT enough

Image generated with ChatGPT

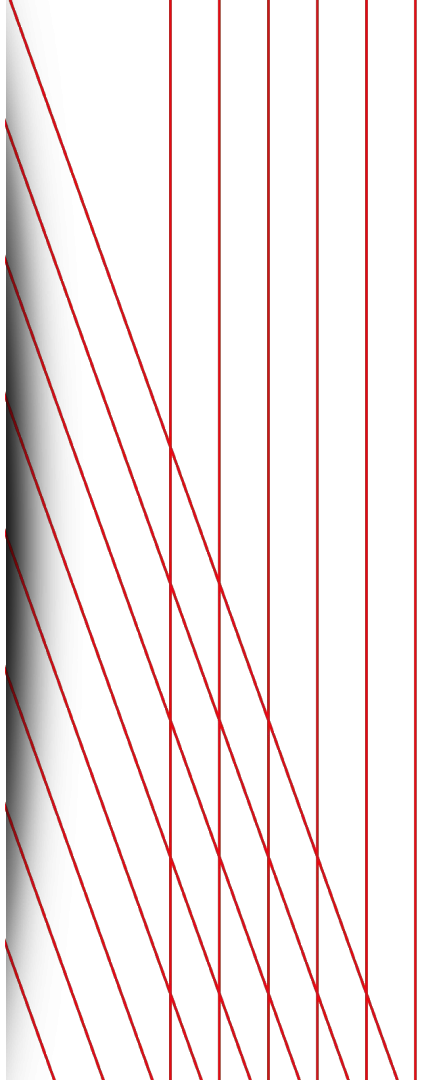
Observation

Although we provided changes at scale,

Our users were not fully taking advantage of this productivity boost.

We probably became good at creating and scheduling changes.

We were still missing something.



What was missing?

Why are people not merging and deploying changes?

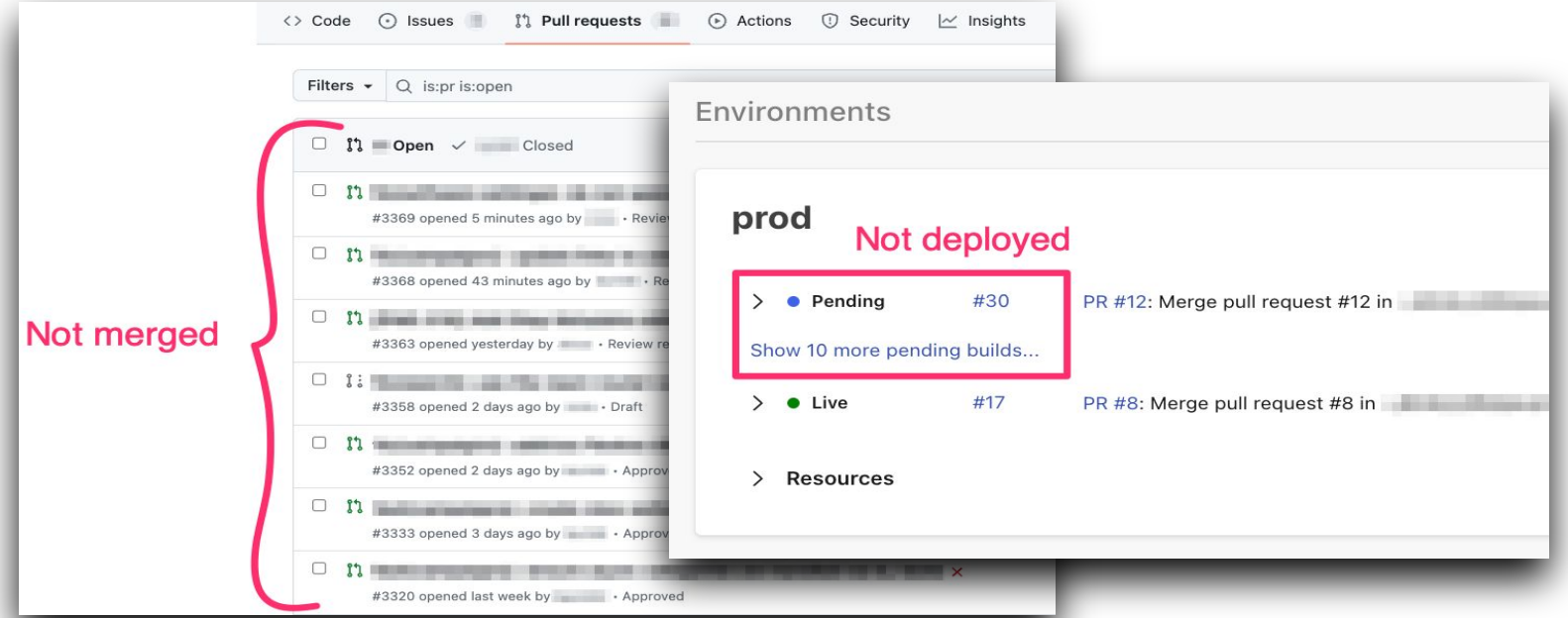


Image from screenshots within SCM tool and within Deployment UI with Spinnaker

Can we get to zero-touch deploys for these campaigns?

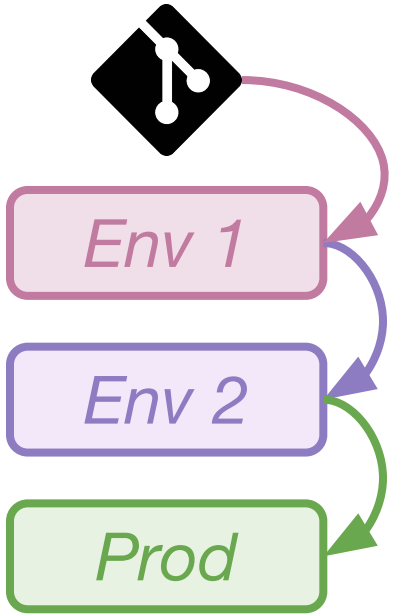


Image from https://book.git-scm.com/downloads/logos

We sent a survey

Image from https://lordicon.com/icons/wired/outline/967-questionnaire

Confidence questions

Automatic commit to deployment workflows with the following types of changes:

1. Application code changes made by your own team?

1. Application code changes made from engineers outside your team?

1. Changes outside of the application code?

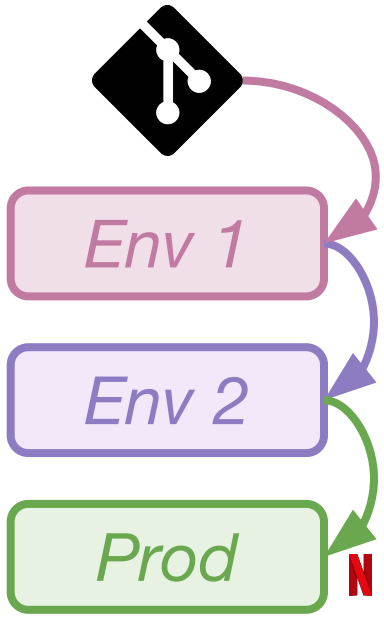
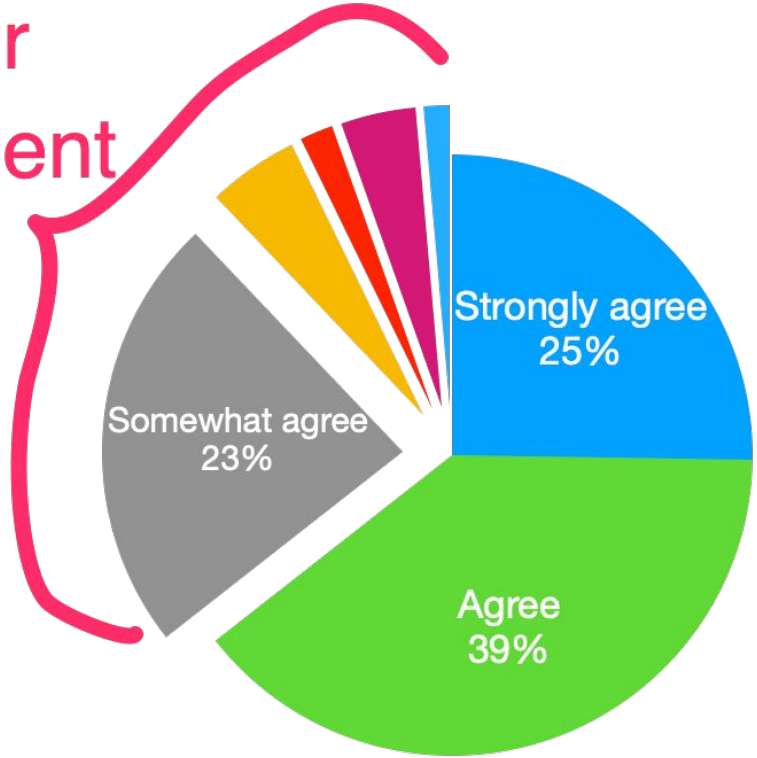


Image from https://lordicon.com/icons/wired/lineal/424-question-bubble and https://book.git-scm.com/downloads/logos

Confidence Rating per app

for changes outside of application code

Targets for improvement



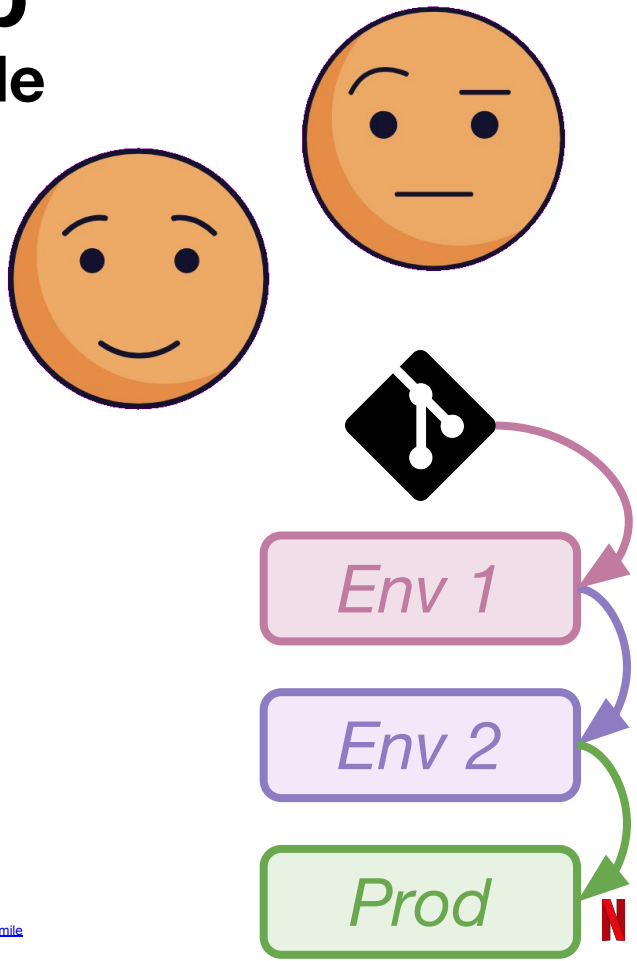
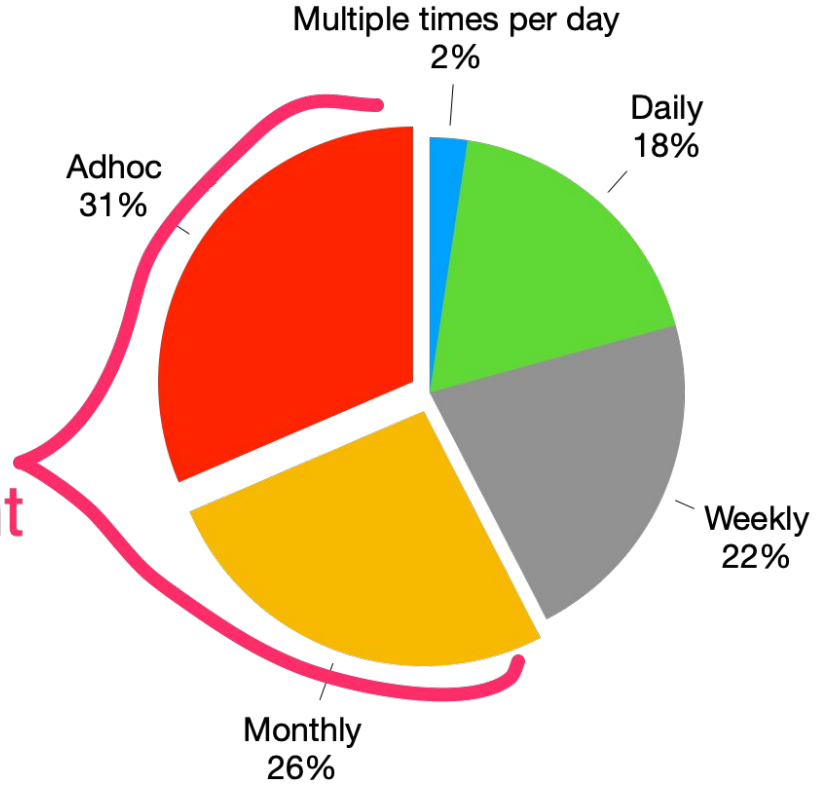


Image from internal data of several thousand survey responses in Q4 2023
Images from https://book.git-scm.com/downloads/logos, from https://lordicon.com/icons/wired/lineal/2340-thinking-skeptic-person-avatar, and from https://lordicon.com/icons/wired/lineal/261-emoji-smile

Deployment Frequency



Targets for improvement

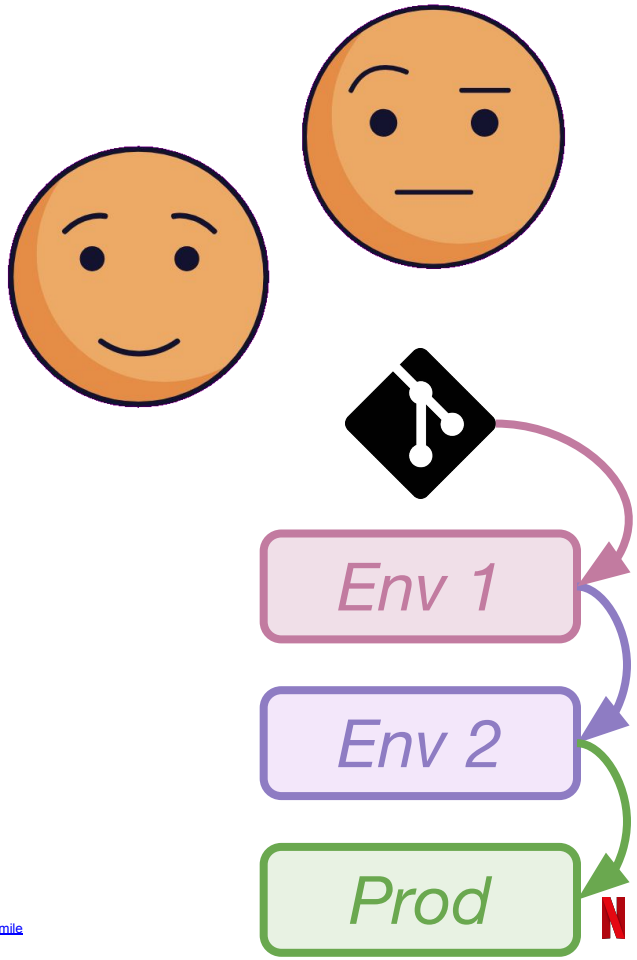


Image from internal data of applicable applications
Images from https://book.git-scm.com/downloads/logos, from https://lordicon.com/icons/wired/lineal/2340-thinking-skeptic-person-avatar, and from https://lordicon.com/icons/wired/lineal/261-emoji-smile

What is preventing people from achieving these outcomes?

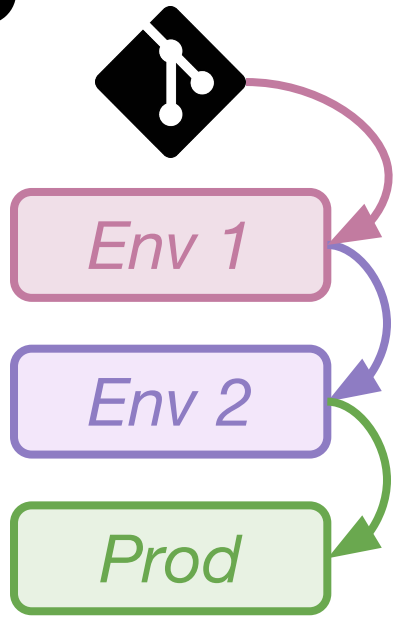


Image from https://book.git-scm.com/downloads/logos

Maybe the paved path had some gravel sections

🤔

Image from https://www.rotim.com/app/uploads/2017/11/Gravenmoer2-e1530023276534.jpg

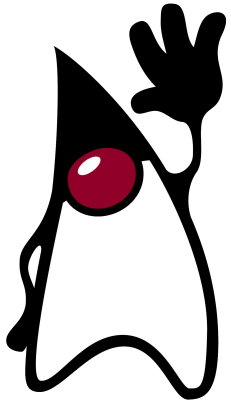
Virtual team

23 passionate individuals from:

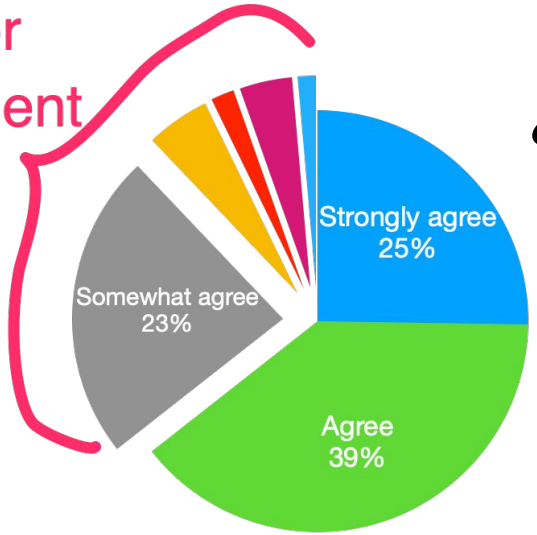
- Platform Engineering teams
- Product Engineering teams
- Product Management
- Subject Matter Experts

Image generated with ChatGPT

Scope

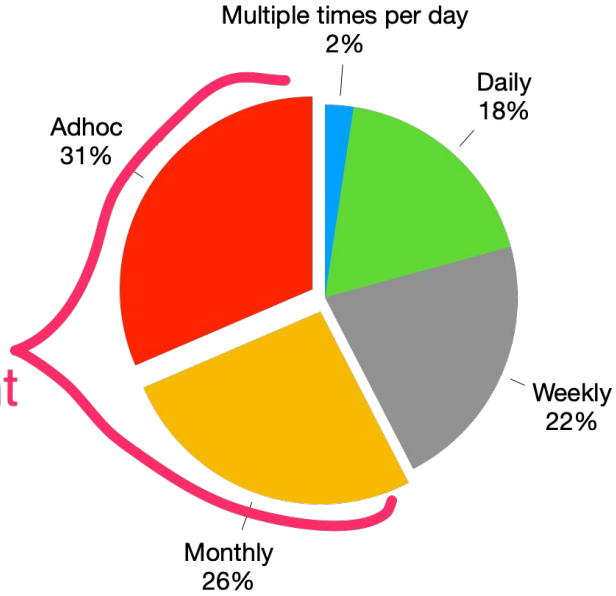


Targets for improvement



Confidence

Targets for improvement



Frequency

Image from internal surveys and gathered data, and from https://wiki.openjdk.org/display/duke/Gallery

Virtual team focus areas

- Clean up abandoned services
- Dependency updates
- Confidence by default
- Frequency
- Delivery
- Monitoring
- Validations
- And more!

Engineering requirements

Image generated with ChatGPT

How long did this whole campaign take for the virtual team?

Some folks worked on this one quarter. Some folks are still working on this 3 quarters later.

?

We validated ✅ these requirements

Validation Guided Interviews

Imagine your same production application *now* conformed to all of those requirements

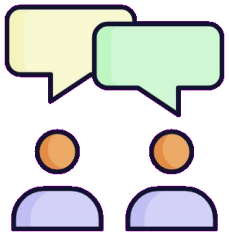
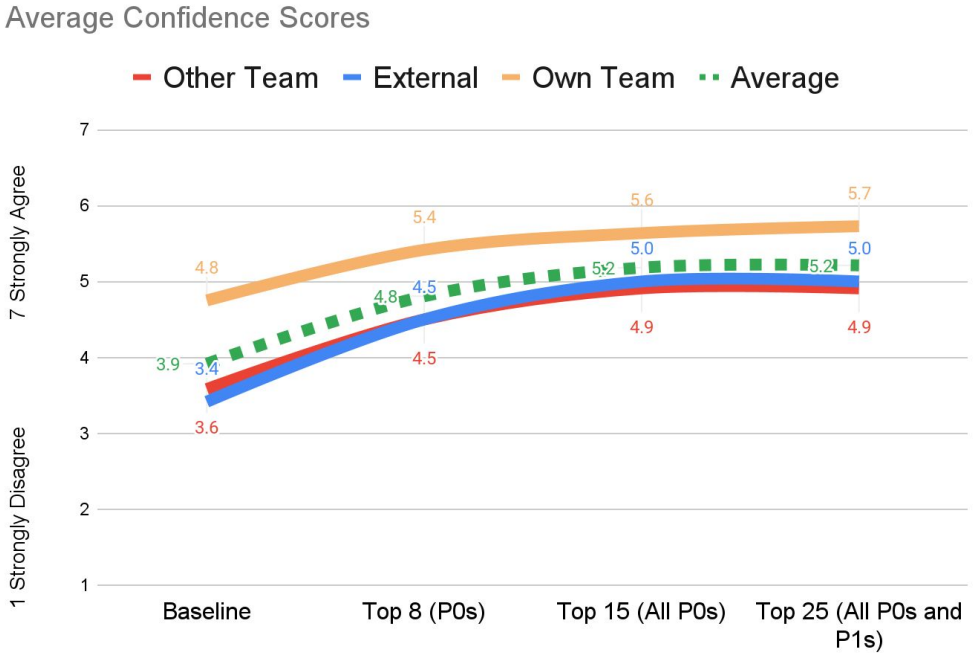


Image from https://lordicon.com/icons/wired/lineal/981-consultation

Requirements Validation



23% *average* change potential

46% *external to the application* change potential

Image from internal engineering surveys and from https://lordicon.com/icons/wired/lineal/981-consultation

What were the requirements?

How did we make them actionable?

Image from https://lordicon.com/icons/wired/lineal/424-question-bubble

Engineering requirements

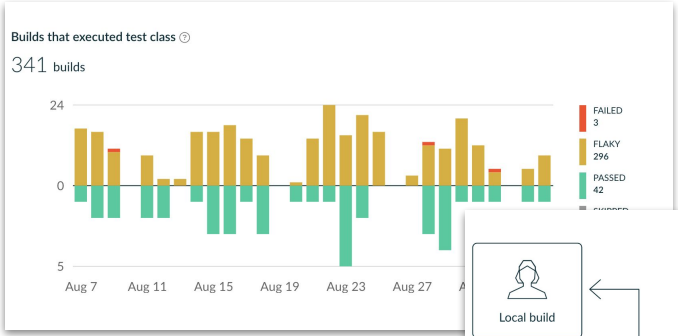
Require improvements in different categories:

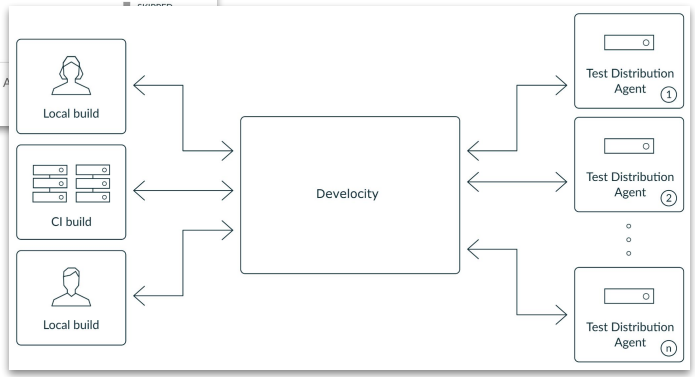
- Application metadata
- Build and Inner Dev Loop Testing infrastructure
- Delivery infrastructure
- Security
- Source Code quality
- Resiliency and Scalability Testing
- Observability
- And more!

Image from https://www.educba.com/requirement-engineering/

Engineering requirements

Java Ecosystem was more mature because of previous investments in the testing experience!





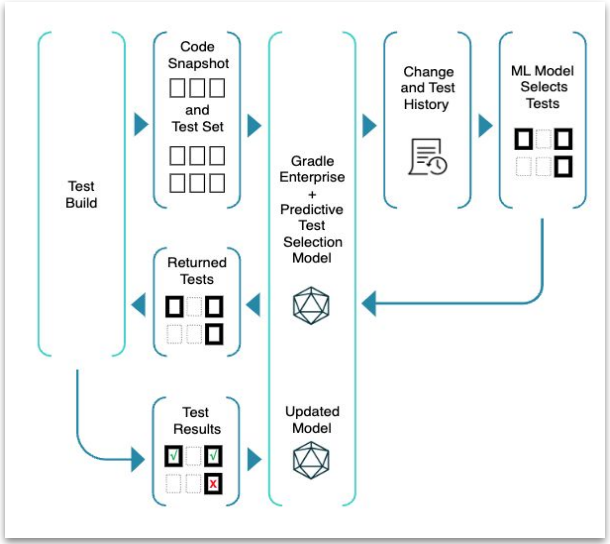


Image from Gradle's Develocity: https://gradle.com/develocity/ , Develocity UI with Netflix build data using Gradle's Develocity, and https://testcontainers.com/cloud/

DPE SUMMIT

San Francisco, CA
September 20-21

How Improving the Testing Experience Goes Beyond Quality: A Dev Productivity Point of View

Roberto Perez Alcolea & Aubrey Chipman
Developer Productivity Team
Netflix

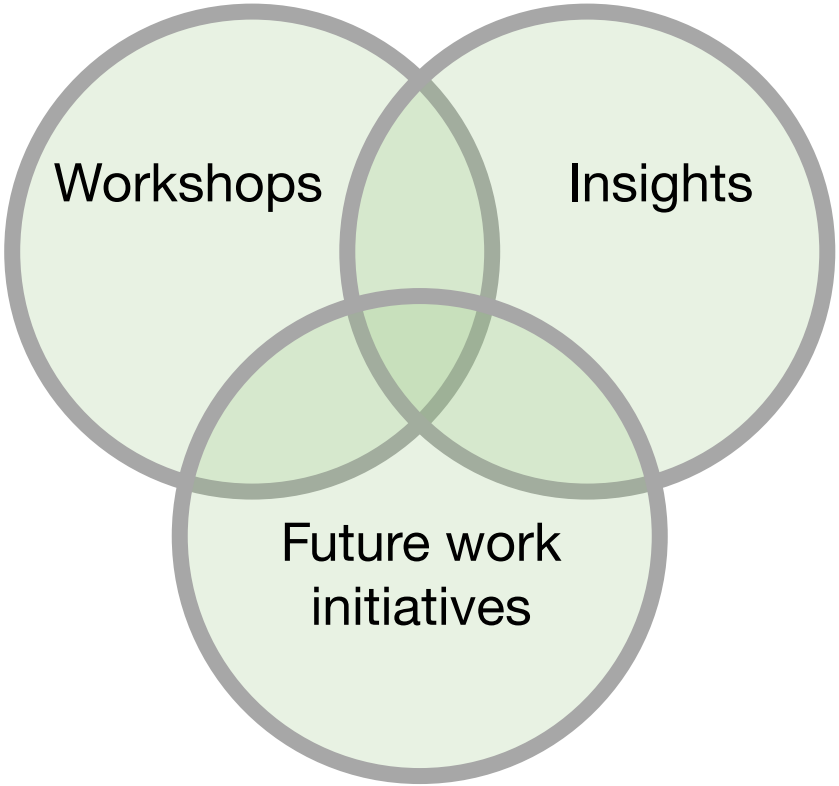
Major areas of investment

- Testing developer experience
- Language agnostic tooling
- Language maturity models
- Continuous Integration
- Production Always Ready initiative
- Tech debt as innovation

Key focus area:
The importance of adapting to changes

Image generated with ChatGPT

Engineering investments



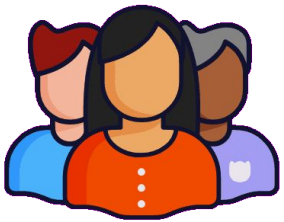
Shifting the company culture

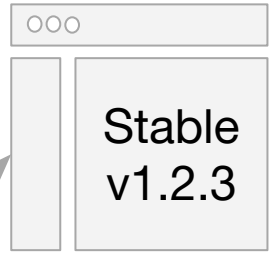
Image from https://myva360.com/blog/the-9-types-of-company-culture

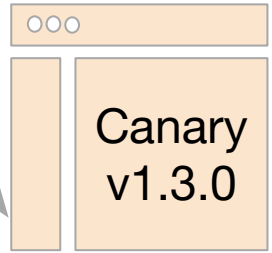
Workshops

Testing

Auto-deployment configuration



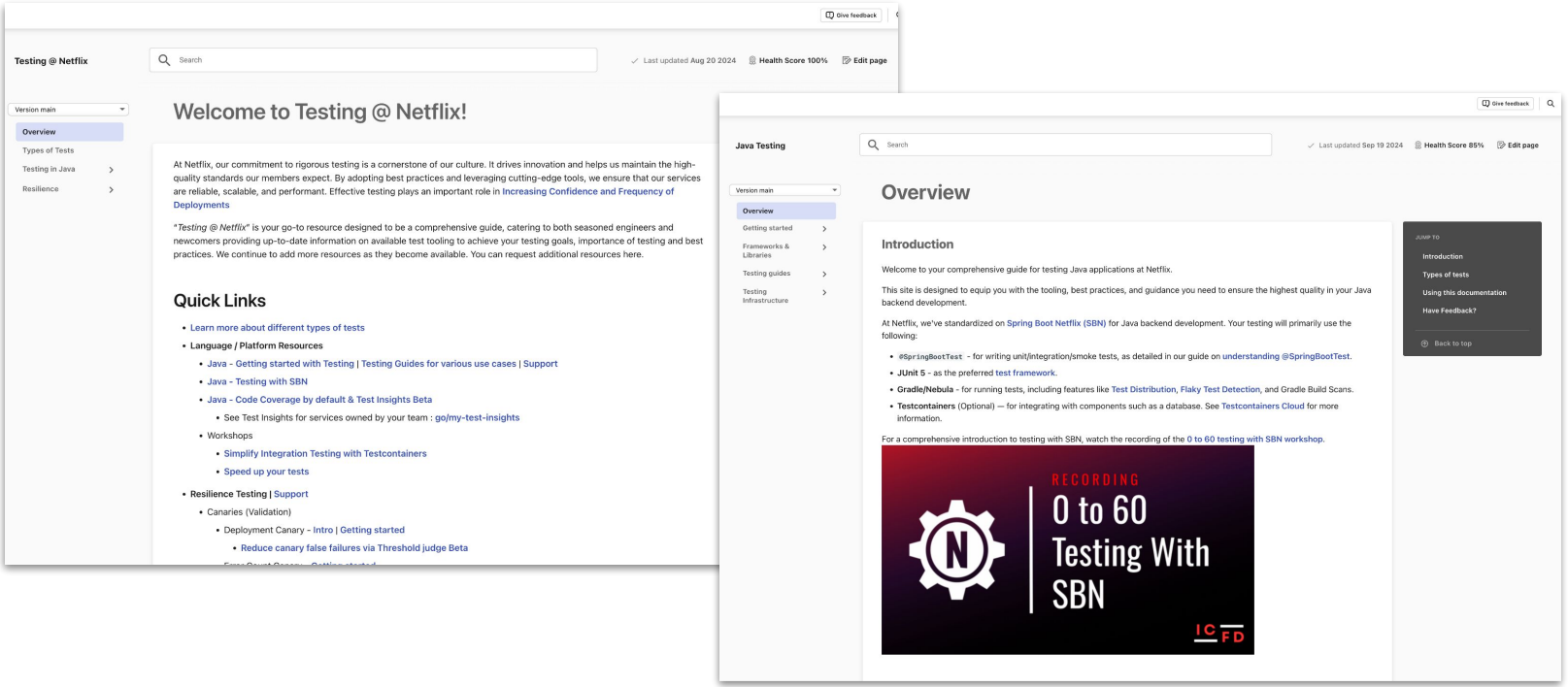




Monitoring

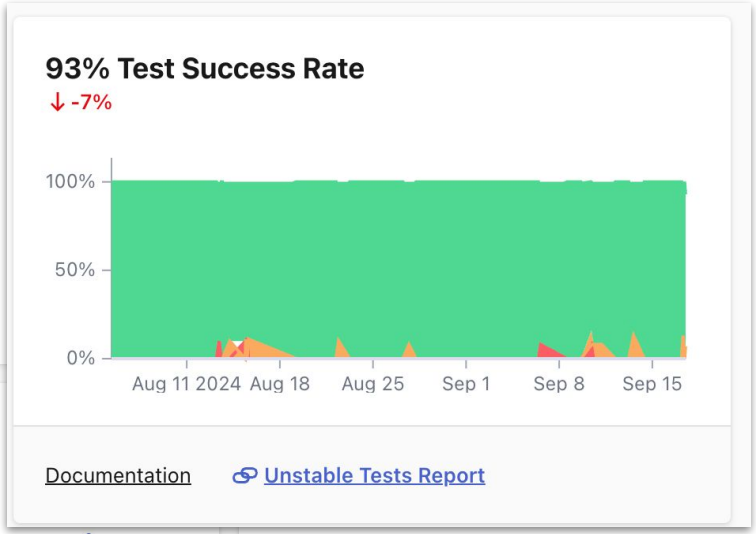
Images from https://lordicon.com/icons/wired/lineal/275-female-and-two-males , https://lordicon.com/icons/wired/lineal/962-social-media-marketing , https://lordicon.com/icons/wired/lineal/1142-form-sheet-questionnaire

Documentation



Improving documentation increased our CSAT considerably!

Testing insights



Images from internal UI backed by internal data

Testing Insights

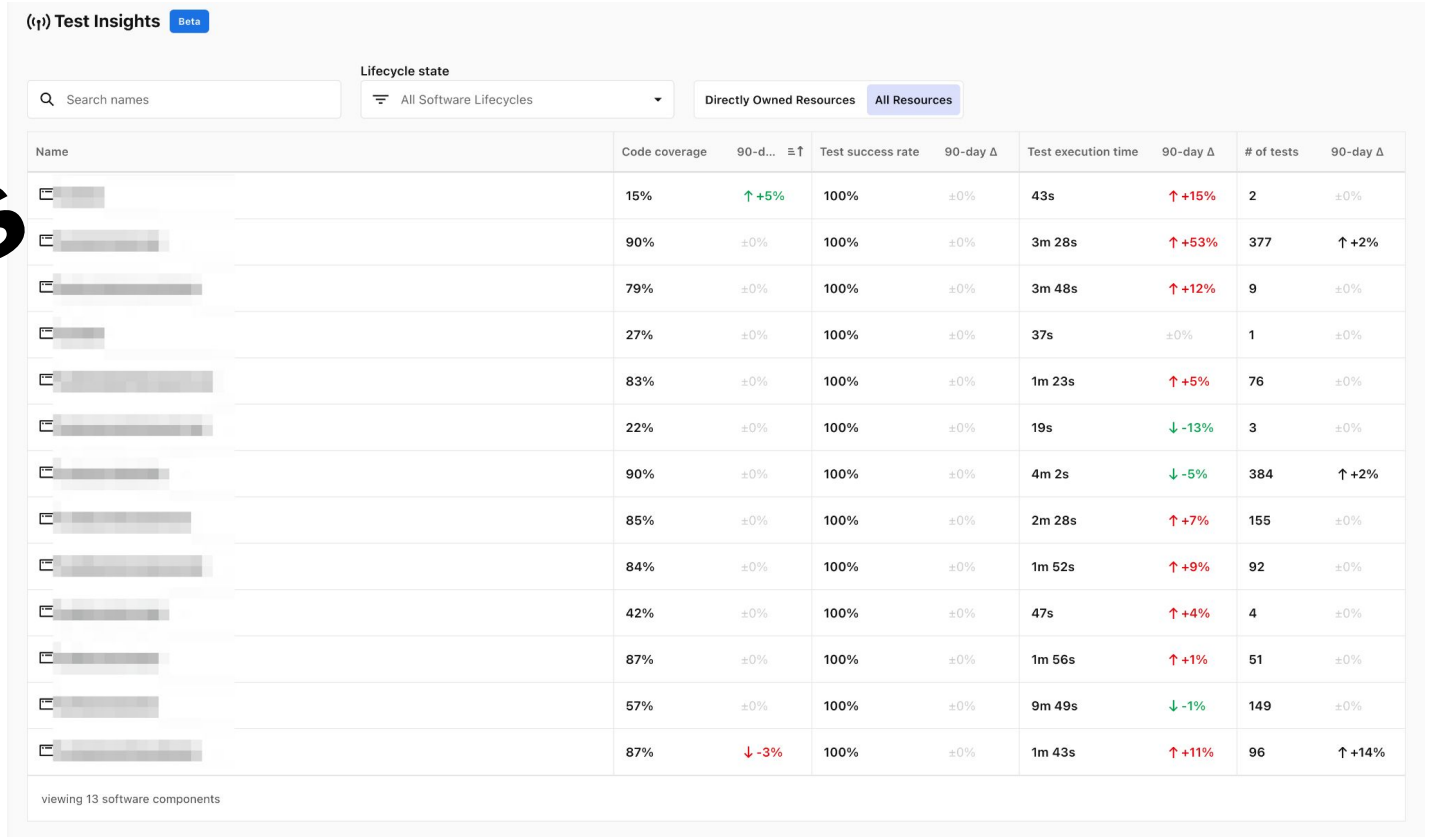


Image from internal UI backed by internal data

Delivery notifications

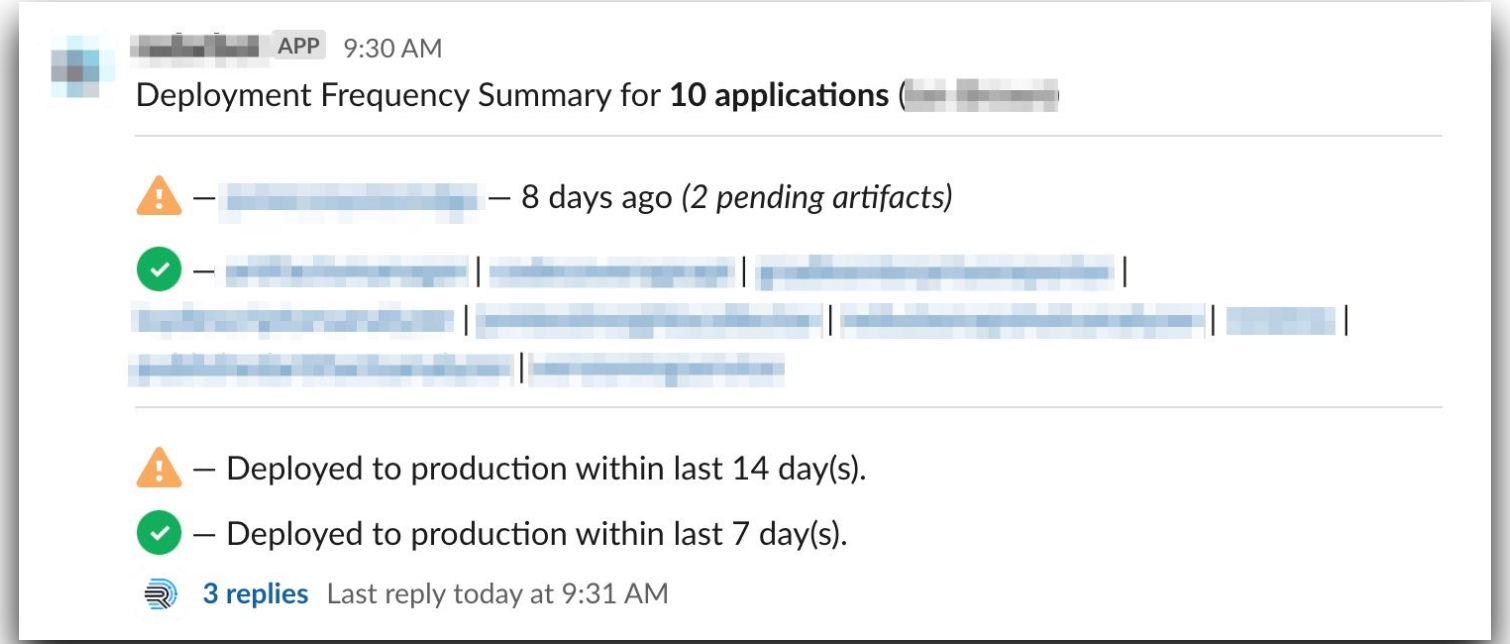


Image from internal Slack notifications backed by internal data

Adopting new practices and tools

Our strategy

- Instrumentation for free!
- Don’t reinvent the wheel
- Do more for customers (managed)
- Heavy focus on cultural change, learning and education
- Don’t pursue major architectural changes
- Leverage our campaign tooling to increase confidence and frequency of deployments

Images from https://productled.com/blog/netflix-product-strategy-2020

The ICFD Campaign

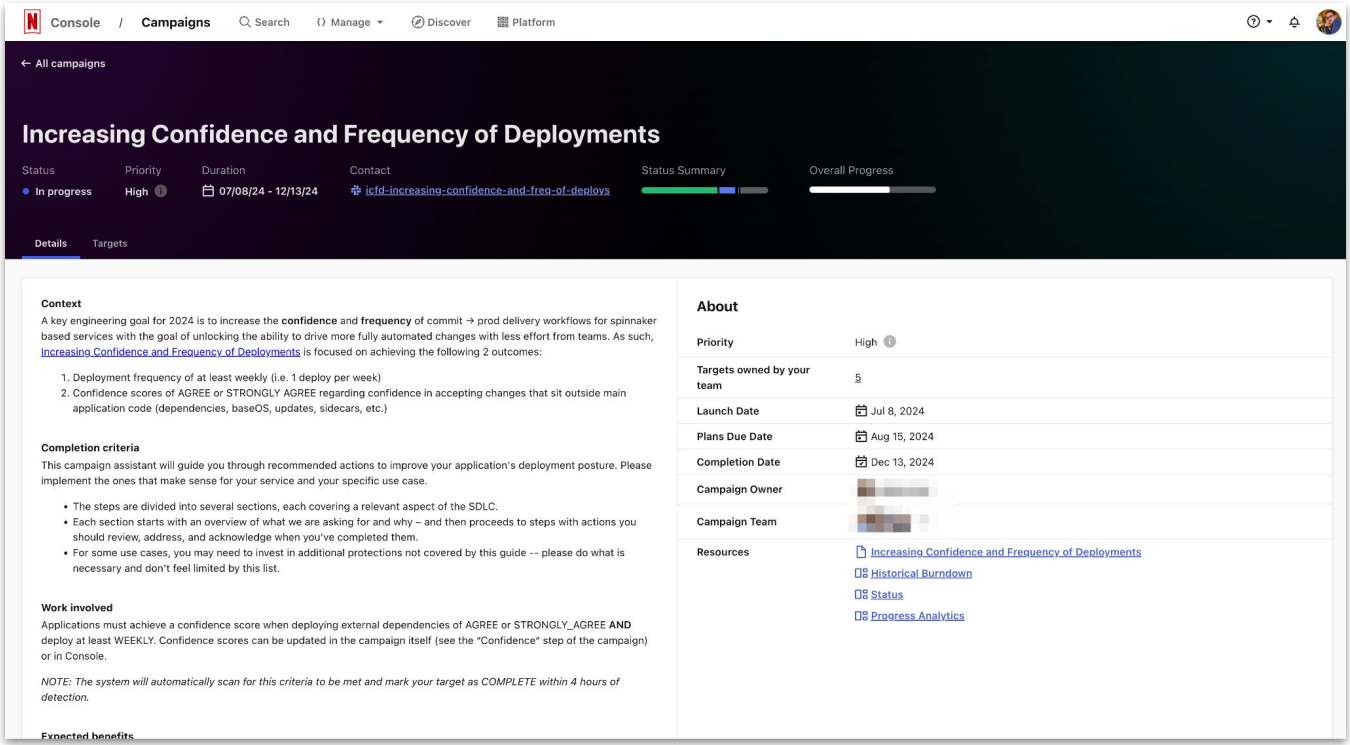
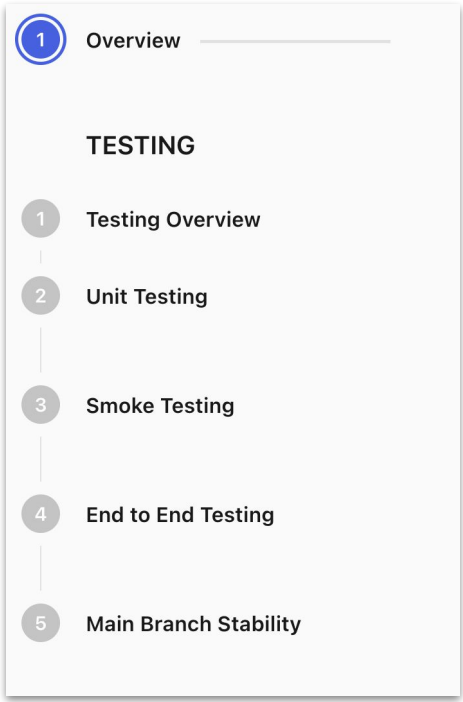
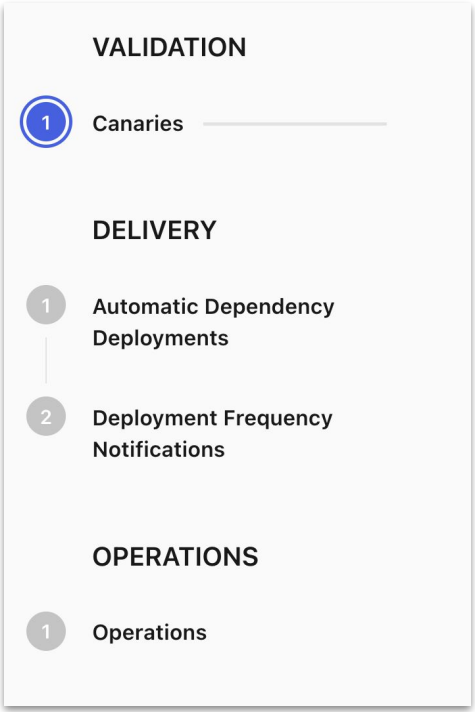
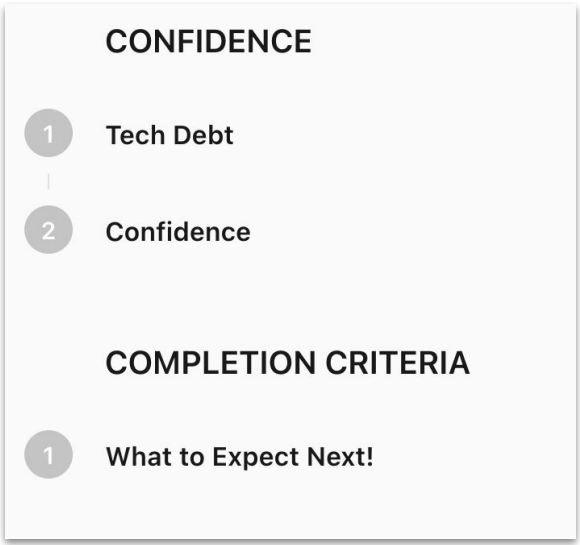


Image from internal tool for campaign lifecycle

The ICFD Campaign



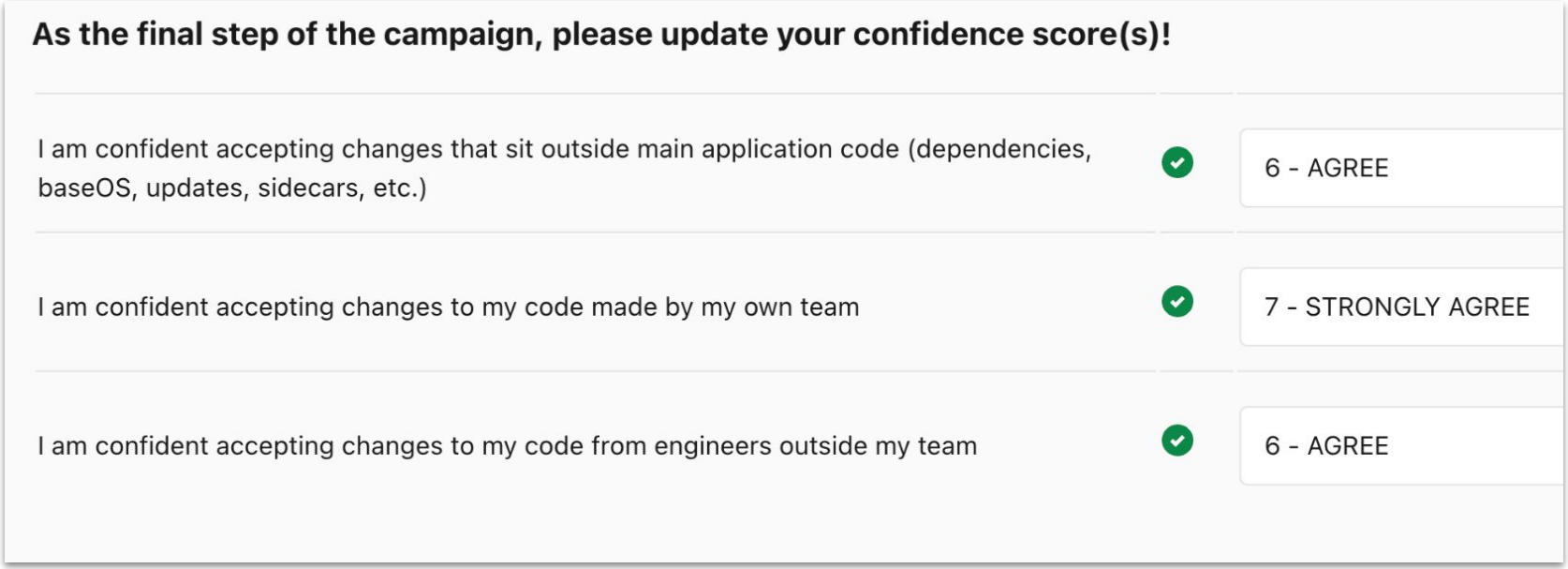




Guiding engineers through potential improvements across a variety of areas

Image from internal tool for campaign lifecycle

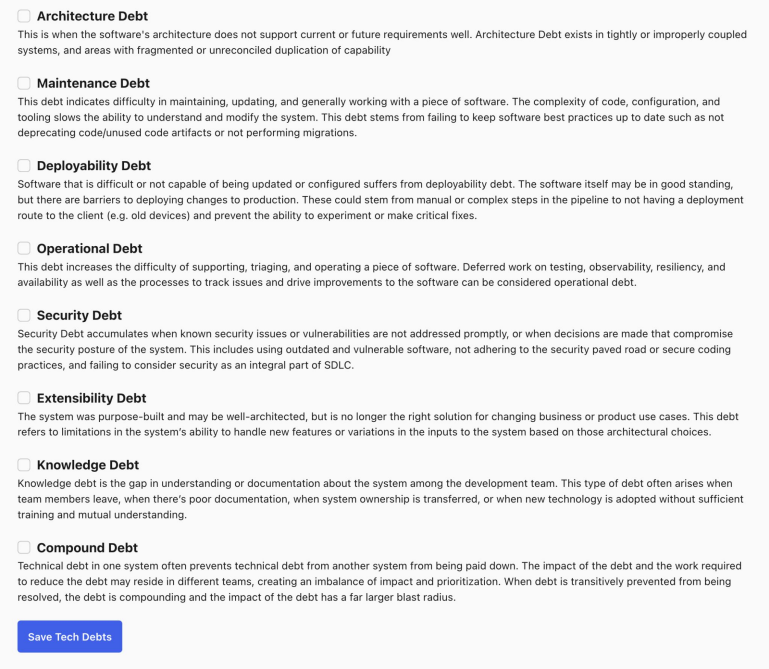
The ICFD Campaign



Asking them to self-score after their work and provide insight into tech debt

Image from internal tool for campaign lifecycle

The ICFD Campaign



Focusing on these Tech Debt types

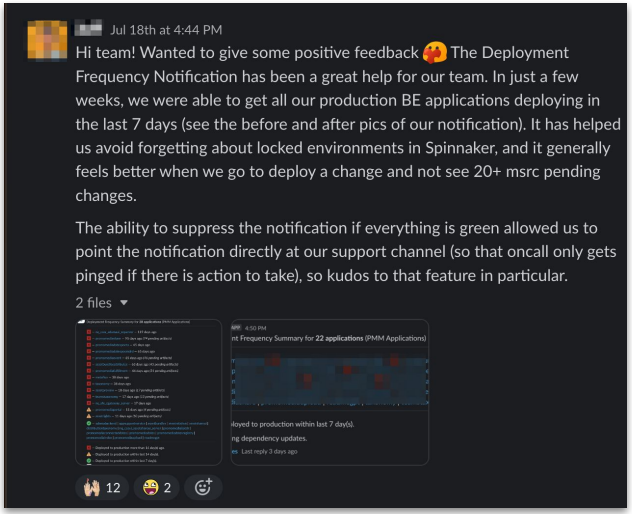
- Architecture
- Maintenance
- Deployability
- Operational
- Security
- Extensibility
- Knowledge
- Compound

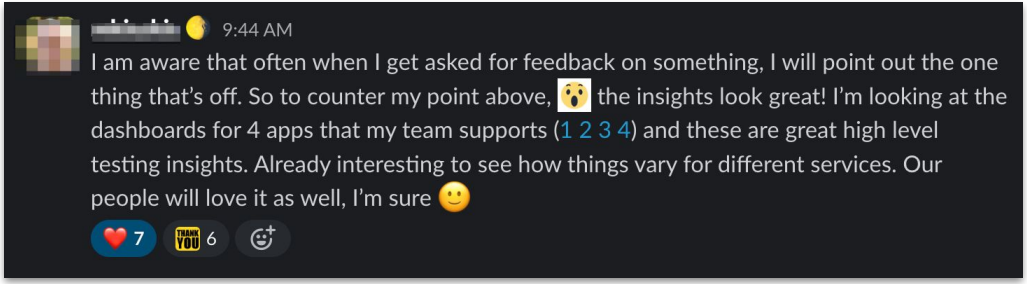
Provide insight into tech debt

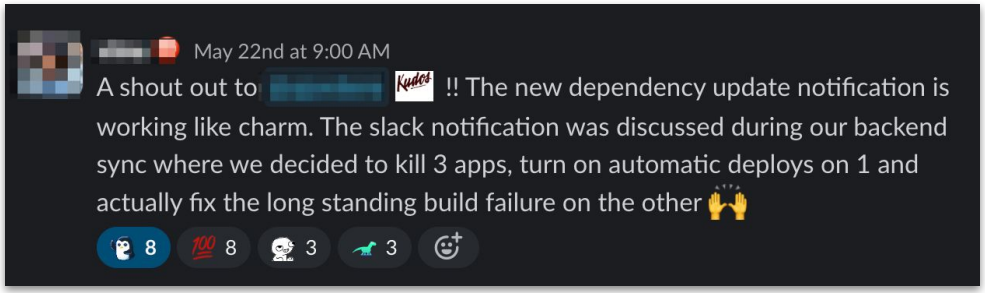
Image from internal tool for campaign lifecycle

Our progress

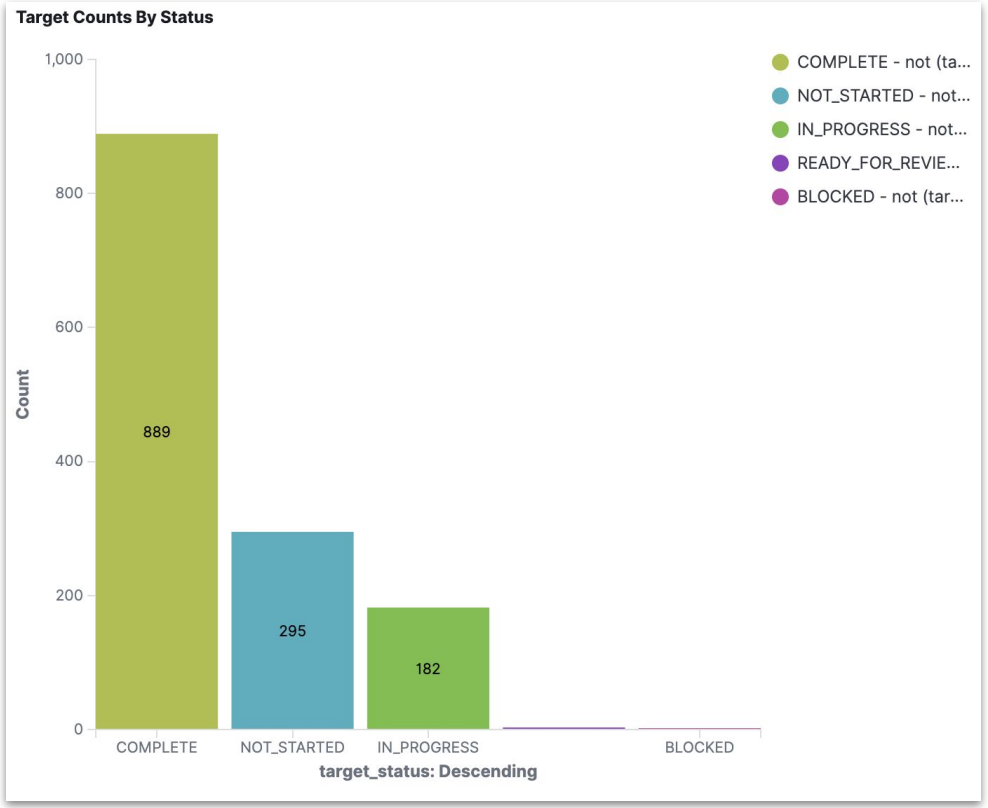
Engineers are excited!





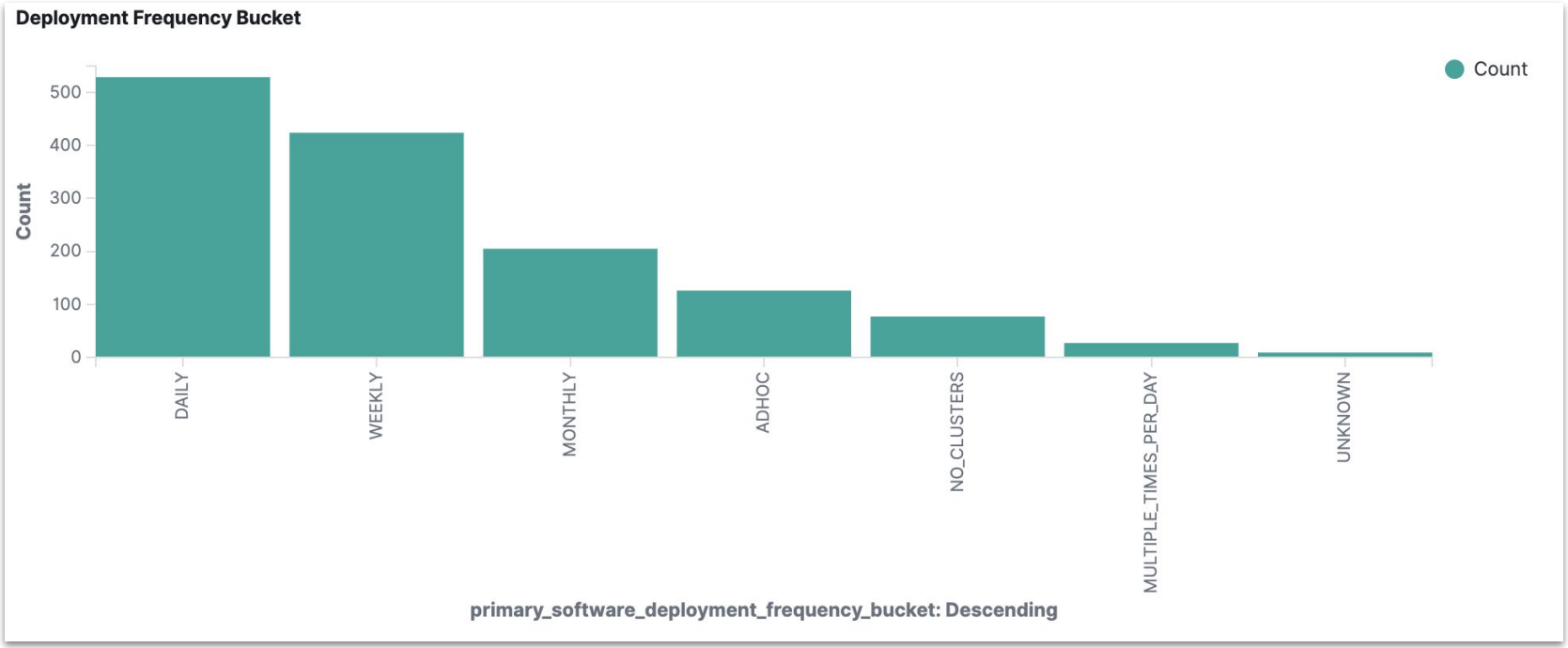


The ICFD Campaign



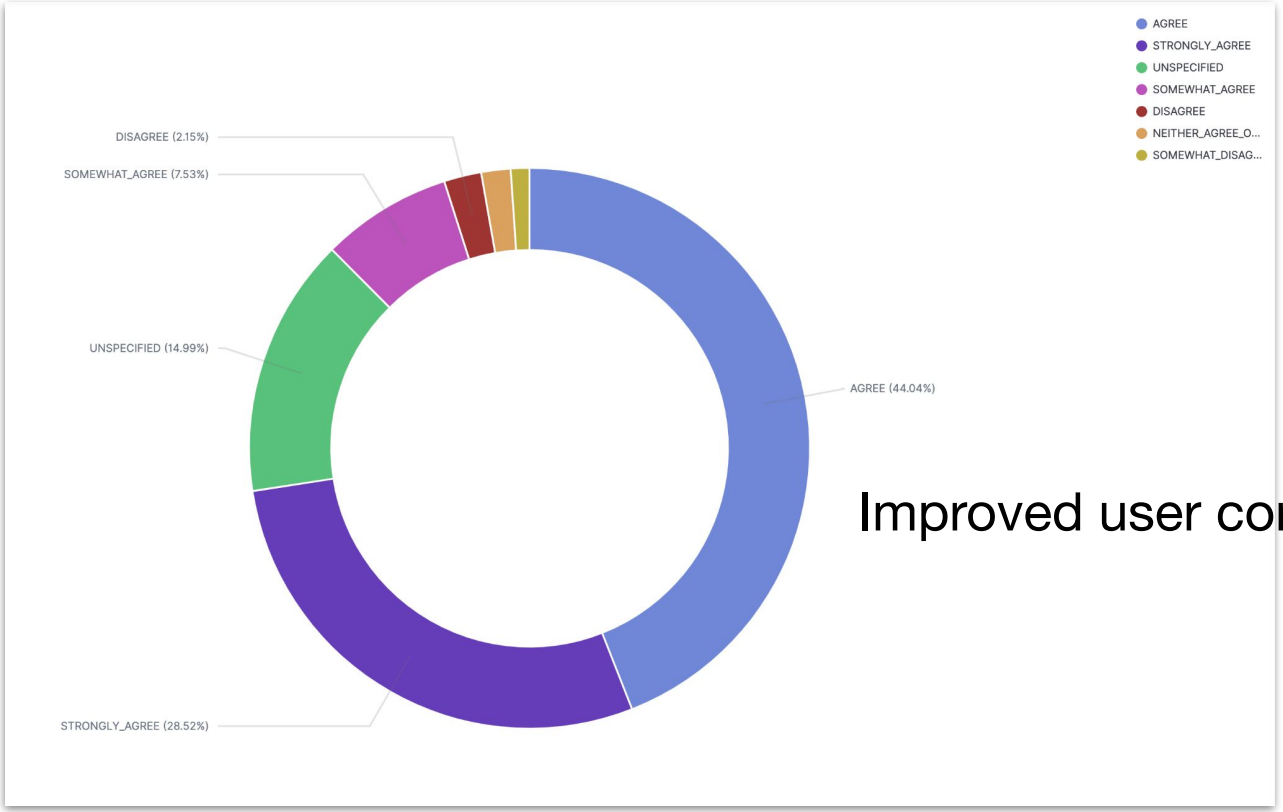
Target applications by status

The ICFD Campaign



Deployment frequency configuration

The ICFD Campaign

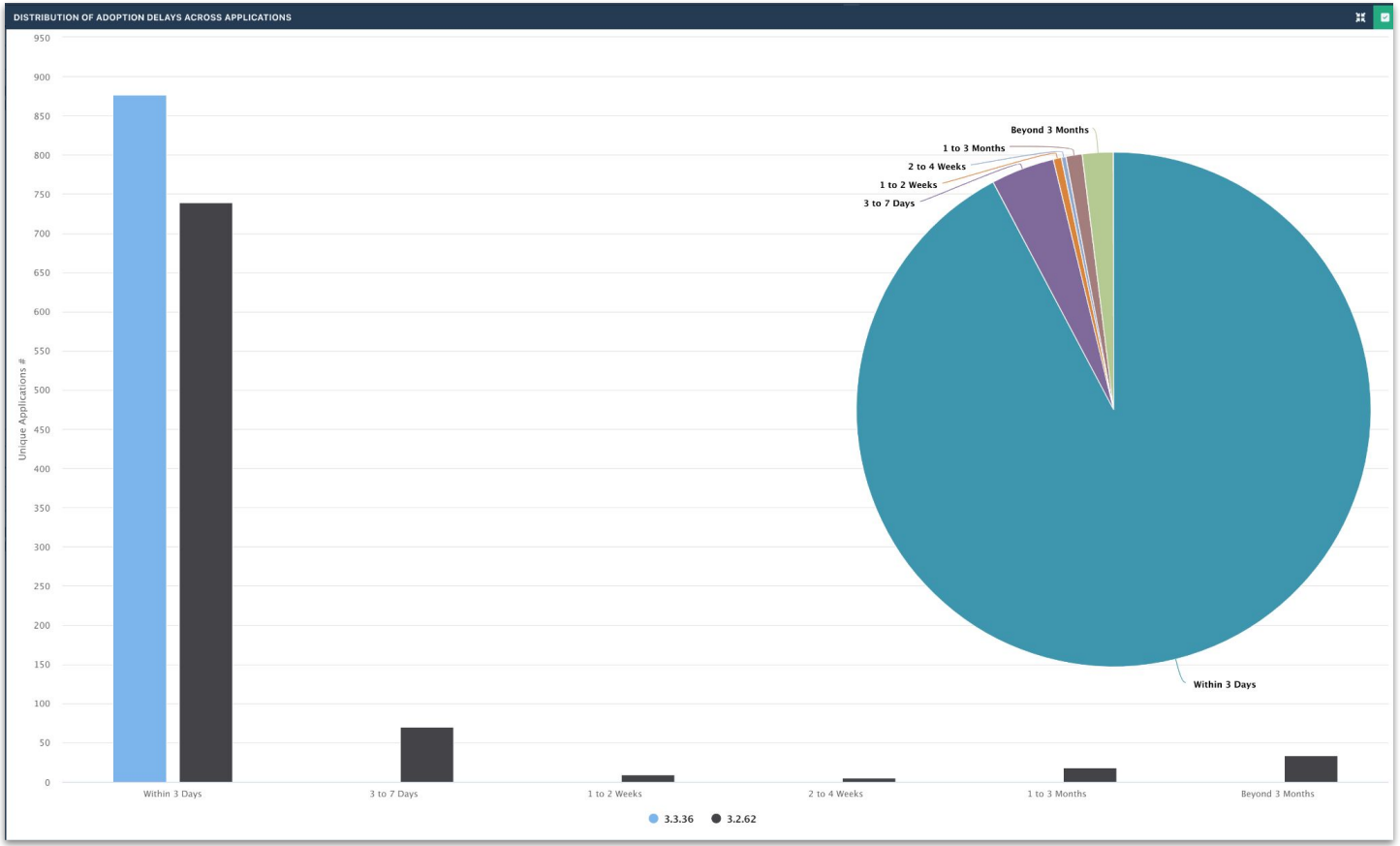


Improved user confidence!

Confidence score distribution

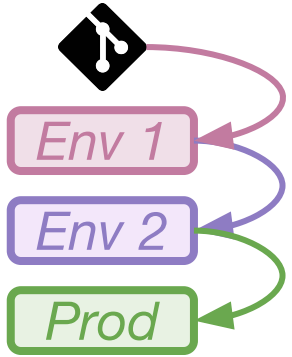
It’s paying off

Adoption of platform-provided libraries has become faster



Confidence chimp campaign

For those who are feeling confident and are fully automated



What is it?

- “Throw a switch” to build and deploy latest commit
- A platform-led exercise that is run periodically to identify gaps and/or friction in Netflix's SDLC workflows

Expected benefits

To our company:

- Better prepare for automatic remediation of incidents and automating code changes for migrations and/or upgrades.

To our engineers:

- Follow-ups will contribute to improvements in CI/CD, which leads to increased reliability and stability for service owners.

Confidence chimp campaign

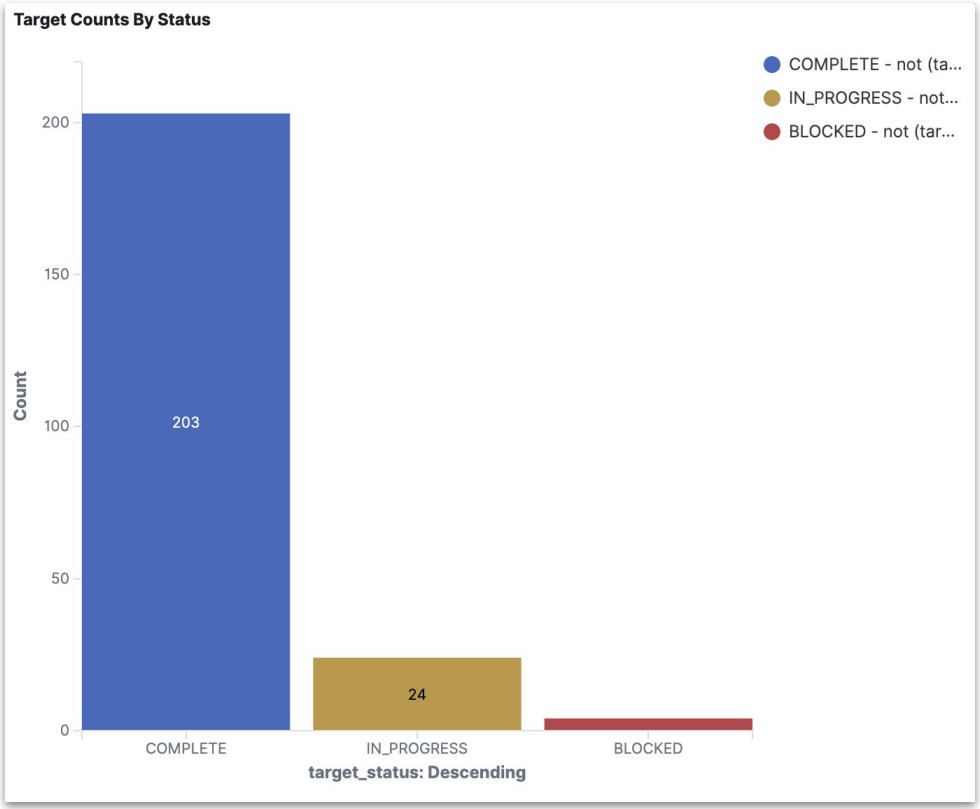
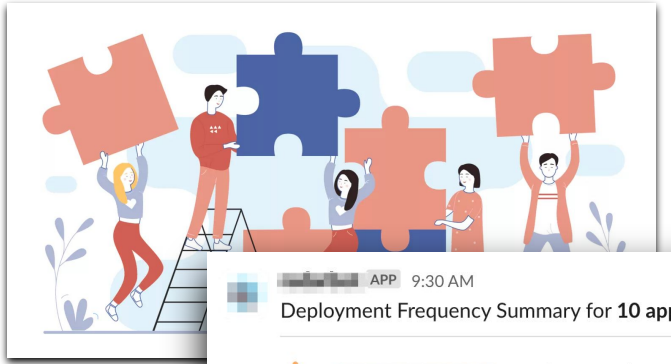
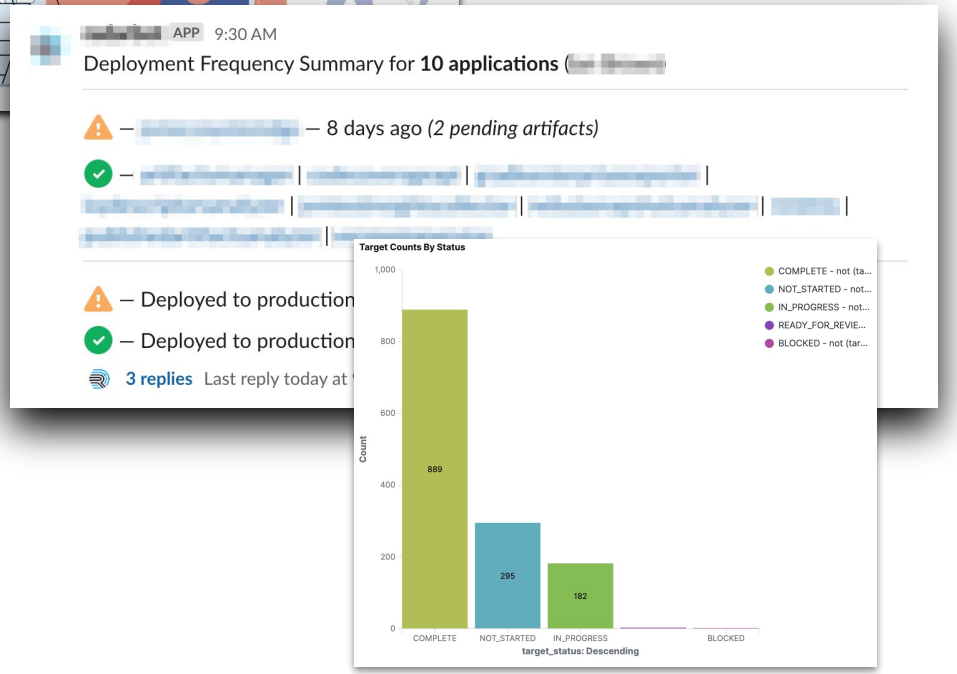


Image from https://netflix.github.io/chaosmonkey/

Target applications by status

Recap





Images from https://lordicon.com/icons/wired/lineal/45-clock-time , https://myva360.com/blog/the-9-types-of-company-culture , https://lordicon.com/icons/wired/outline/967-questionnaire, generated with Chat GPT, and from internal screenshots

Takeaways

1

Code changes are a part of life.

We must be adaptable and flexible to these changes.

And why do we care about doing this well?

Doing this badly costs toil, friction, and money

Image generated with ChatGPT

Takeaways

2

Implementing code changes goes beyond *transformations* and *scheduling.*

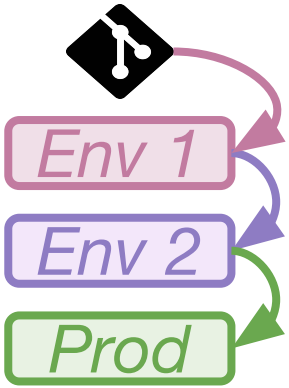
There are great companies solving and tackling this aspect though

Image from https://book.git-scm.com/downloads/logo and https://myva360.com/blog/the-9-types-of-company-culture

Takeaways

3

Making and releasing
fully automated code changes
within the organization
has to be a priority with a
cultural
focus.



Takeaways

4

This takes a highly cross-functional team and effort and a company priority

Image generated with ChatGPT

Questions?

Roberto Perez Alcolea
rperezalcolea@netflix.com

Aubrey Chipman
achipman@netflix.com

Thank you!

Roberto Perez Alcolea
rperezalcolea@netflix.com

Aubrey Chipman
achipman@netflix.com