/thoughtworks

TECHNOLOGY RADAR

An opinionated guide
to technology frontiers

Volume 24

#TWTechRadar
thoughtworks.com/radar

Contributors

The Technology Radar is prepared by the Thoughtworks Technology Advisory Board

The Technology Advisory Board (TAB) is a group of 20 senior technologists at Thoughtworks. The TAB meets twice a year face-to-face and biweekly by phone. Its primary role is to be an advisory group for Thoughtworks CTO, Rebecca Parsons.

The TAB acts as a broad body that can look at topics that affect technology and technologists at Thoughtworks. With the ongoing global pandemic, we once again created this volume of the Technology Radar via a virtual event.

Rebecca Parsons (CTO)

Martin Fowler (Chief Scientist)

Bharani Subramaniam

Birgitta Böckeler

Brandon Byars

Camilla Crispim

Cassie Shum

Erik Dörnenburg

Evan Bottcher

Fausto de la Torre

Hao Xu

Ian Cartwright

James Lewis

Lakshminarasimhan Sudarshan

Mike Mason

Neal Ford

Perla Villarreal

Rachel Laycock

Scott Shaw

Shangqi Liu

Zhamak Dehghani

About the Radar

Thoughtworkers are passionate about technology. We build it, research it, test it, open source it, write about it and constantly aim to improve it — for everyone. Our mission is to champion software excellence and revolutionize IT. We create and share the Thoughtworks Technology Radar in support of that mission. The Thoughtworks Technology Advisory Board, a group of senior technology leaders at Thoughtworks, creates the Radar. They meet regularly to discuss the global technology strategy for Thoughtworks and the technology trends that significantly impact our industry.

The Radar captures the output of the Technology Advisory Board's discussions in a format that provides value to a wide range of stakeholders, from developers to CTOs. The content is intended as a concise summary.

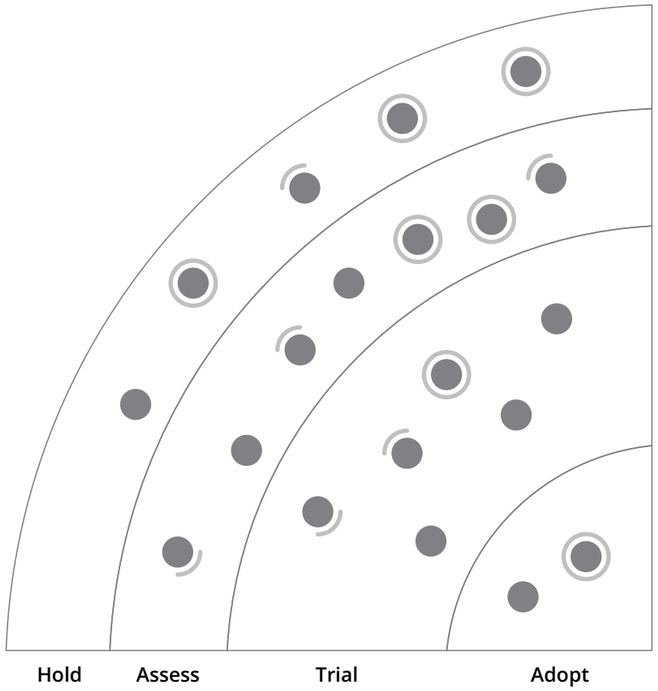
We encourage you to explore these technologies. The Radar is graphical in nature, grouping items into techniques, tools, platforms and languages & frameworks. When Radar items could appear in multiple quadrants, we chose the one that seemed most appropriate. We further group these items in four rings to reflect our current position on them.

For more background on the Radar, see thoughtworks.com/radar/faq.

Radar at a glance

The Radar is all about tracking interesting things, which we refer to as blips. We organize the blips in the Radar using two categorizing elements: quadrants and rings. The quadrants represent different kinds of blips. The rings indicate what stage in an adoption lifecycle we think they should be in.

A blip is a technology or technique that plays a role in software development. Blips are things that are "in motion" — that is we find their position in the Radar is changing — usually indicating that we're finding increasing confidence in them as they move through the rings.



New

Moved in/out

No change

Our Radar is forward looking. To make room for new items, we fade items that haven't moved recently, which isn't a reflection on their value but rather on our limited Radar real estate.

Adopt

We feel strongly that the industry should be adopting these items. We use them when appropriate in our projects.

Trial

Worth pursuing. It's important to understand how to build up this capability. Enterprises can try this technology on a project that can handle the risk.

Assess

Worth exploring with the goal of understanding how it will affect your enterprise.

Hold

Proceed with caution.

Themes for this edition

Platform Teams Drive Speed to Market

Increasingly, organizations are adopting a platform team concept: set up a dedicated group that creates and supports internal platform capabilities — cloud native, continuous delivery, modern observability, AuthZ/N patterns, service mesh, and so on — then use those capabilities to accelerate application development, reduce operational complexity and improve time to market. This growing maturity is welcome and we first featured this technique in the Radar in 2017. But with increasing maturity, we're also discovering antipatterns that organizations should avoid. For example, "one platform to rule them all" may not be optimal, "big platform up front" may take years to deliver value and "build it and they will come" might end up as a wasted effort. Instead, using a product-thinking approach can help you clarify what each of your internal platforms should provide, depending on its customers. Companies that put their platform teams behind a ticketing system like an old-school operations silo find the same disadvantages of misaligned prioritization: slow feedback and response, resource allocation contention and other well-known problems caused by the silo. We've also seen several new tools and integration patterns for teams and technologies emerge, allowing more effective partitioning of both.

Consolidated Convenience over Best in Class

As engineering practices that feature automation, scale and other modern goals become more commonplace with development teams, we see corresponding developer-facing tool integration on many platforms, particularly in the cloud space. For example, artifact repositories, source control, CI/CD pipelines, wikis, and similar tools were usually hand-picked by development teams and stitched together à la carte. Now, delivery platforms such as Azure DevOps and ecosystems such as GitHub have subsumed many of these tool categories. While the level of maturity varies across platform offerings, the appeal of a "one-stop shop" for delivery tooling is undeniable. Overall, it seems that the trade-off lies with having consolidated tool stacks offer greater developer convenience and less churn, but the set of tools rarely represents the best possible.

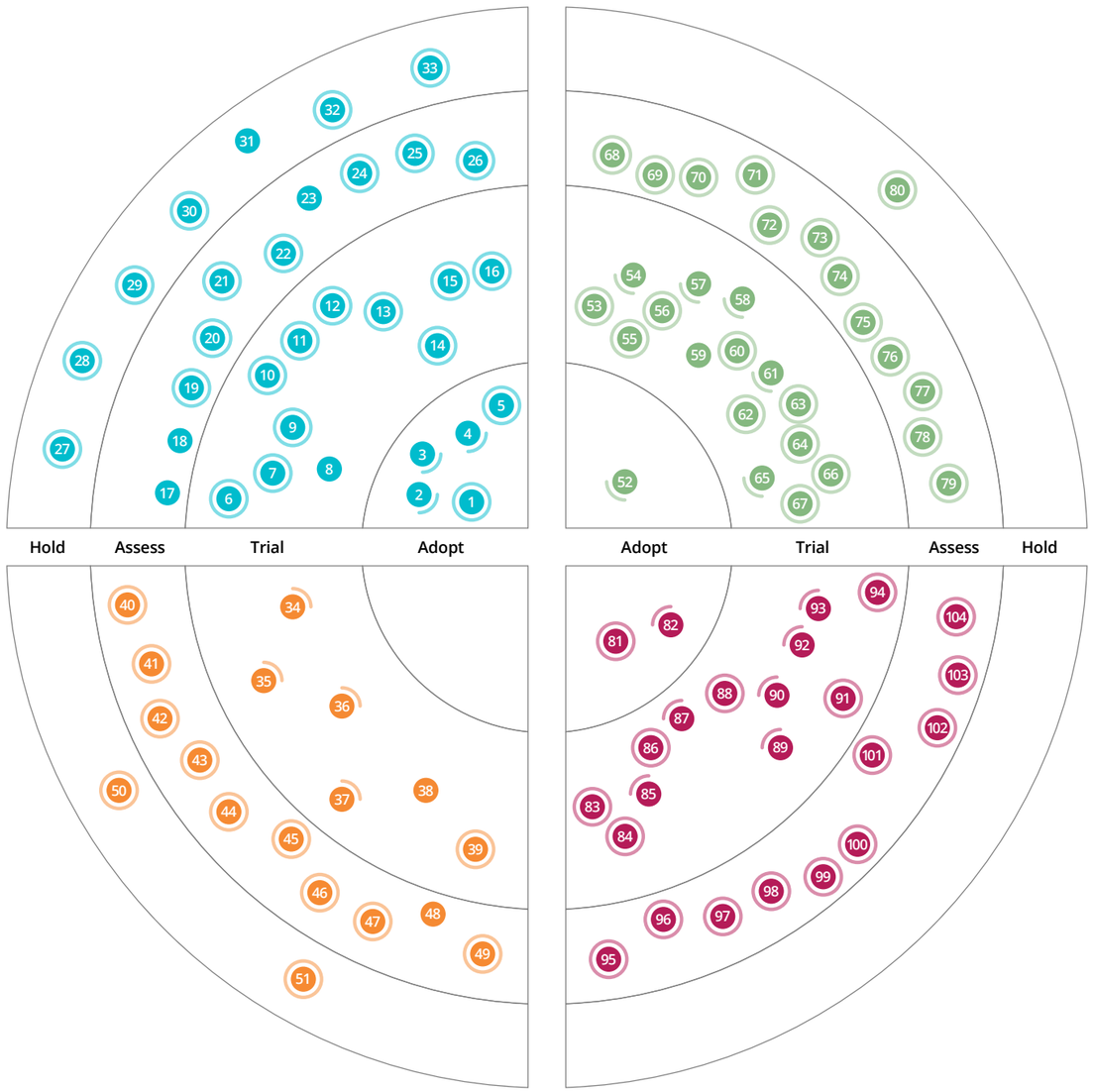
Perennially "Too Complex to Blip"

In Radar nomenclature, the final status after discussion for many complex topics is "TCTB — too complex to blip": items that defy classification because they offer a number of pros and cons, a high amount of nuance as to the applicability of the advice or tool or other reasons that prevent us from summarizing our opinions in a few sentences. Frequently, these topics go on to articles, podcasts, and other non-Radar destinations. Some of our richest conversations center on these topics: they're important but complex, preventing a single succinct point of view. Numerous topics recur meeting after meeting — and, critically, with several of our client engagements — that eventually fall to TCTB, including monorepos, orchestration guidelines for distributed architectures and branching models, among others. For those who wonder why these important topics don't make it into the Radar, it's not for lack of awareness or desire on our part. Like many topics in software development, too many trade-offs exist to allow clear, unambiguous advice. We sometimes do find smaller pieces of the larger topics that we can offer advice on that do make it in the Radar, but the larger topics remain perpetually too nuanced and unsettled for the Radar.

Discerning the Context for Architectural Coupling

A topic that recurs virtually every meeting (see "Perennially 'Too Complex to Blip'") is the appropriate level of coupling in software architecture between microservices, components, API gateways, integration hubs, frontends, and so on... pretty much everywhere two pieces of software might connect, architects and developers struggle finding the correct level of coupling — much common advice encourages extreme decoupling, but that makes building workflows difficult. Coupling in architecture touches on many important considerations: how things are wired, understanding the inherent semantic coupling within each problem domain, how things call one another or how transactionality works (sometimes in combination with other tricky features like scalability). Software can't exist without some level of coupling outside of singular monolithic systems; arriving at the right set of trade-offs to determine the types and levels of coupling becomes a critical skill with modern architectures. We do see specific bad practices such as generating code for client libraries and good practices such as the judicious use of the BFF patterns. However, general advice in this realm is useless and silver bullets don't exist. Invest time and effort in understanding the factors at play when making these decisions on a case-by-case basis rather than seeking a generic but inadequate solution.

The Radar



Techniques

Adopt
1. API expand-contract
2. Continuous delivery for machine learning (CD4ML)
3. Design systems
4. Platform engineering product teams
5. Service account rotation approach

Trial
6. Cloud sandboxes
7. Contextual bandits
8. Distroless Docker images
9. Ethical Explorer
10. Hypothesis-driven legacy renovation
11. Lightweight approach to RFCs
12. Simplest possible ML
13. SPA injection
14. Team cognitive load
15. Tool-managed Xcodeproj
16. UI/BFF shared types

Assess
17. Bounded low-code platforms
18. Decentralized identity
19. Deployment drift radiator
20. Homomorphic encryption
21. Hotwire
22. Import maps for micro frontends
23. Open Application Model (OAM)
24. Privacy-focused web analytics
25. Remote mob programming
26. Secure multiparty computing

Hold
27. GitOps
28. Layered platform teams
29. Naive password complexity requirements
30. Peer review equals pull request
31. SAFe™
32. Separate code and pipeline ownership
33. Ticket-driven platform operating models

Platforms

Adopt

Trial
34. AWS Cloud Development Kit
35. Backstage
36. Delta Lake
37. Materialize
38. Snowflake
39. Variable fonts

Assess
40. Apache Pinot
41. Bit.dev
42. DataHub
43. Feature Store
44. JuiceFS
45. Kafka API without Kafka
46. NATS
47. Opstrace
48. Pulumi
49. Redpanda

Hold
50. Azure Machine Learning
51. Homemade infrastructure-as-code (IaC) products

Tools

Adopt
52. Sentry

Trial
53. axe-core
54. dbt
55. esbuild
56. Flipper
57. Great Expectations
58. k6
59. MLflow
60. OR-Tools
61. Playwright
62. Prowler
63. Pyright
64. Redash
65. Terratest
66. Tuple
67. Why Did You Render

Assess
68. Buildah and Podman
69. GitHub Actions
70. Graal Native Image
71. HashiCorp Boundary
72. imgcook
73. Longhorn
74. Operator Framework
75. Recommender
76. Remote - WSL
77. Spectral
78. Yelp detect-secrets
79. Zally

Hold
80. AWS CodePipeline

Languages & Frameworks

Adopt
81. Combine
82. LeakCanary

Trial
83. Angular Testing Library
84. AWS Data Wrangler
85. Blazor
86. FastAPI
87. io-ts
88. Kotlin Flow
89. LitElement
90. Next.js
91. On-demand modules
92. Streamlit
93. SWR
94. TrustKit

Assess
95. .NET 5
96. bUnit
97. Dagster
98. Flutter for Web
99. Jotai and Zustand
100. Kotlin Multiplatform Mobile
101. LVGL
102. React Hook Form
103. River
104. Webpack 5 Module Federation

Hold

TECHNOLOGY RADAR

Techniques

Techniques

API expand-contract
Adopt

The API expand-contract pattern, sometimes called parallel change, will be familiar to many, especially when used with databases or code; however, we only see low levels of adoption with APIs. Specifically, we're seeing complex versioning schemes and breaking changes used in scenarios where a simple expand and then contract would suffice. For example, first adding to an API while deprecating an existing element, and then only later removing the deprecated elements once consumers are switched to the newer schema. This approach does require some coordination and visibility of the API consumers, perhaps through a technique such as consumer-driven contract testing.

Continuous delivery for machine learning (CD4ML)
Adopt

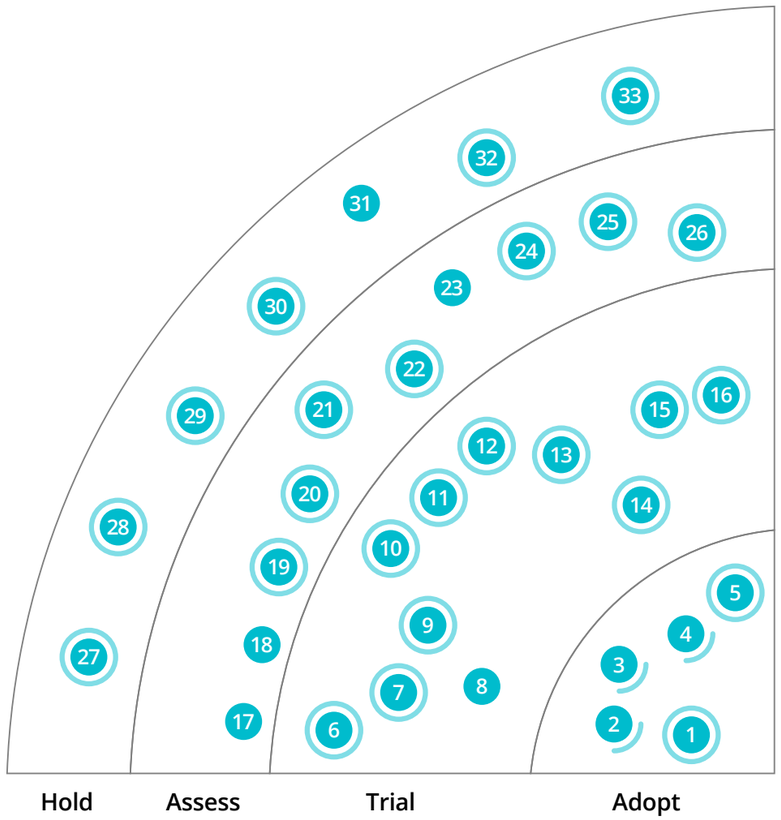
We see continuous delivery for machine learning (CD4ML) as a good default starting point for any ML solution that is being deployed into production. Many organizations are becoming more reliant on ML solutions for both customer offerings and internal operations so it makes sound business sense to apply the lessons and good practice captured by continuous delivery (CD) to ML solutions.

Design systems
Adopt

As application development becomes increasingly dynamic and complex, it's a challenge to deliver accessible and usable products with consistent style. This is particularly true in larger organizations with multiple teams working on different products. Design systems define a collection of design patterns, component libraries and good design and engineering practices that ensure consistent digital products. Built on the corporate style guides of the past, design systems offer shared libraries and documents that are easy to find and use. Generally, guidance is written down as code and kept under version control so that the guide is less ambiguous and easier to maintain than simple documents. Design systems have become a standard approach when working across teams and disciplines in product development because they allow teams to focus. They can address strategic challenges around the product itself without reinventing the wheel every time a new visual component is needed.

Platform engineering product teams
Adopt

As noted in one of the themes for this edition, the industry is increasingly gaining experience with platform engineering product teams that create and support internal platforms. These platforms are used by teams across an organization



Adopt
1. API expand-contract
2. Continuous delivery for machine learning (CD4ML)
3. Design systems
4. Platform engineering product teams
5. Service account rotation approach

Trial
6. Cloud sandboxes
7. Contextual bandits
8. Distroless Docker images
9. Ethical Explorer
10. Hypothesis-driven legacy renovation
11. Lightweight approach to RFCs
12. Simplest possible ML
13. SPA injection
14. Team cognitive load
15. Tool-managed Xcodeproj
16. UI/BFF shared types

Assess
17. Bounded low-code platforms
18. Decentralized identity
19. Deployment drift radiator
20. Homomorphic encryption
21. Hotwire
22. Import maps for micro frontends
23. Open Application Model (OAM)
24. Privacy-focused web analytics
25. Remote mob programming
26. Secure multiparty computing

Hold
27. GitOps
28. Layered platform teams
29. Naive password complexity requirements
30. Peer review equals pull request
31. SAFe™
32. Separate code and pipeline ownership
33. Ticket-driven platform operating models

Techniques

Named after "bandits," or slot machines, in casinos, this algorithm explores different options to learn more about expected outcomes and balances by exploiting the options that perform well.

(Contextual bandits)

and accelerate application development, reduce operational complexity and improve time to market. With increasing adoption we're also clearer on both good and bad patterns for this approach. When creating a platform, it's critical to have clearly defined customers and products that will benefit from it rather than building in a vacuum. We caution against layered platform teams that simply preserve existing technology silos but apply the "platform team" label as well as against ticket-driven platform operating models. We're still big fans of using concepts from Team Topologies as we think about how best to organize platform teams. We consider platform engineering product teams to be a standard approach and a significant enabler for high-performing IT.

Service account rotation approach

Adopt

We strongly advise organizations to make sure, when they really need to use cloud service accounts, that they are rotating the credentials. Rotation is one of the three R's of security. It is far too easy for organizations to forget about these accounts unless an incident occurs. This is leading to accounts with unnecessarily broad permissions remaining in use for long periods alongside a lack of planning for how to replace or rotate them. Regularly applying a cloud service account rotation approach also provides a chance to exercise the principle of least privilege.

Cloud sandboxes

Trial

As the cloud is becoming more and more a commodity and being able to spin up cloud sandboxes is easier and available at scale, our teams prefer cloud-only (as opposed to local) development environments to reduce maintenance complexity. We're seeing that the tooling to do local simulation of cloud-native services limits the confidence in developer build and test cycles; therefore, we're looking to focus on standardizing cloud sandboxes over running cloud-native components on a developer machine. This will drive good infrastructure-as-code practices as a forcing function and good onboarding processes for provisioning sandbox environments for developers. There are risks associated with this transition, as it assumes that developers will have an absolute dependency on cloud environment availability, and it may slow down the developer feedback loop. We strongly recommend you adopt some lean governance practices regarding standardization of these sandbox environments, especially with regard to security, IAM and regional deployments.

Contextual bandits

Trial

Contextual bandits is a type of reinforcement learning that is well suited for problems with exploration/exploitation trade-offs. Named after "bandits," or slot machines, in casinos, the algorithm explores different options to learn more about expected outcomes and balances it by exploiting the options that perform well. We've successfully used this technique in scenarios where we've had little data to train and deploy other machine-learning models. The fact that we can add context to this explore/exploit trade-off makes it suitable for a wide variety of use cases including A/B testing, recommendations and layout optimizations.

Distroless Docker images

Trial

When building Docker images for our applications, we're often concerned with two things: the security and the size of the image. Traditionally, we've used container security scanning tools to detect and patch common vulnerabilities and exposures and small distributions such as Alpine Linux to address the image size and distribution performance. But with rising security threats, eliminating all possible attack vectors is more important than ever. That's why distroless Docker images are becoming the default choice for deployment containers. Distroless Docker images reduce the footprint and dependencies by doing away with a full operating system distribution. This technique reduces security scan noise and the application attack surface. Moreover, fewer vulnerabilities need to be patched and as a bonus, these smaller images are more efficient. Google has published a set of distroless container images for different languages. You can create distroless application images using the Google build tool Bazel or simply use multistage Dockerfiles. Note that distroless containers by default don't have a shell for debugging. However, you can easily find debug versions of distroless containers online, including a BusyBox shell. Distroless Docker images is a technique pioneered by Google and, in our experience, is still largely confined to Google-generated images. We would be more comfortable if there were more than one provider to choose from. Also, use caution when applying Trivy or similar vulnerability scanners since distroless containers are only supported in more recent versions.

Ethical Explorer

Trial

The group behind Ethical OS — the Omidyar Network, a self-described social change venture created by eBay founder Pierre Omidyar — has released a new iteration called Ethical Explorer. The new Ethical Explorer pack draws

on lessons learned from using Ethical OS and adds further questions for product teams to consider. The kit, which can be downloaded for free and folded into cards to trigger discussion, has open-ended question prompts for several technical "risk zones," including surveillance ("can someone use our product or service to track or identify other users?"), disinformation, exclusion, algorithmic bias, addiction, data control, bad actors and outsized power. The included field guide has activities and workshops, ideas for starting conversations and tips for gaining organizational buy-in. While we've a long way to go as an industry to better represent the ethical externalities of our digital society, we've had some productive conversations using Ethical Explorer, and we're encouraged by the broadening awareness of the importance of product decisions in addressing societal issues.

Hypothesis-driven legacy renovation

Trial

We're often asked to refresh, update or remediate legacy systems that we didn't originally build. Sometimes, technical issues need our attention such as improving performance or reliability. One common approach to address these issues is to create "technical stories" using the same format as a user story but with a technical outcome rather than a business one. But these technical tasks are often difficult to estimate, take longer than anticipated or don't end up having the desired outcome. An alternative, more successful method is to apply hypothesis-driven legacy renovation. Rather than working toward a standard backlog, the team takes ownership of a measurable technical outcome and collectively establishes a set of hypotheses about the problem. They then conduct iterative, time-boxed experiments to verify or disprove each hypothesis in order of priority. The resulting workflow is optimized for reducing uncertainty rather than following a plan toward a predictable outcome.

Lightweight approach to RFCs

Trial

As organizations drive toward evolutionary architecture, it's important to capture decisions around design, architecture, techniques and teams' ways of workings. The process of collecting and aggregating feedback that will lead to these decisions begin with Request for Comments (RfCs). RfCs are a technique for collecting context, design and architectural ideas and collaborating with teams to ultimately come to decisions along with their context and consequences. We recommend that organizations take a lightweight approach to RFCs by using a simple standardized template across many teams as well as version control to capture RfCs.

It's important to capture these in an audit of these decisions to benefit future team members and to capture the technical and business evolution of an organization. Mature organizations have used RfCs in autonomous teams to drive better communication and collaboration especially in cross-team relevant decisions.

Simplest possible ML

Trial

All major cloud providers offer a dazzling array of machine-learning (ML) solutions. These powerful tools can provide a lot of value, but come at a cost. There is the pure run cost for these services charged by the cloud provider. In addition, there is a kind of operations tax. These complex tools need to be understood and operated, and with each new tool added to the architecture this tax burden increases. In our experience, teams often choose complex tools because they underestimate the power of simpler tools such as linear regression. Many ML problems don't require a GPU or neural networks. For that reason we advocate for the simplest possible ML, using simple tools and models and a few hundred lines of Python on the compute platform you have at hand. Only reach for the complex tools when you can demonstrate the need for them.

SPA injection

Trial

The strangler fig pattern is often the default strategy for legacy modernization, where the new code wraps around the old and slowly absorbs the ability to handle all the needed functionality. That sort of "outside-in" approach works well for a number of legacy systems, but now that we've had enough experience with single-page applications (SPA) for them to become legacy systems themselves, we're seeing the opposite "inside-out" approach used to replace them. Instead of wrapping the legacy system, we instead embed the beginning of the new SPA into the HTML document containing the old one and let it slowly expand in functionality. The SPA frameworks don't even need to be the same as long as users can tolerate the performance hit of the increased page size (e.g., embedding a new React app inside an old AngularJS one). SPA injection allows you to iteratively remove the old SPA until the new one completely takes over. Whereas a strangler fig can be viewed as a type of parasite that uses the host tree's stable external surface to support itself until it takes root and the host itself dies, this approach is more like injecting an outside agent into the host, relying on functionality of the original SPA until it can completely take over.

Techniques

When it comes to legacy modernization in single-page apps (SPAs), instead of wrapping the legacy system we instead embed the beginning of the new SPA into the HTML document containing the old one and let it slowly expand in functionality.

(SPA Injection)

Techniques

Team interaction is one of the variables that impacts speed and the ease with which teams deliver value to their customers. The Team Topologies author developed an assessment for measuring these interactions which we call team cognitive load.

(Team cognitive load)

Team cognitive load
Trial

A system's architecture mimics organizational structure and its communication. It's not big news that we should be intentional about how teams interact — see, for instance, the Inverse Conway Maneuver. Team interaction is one of the variables for how fast and how easily teams can deliver value to their customers. We were happy to find a way to measure these interactions; we used the Team Topologies author's assessment which gives you an understanding of how easy or difficult the teams find it to build, test and maintain their services. By measuring team cognitive load, we could better advise our clients on how to change their teams' structure and evolve their interactions.

Tool-managed Xcodeproj
Trial

Many of our developers coding iOS in Xcode often get headaches because the Xcodeproj file changes with every project change. The Xcodeproj file format is not human-readable, hence trying to handle merge conflicts is quite complicated and can lead to productivity loss and risk of messing up the entire project — if anything goes wrong with the file, Xcode won't work properly and developers will very likely be blocked. Instead of trying to merge and fix the file manually or version it, we recommend you use a tool-managed Xcodeproj approach: Define your Xcode project configuration in YAML (XcodeGen, Struct), Ruby (Xcake) or Swift (Tuist). These tools generate the Xcodeproj file based on a configuration file and the project structure. As a result, merge conflicts in the Xcodeproj file will be a thing of the past, and when they do happen in the configuration file, they're much easier to handle.

UI/BFF shared types
Trial

With TypeScript becoming a common language for front-end development and Node.js becoming the preferred BFF technology, we're seeing increasing use of UI/BFF shared types. In this technique, a single set of type definitions is used to define both the data objects returned by front-end queries and the data served to satisfy those queries by the back-end server. Ordinarily, we would be cautious about this practice because of the unnecessarily tight coupling it creates across process boundaries. However, many teams are finding that the benefits of this approach outweigh any risks of tight coupling. Since the BFF pattern works best when the same team owns both the UI code and the BFF, often storing both components in the same repository, the UI/BFF pair can be viewed as a single cohesive system. When the BFF offers strongly typed queries, the results can be tailored to the specific needs of the frontend rather than reusing a single, general-purpose entity that must serve the needs of many consumers and contain more fields than actually required. This reduces the risk of accidentally exposing data that the user shouldn't see, prevents incorrect interpretation of the returned data object and makes the query more expressive. This practice is particularly useful when implemented with io-ts to enforce the run-time type safety.

Bounded low-code platforms
Assess

One of the most nuanced decisions facing companies at the moment is the adoption of low-code or no-code platforms, that is, platforms that solve very specific problems in very limited domains. Many vendors are pushing aggressively into this space. The problems we see with these platforms typically relate to an inability to apply good engineering practices such as versioning. Testing too is typically really hard. However, we noticed some interesting new entrants to the market — including Amazon Honeycode, which makes it easy to create simple task or event management apps, and Parabola for IFTTT-like cloud workflows — which is why we're once again including. bounded low-code platforms in this volume. Nevertheless, we remain deeply skeptical about their wider applicability since these tools, like Japanese Knotweed, have a knack of escaping their bounds and tangling everything together. That's why we still strongly advise caution in their adoption.

Decentralized identity
Assess

In 2016, Christopher Allen, a key contributor to SSL/TLS, inspired us with an introduction of 10 principles underpinning a new form of digital identity and a path to get there, the path to self-sovereign identity. Self-sovereign identity, also known as decentralized identity, is a "lifetime portable identity for any person, organization, or thing that does not depend on any centralized authority and can never be taken away," according to the Trust over IP standard. Adopting and implementing decentralized identity is gaining momentum and becoming attainable. We see its adoption in privacy-respecting customer health applications, government healthcare infrastructure and corporate legal identity. If you want to rapidly get started with decentralized identity, you can assess Sovrin Network, Hyperledger Aries and Indy OSS, as well as decentralized identifiers and verifiable credentials standards. We're watching this space closely as we help our clients with their strategic positioning in the new era of digital trust.

Deployment drift radiator
Assess

A deployment drift radiator makes version drift visible for deployed software across multiple environments. Organizations using automated deployments may require manual approvals for environments that get closer to production, meaning the code in these environments might well be lagging several versions behind current development. This technique makes this lag visible via a simple dashboard showing how far behind each deployed component is for each environment. This helps to highlight the opportunity cost of completed software not yet in production while drawing attention to related risks such as security fixes not yet deployed.

Homomorphic encryption
Assess

Fully homomorphic encryption (HE) refers to a class of encryption methods that allow computations (such as search and arithmetic) to be performed directly on encrypted data. The result of such a computation remains in encrypted form, which at a later point can be decrypted and revealed. Although the HE problem was first proposed in 1978, a solution wasn't constructed until 2009. With advances in computing power and the availability of easy-to-use open-source libraries — including SEAL, Lattigo, HElib and partially homomorphic encryption in Python — HE is becoming feasible in real-world applications. The motivating scenarios include privacy-preserving use cases, where computation can be outsourced to an untrusted party, for example, running computation on encrypted data in the cloud, or enabling a third party to aggregate homomorphically encrypted intermediate federated machine learning results. Moreover, most HE schemes are considered to be secure against quantum computers, and efforts are underway to standardize HE. Despite its current limitations, namely performance and feasibility of the types of computations, HE is worth your attention.

Hotwire
Assess

Hotwire (HTML over the wire) is a technique to build web applications. Pages are constructed out of components, but unlike modern SPAs the HTML for the components is generated on the server side and then sent "over the wire" to the browser. The application has only a small amount of JavaScript code in the browser to stitch the HTML fragments together. Our teams, and doubtlessly others too, experimented with this technique after asynchronous web requests gained cross-browser support around 2005, but for various reasons it never gained much traction.

Today, Hotwire uses modern web browser and HTTP capabilities to achieve the speed, responsiveness and dynamic nature of single-page apps (SPAs). It embraces simpler web application design by localizing the logic to the server and keeping the client-side code simple. The team at Basecamp has released a few Hotwire frameworks that power their own application, including Turbo and Stimulus. Turbo includes a set of techniques and frameworks to speed up the application responsiveness by preventing whole page reloading, page preview from cache and decomposing the page into fragments with progressive enhancements on request. Stimulus is designed to enhance static HTML in the browser by connecting JavaScript objects to the page elements on the HTML.

Import maps for micro frontends
Assess

When composing an application out of several micro frontends, some part of the system needs to decide which micro frontends to load and where to load them from. So far, we've either built custom solutions or relied on a broader framework like single-spa. Now there is a new standard, import maps, that helps in both cases. Our first experiences show that using import maps for micro frontends allows for a neat separation of concerns. The JavaScript code states what to import and a small script tag in the initial HTML response specifies where to load the frontends from. That HTML is obviously generated on the server side, which makes it possible to use some dynamic configuration during its rendering. In many ways this technique reminds us of linker/loader paths for dynamic Unix libraries. At the moment import maps are only supported by Chrome, but with the SystemJS polyfill they're ready for wider use.

Open Application Model (OAM)
Assess

The Open Application Model (OAM) is an attempt to bring some standardization to the space of shaping infrastructure platforms as products. Using the abstractions of components, application configurations, scopes and traits, developers can describe their applications in a platform-agnostic way, while platform implementers define their platform in terms of workload, trait and scope. Since we last talked about the OAM, we've followed one of its first implementations with interest, KubeVela. KubeVela is close

Techniques

Organizations using automated deployments sometimes require manual approvals for environments that get closer to production, leading the code in these environments to lag behind current development. A deployment drift radiator makes this lag visible via a simple dashboard.

(Deployment drift radiator)

Techniques

Secure multiparty computing solves the problem of collaborative computing that protects privacy between parties that do not trust each other without involving a third party.

(Secure multiparty computing)

to release 1.0, and we're curious to see if implementations like this can substantiate the promise of the OAM idea.

Privacy-focused web analytics
Assess

Privacy-focused web analytics is a technique for gathering web analytics without compromising end user privacy by keeping the end users truly anonymous. One surprising consequence of General Data Protection Regulation (GDPR) compliance is the decision taken by many organizations to degrade the user experience with complex cookie consent processes, especially when the user doesn't immediately consent to the "all the cookies" default settings. Privacy-focused web analytics has the dual benefit of both observing the spirit and letter of GDPR while also avoiding the need to introduce intrusive cookie consent forms. One implementation of this approach is Plausible.

Remote mob programming
Assess

Mob programming is one of those techniques that our teams have found to be easier when done remotely. Remote mob programming is allowing teams to quickly "mob" around an issue or piece of code without the physical constraints of only being able to fit so many people around a pairing station. Teams can quickly collaborate on an issue or piece of code without having to connect to a big display, book a physical meeting room or find a whiteboard.

Secure multiparty computing
Assess

Secure multiparty computing (MPC) solves the problem of collaborative computing that protects privacy between parties that do not trust each other. It's aim is to safely calculate an agreed-upon problem without a trusted third party, while each participant is required to partake in the calculation result and can't be obtained by other entities. A simple illustration for MPC is the millionaires' problem: two millionaires want to understand who is the richest, but neither want to share their actual net worth with each other nor trust a third party. The implementation approaches of MPC vary; scenarios may include secret sharing, oblivious transfer, garbled circuits or homomorphic encryption. Some commercial MPC solutions that have recently appeared (e.g., Antchain Morse) claim to help solve the problems of secret sharing and secure machine learning in scenarios such as multiparty joint credit investigation and medical records data exchange. Although these platforms are attractive from a marketing perspective, we've yet to see whether they're really useful.

GitOps
Hold

We suggest approaching GitOps with a degree of care, especially with regard to branching strategies. GitOps can be seen as a way of implementing infrastructure as code that involves continuously synchronizing and applying infrastructure code from Git into various environments. When used with a "branch per environment" infrastructure, changes are promoted from one environment to the next by merging code. While treating code as the single source of truth is clearly a sound approach, we're seeing branch per environment lead to environmental drift and eventually environment-specific configs as code merges become problematic or even stop entirely. This is very similar to what we've seen in the past with long-lived branches with GitFlow.

Layered platform teams
Hold

The explosion of interest around software platforms has created a lot of value for organizations, but the path to building a platform-based delivery model is fraught with potential dead ends. It's common in the excitement of new paradigms to see a resurgence of older techniques rebranded with the new vernacular, making it easy to lose sight of the reasons we moved past those techniques in the first place. For an example of this rebranding, see our blip on traditional ESBs make a comeback as API gateways in the previous Radar. Another example we're seeing is rehashing the approach of dividing teams by technology layer but calling them platforms. In the context of building an application, it used to be common to have a front-end team separate from the business logic team separate from the data team, and we see analogs to that model when organizations segregate platform capabilities among teams dedicated to a business or data layer. Thanks to Conway's Law, we know that organizing platform capability teams around business capabilities is a more effective model, giving the team end-to-end ownership of the capability, including data ownership. This helps to avoid the dependency management headaches of layered platform teams, with the front-end team waiting on the business logic team waiting on the data team to get anything done.

Naive password complexity requirements
Hold

Password policies are a standard default for many organizations today. However, we're still seeing organizations requiring passwords to include a variety of symbols, numbers, uppercase and lowercase letters as well as inclusion of special characters.

These are naive password complexity requirements that lead to a false sense of security as users will opt for more insecure passwords because the alternative is difficult to remember and type. According to NIST recommendations, the primary factor in password strength is password length, and therefore users should choose long passphrases with a maximum requirement of 64 characters (including spaces). These passphrases are more secure and memorable.

Peer review equals pull request

Hold

Some organizations seem to think peer review equals pull request; they've taken the view that the only way to achieve a peer review of code is via a pull request. We've seen this approach create significant team bottlenecks as well as significantly degrade the quality of feedback as overloaded reviewers begin to simply reject requests. Although the argument could be made that this is one way to demonstrate code review "regulatory compliance" one of our clients was told this was invalid since there was no evidence the code was actually read by anyone prior to acceptance. Pull requests are only one way to manage the code review workflow; we urge people to consider other approaches, especially where there is a need to coach and pass on feedback carefully.

SAFe™

Hold

Our positioning regarding "being agile before doing agile" and our opinions around this topic shouldn't come as a surprise; but since SAFe™ (Scaled Agile Framework®), per Gartner's May 2019 report, is the most considered and most used enterprise agile framework, and since we're seeing more and more enterprises going through organizational changes, we thought it was time to raise awareness on this topic again. We've come across organizations struggling with SAFe's over-standardized, phase-gated processes. Those processes create friction in the organizational structure and its operating model. It can also promote silos in the organization, preventing platforms from becoming real business capabilities enablers. The top-down control generates waste in the value stream and discourages engineering talent creativity, while limiting autonomy and experimentation in the teams. Rather than measuring effort and focusing on standardized ceremonies, we recommend a leaner, value-driven approach and governance to help eliminate organizational friction such as EDGE, as well as a team cognitive load assessment to identify types of teams and determine how they should better interact with each other.

Scaled Agile Framework® and SAFe™ are trademarks of Scaled Agile, Inc.

Separate code and pipeline ownership

Hold

Ideally, but especially when teams are practicing DevOps, the deployment pipeline and the code being deployed should be owned by the same team. Unfortunately, we still see organizations where there is separate code and pipeline ownership, with the deployment pipeline configuration owned by the infrastructure team; this results in delays to changes, barriers to improvements and a lack of development team ownership and involvement in deployments. One cause of this can clearly be the separate team, another can be the desire to retain "gatekeeper" processes and roles. Although there can be legitimate reasons for using this approach (e.g., regulatory control), in general we find it painful and unhelpful.

Ticket-driven platform operating models

Hold

One of the ultimate goals of a platform should be to reduce ticket-based processes to an absolute minimum, as they create queues in the value stream. Sadly, we still see organizations not pushing forcefully enough toward this important goal, resulting in a ticket-driven platform operating model. This is particularly frustrating when ticket-based processes are put in front of platforms that are built on top of the self-service and API-driven features of public cloud vendors. It's hard and not necessary to achieve self-service with very few tickets right from the start, but it needs to be the destination.

Over-reliance on bureaucracy and lack of trust are among the causes of this reluctance to move away from ticket-based processes. Baking more automated checks and alerts into your platform is one way to help cut the cord from approval processes with tickets. For example, provide teams with visibility into their run costs and put in automated guardrails to avoid accidental explosion of costs. Implement security policy as code and use configuration scanners or analyzers like Recommender to help teams do the right thing.

Techniques

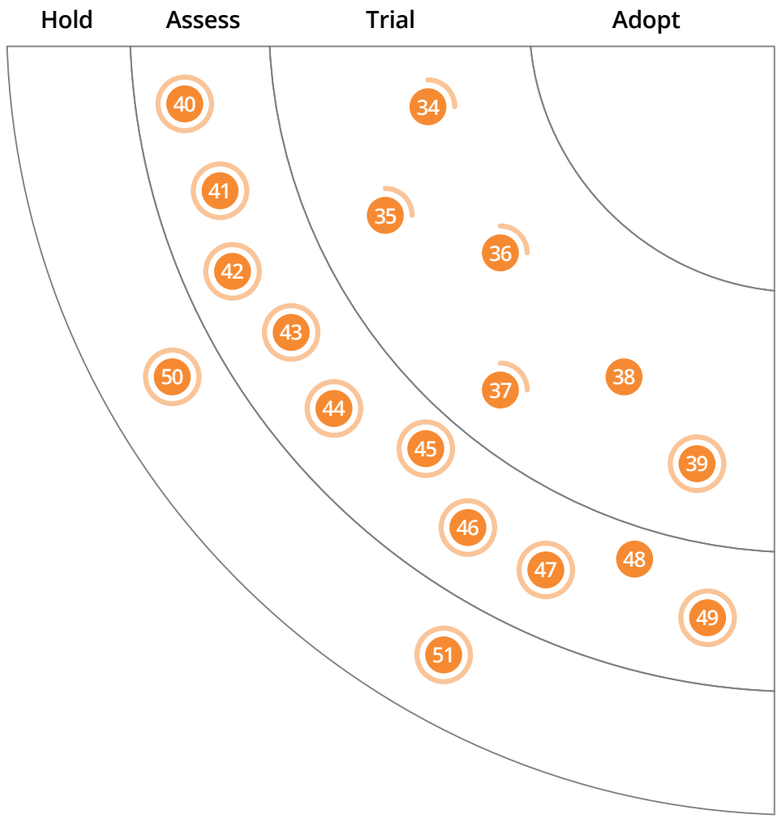
Some organizations seem to think peer review equals pull request. We've seen this approach create significant team bottlenecks as well as significantly degrade the quality of feedback.

(Peer review equals pull request)

TECHNOLOGY RADAR

Platforms

Platforms



AWS Cloud Development Kit
Trial

Many of our teams who are already on AWS have found AWS Cloud Development Kit (AWS CDK) to be a sensible AWS default for enabling infrastructure provisioning. In particular, they like the use of first-class programming languages instead of configuration files which allows them to use existing tools, test approaches and skills. Like similar tools, care is still needed to ensure deployments remain easy to understand and maintain. The development kit currently supports TypeScript, JavaScript, Python, Java, C# and .NET. New providers are being added to the CDK core. We've also used both AWS Cloud Development Kit and HashiCorp's Cloud Development Kit for Terraform to generate Terraform configurations and enable provisioning with the Terraform platform with success.

Backstage
Trial

We continue to see interest in and use of Backstage grow, alongside the adoption of developer portals, as organizations look to support and streamline their development environments. As the number of tools and technologies increases, some form of standardization is becoming increasingly important for consistency so that developers are able to focus on innovation and product development instead of getting bogged down with reinventing the wheel. Backstage is an open-source developer portal platform created by Spotify, it's based upon software templates, unifying infrastructure tooling and consistent and centralized technical documentation. The plugin architecture allows for extensibility and adaptability into an organization's infrastructure ecosystem.

Delta Lake
Trial

Delta Lake is an open-source storage layer, implemented by Databricks, that attempts to bring ACID transactions to big data processing. In our Databricks-enabled data lake or data mesh projects, our teams continue to prefer using Delta Lake storage over the direct use of file storage types such S3 or ADLS. Of course this is limited to projects that use storage platforms that support Delta Lake when using Parquet file formats. Delta Lake facilitates concurrent data read/write use cases where file-level transactionality is required. We find Delta Lake's seamless integration with Apache Spark batch and micro-batch APIs greatly helpful, particularly features such as time travel — accessing data at a particular point in time or commit reversion — as well as schema evolution support on write; though there are some limitations on these features.

Adopt

Trial
34. AWS Cloud Development Kit
35. Backstage
36. Delta Lake
37. Materialize
38. Snowflake
39. Variable fonts

Assess
40. Apache Pinot
41. Bit.dev
42. DataHub
43. Feature Store
44. JuiceFS
45. Kafka API without Kafka
46. NATS
47. Opstrace
48. Pulumi
49. Redpanda

Hold
50. Azure Machine Learning
51. Homemade infrastructure-as-code (IaC) products

Platforms

LinkedIn has evolved WhereHows to DataHub, the next-generation platform that addresses data discoverability via an extensible metadata system.

(DataHub)

Materialize
Trial

Materialize is a streaming database that enables you to do incremental computation without complicated data pipelines. Just describe your computations via standard SQL views and connect Materialize to the data stream. The underlying differential data flow engine performs incremental computation to provide consistent and correct output with minimal latency. Unlike traditional databases, there are no restrictions in defining these materialized views, and the computations are executed in real time. We've used Materialize, together with Spring Cloud Stream and Kafka, to query over streams of events for insights in a distributed event-driven system, and we quite like the setup.

Snowflake
Trial

Since we last mentioned Snowflake in the Radar, we've gained more experience with it as well as with data mesh as an alternative to data warehouses and lakes. Snowflake continues to impress with features like time travel, zero-copy cloning, data sharing and its marketplace. We also haven't found anything we don't like about it, all of which has led to our consultants generally preferring it over the alternatives. Redshift is moving toward storage and compute separation, which has been a strong point of Snowflake, but even with Redshift Spectrum it isn't as convenient and flexible to use, partly because it is bound by its Postgres heritage (we do still like Postgres, by the way). Federated queries can be a reason to go with Redshift. When it comes to operations, Snowflake is much simpler to run. BigQuery, which is another alternative, is very easy to operate, but in a multicloud setup Snowflake is a better choice. We can also report that we've used Snowflake successfully with GCP, AWS, and Azure.

Variable fonts
Trial

Variable fonts are a way of avoiding the need to find and include separate font files for different weights and styles. Everything is in one font file, and you can use properties to select which style and weight you need. While not new, we still see sites and projects that could benefit from this simple approach. If you have pages that are including many variations of the same font, we suggest trying out variable fonts.

Apache Pinot
Assess

Apache Pinot is a distributed OLAP data store, built to deliver real-time analytics with low latency. It can ingest from batch data sources (such as Hadoop HDFS, Amazon S3, Azure ADLS or Google Cloud Storage) as well as stream data sources (such as Apache Kafka). If the need is user-facing, low-latency analytics, SQL-on-Hadoop solutions don't offer the low latency that is needed. Modern OLAP engines like Apache Pinot (or Apache Druid and Clickhouse among others) can achieve much lower latency and are particularly suited in contexts where fast analytics, such as aggregations, are needed on immutable data, possibly, with real-time data ingestion. Originally built by LinkedIn, Apache Pinot entered Apache incubation in late 2018 and has since added a plugin architecture and SQL support among other key capabilities. Apache Pinot can be fairly complex to operate and has many moving parts, but if your data volumes are large enough and you need low-latency query capability, we recommend you assess Apache Pinot.

Bit.dev
Assess

Bit.dev is a cloud-hosted collaborative platform for UI components extracted, modularized and reused with Bit. Web components have been around for a while, but building a modern front-end application by assembling small, independent components extracted from other projects has never been easy. Bit was designed to let you do exactly that: extract a component from an existing library or project. You can either build your own service on top of Bit for component collaboration or use Bit.dev.

DataHub
Assess

Since we first mentioned data discoverability in the Radar, LinkedIn has evolved WhereHows to DataHub, the next generation platform that addresses data discoverability via an extensible metadata system. Instead of crawling and pulling metadata, DataHub adopts a push-based model where individual components of the data ecosystem publish metadata via an API or a stream to the central platform. This push-based integration shifts the ownership from the central entity to individual teams making them accountable for their metadata. As more and more companies are trying to become data driven, having a system that helps with data discovery and understanding data quality and lineage is critical, and we recommend you assess DataHub in that capacity.

Feature Store
Assess

Feature Store is an ML-specific data platform that addresses some of the key challenges we face today in feature engineering with three fundamental capabilities: (1) it uses managed data pipelines to remove struggles with pipelines as new data arrives; (2) catalogs and stores feature data to promote discoverability and collaboration of features across models; and (3) consistently serves feature data during training and interference.

Since Uber revealed their Michelangelo platform, many organizations and startups have built their own versions of a feature store; examples include Hopsworks, Feast and Tecton. We see potential in Feature Store and recommend you carefully assess it.

JuiceFS
Assess

JuiceFS is an open-source, distributed POSIX file system built on top of Redis and an object store service (for example, Amazon S3). If you're building new applications, then our recommendation has always been to interact directly with the object store without going through another abstraction layer. However, JuiceFS can be an option if you're migrating legacy applications that depend on traditional POSIX file systems to the cloud.

Kafka API without Kafka
Assess

As more businesses turn to events as a way to share data among microservices, collect analytics or feed data lakes, Apache Kafka has become a favorite platform to support an event-driven architectural style. Although Kafka was a revolutionary concept in scalable persistent messaging, a lot of moving parts are required to make it work, including ZooKeeper, brokers, partitions, and mirrors. While these can be particularly tricky to implement and operate, they do offer great flexibility and power when needed, especially at an industrial enterprise scale. Because of the high barrier to entry presented by the full Kafka ecosystem, we welcome the recent explosion of platforms offering the Kafka API without Kafka. Recent entries such as Kafka on Pulsar and Redpanda offer alternative architectures, and Azure Event Hubs for Kafka provides some compatibility with Kafka producer and consumer APIs. Some features of Kafka, like the streams client library, are not compatible with these alternative brokers, so there are still reasons to choose Kafka over alternative brokers. It remains to be seen, however, if developers actually adopt this strategy or if it is merely an attempt by competitors to lure users away from the Kafka platform. Ultimately, perhaps Kafka's most enduring impact could be the convenient protocol and API provided to clients.

NATS
Assess

NATS is a fast, secure message queueing system with an unusually wide range of features and potential deployment targets. At first glance, you would be forgiven for asking why the world needs another message queueing system. Message queues have been around in various forms for nearly as long as businesses have been using computers and have undergone years of refinement and optimization for various tasks. But NATS has several interesting characteristics and is unique in its ability to scale from embedded controllers to global, cloud-hosted superclusters. We're particularly intrigued by NATS's intent to support a continuous streaming flow of data from mobile devices and IoT and through a network of interconnected systems. However, some tricky issues need to be addressed, not the least of which is ensuring consumers see only the messages and topics to which they're allowed access, especially when the network spans organizational boundaries. NATS 2.0 introduced a security and access control framework that supports multitenant clusters where accounts restrict a user's access to queues and topics. Written in Go, NATS has primarily been embraced by the Go language community. Although clients exist for pretty much all widely used programming languages, the Go client is by far the most popular. However, some of our developers have found that all the language client libraries tend to reflect the Go origins of codebase. Increasing bandwidth and processing power on small, wireless devices means that the volume of data businesses must consume in real time will only increase. Assess NATS as a possible platform for streaming that data within and among businesses.

Opstrace
Assess

Opstrace is an open-source observability platform intended to be deployed in the user's own network. If we don't use commercial solutions like Datadog (for example, because of cost or data residency concerns), the only solution is to build your own platform composed of open-source tools. This can take a lot of effort — Opstrace is intended to fill this gap. It uses open-source APIs and interfaces such as Prometheus and Grafana and adds additional features on top like TLS and authentication. At the heart of Opstrace runs a Cortex cluster to provide the scalable Prometheus API as well as a Loki cluster for the logs. It's fairly new and still misses

Platforms

As more businesses turn to events as a way to share data among microservices, collect analytics or feed data lakes, Apache Kafka has become a favorite platform. Because of the high barrier to entry presented by the full Kafka ecosystem, we welcome the recent explosion of platforms offering the Kafka API without Kafka.

(Kafka API without Kafka)

Platforms

Sometimes organizations build frameworks or abstractions on top of existing external products to cover very specific needs, thinking that the adaptation will provide them more benefits. However, they underestimate the required effort to keep those solutions evolving according to their needs, and soon realize that the original version is in much better shape than their own.

(Homemade infrastructure-as-code (IaC) products)

features when compared to solutions like Datadog or SignalFX. Still, it's a promising addition to this space and worth keeping an eye on.

Pulumi

Assess

We've seen interest in Pulumi slowly but steadily rising. Pulumi fills a gaping hole in the infrastructure coding world where Terraform maintains a firm hold. While Terraform is a tried-and-true standby, its declarative nature suffers from inadequate abstraction facilities and limited testability. Terraform is adequate when the infrastructure is entirely static, but dynamic infrastructure definitions call for a real programming language. Pulumi distinguishes itself by allowing configurations to be written in TypeScript/JavaScript, Python and Go — no markup language or templating required. Pulumi is tightly focused on cloud-native architectures — including containers, serverless functions and data services — and provides good support for Kubernetes. Recently, AWS CDK has mounted a challenge, but Pulumi remains the only cloud-neutral tool in this area. We're anticipating wider Pulumi adoption in the future and looking forward to a viable tool and knowledge ecosystem emerging to support it.

Redpanda

Assess

Redpanda is a streaming platform that provides a Kafka-compatible API, allowing it to benefit from the Kafka ecosystem without having to deal with the complexities of a Kafka installation. For example, Redpanda simplifies operations by shipping as a single binary and avoiding the need for an external dependency such as ZooKeeper. Instead, it implements the Raft protocol and performs comprehensive tests to validate it's been implemented correctly. One of Redpanda's capabilities (available for enterprise customers only) is inline WebAssembly (WASM) transformations, using an embedded WASM engine. This allows developers to create event transformers in their language of choice and compile it to WASM. Redpanda also offers much reduced tail latencies and increased throughput due to a series of optimizations. Redpanda is an exciting alternative to Kafka and is worth assessing.

Azure Machine Learning

Hold

We've observed before that the cloud providers push more and more services onto the market. We've also documented our concerns that sometimes the services are made available when they're not quite ready for prime time. Unfortunately, in our experience, Azure Machine Learning falls into the latter category. One of several recent entrants in the field of bounded low-code platforms, Azure ML promises more convenience for data scientists. Ultimately, however, it doesn't live up to its promise; in fact, it still feels easier for our data scientists to work in Python. Despite significant efforts, we struggled to make it scale and lack of adequate documentation proved to be another issue which is why we moved it to the Hold ring.

Homemade infrastructure-as-code (IaC) products

Hold

Products supported by companies or communities are in constant evolution, at least the ones that get traction in the industry. Sometimes organizations tend to build frameworks or abstractions on top of the existing external products to cover very specific needs, thinking that the adaptation will provide more benefits than the existing ones. We're seeing organizations trying to create homemade infrastructure-as-code (IaC) products on top of the existing ones; they underestimate the required effort to keep those solutions evolving according to their needs, and after a short period of time, they realize that the original version is in much better shape than their own; there are even cases where the abstraction on top of the external product reduces the original capabilities. Although we've seen success stories of organizations building homemade solutions, we want to caution about this approach as the effort required to do so isn't negligible, and a long-term product vision is required to have the expected outcomes.

TECHNOLOGY RADAR

Tools

Tools

Sentry
Adopt

Sentry has become the default choice for many of our teams when it comes to front-end error reporting. The convenience of features like the grouping of errors or defining patterns for discarding errors with certain parameters helps deal with the flood of errors coming from many end user devices. Integrating Sentry in your CD pipeline allows you to upload source maps for more efficient error debugging, and it helps easily trace back which errors occurred in which version of the software. We also appreciate that while Sentry is primarily a SaaS offering, its source code is publicly available and it can be used for free for smaller use cases and self-hosting.

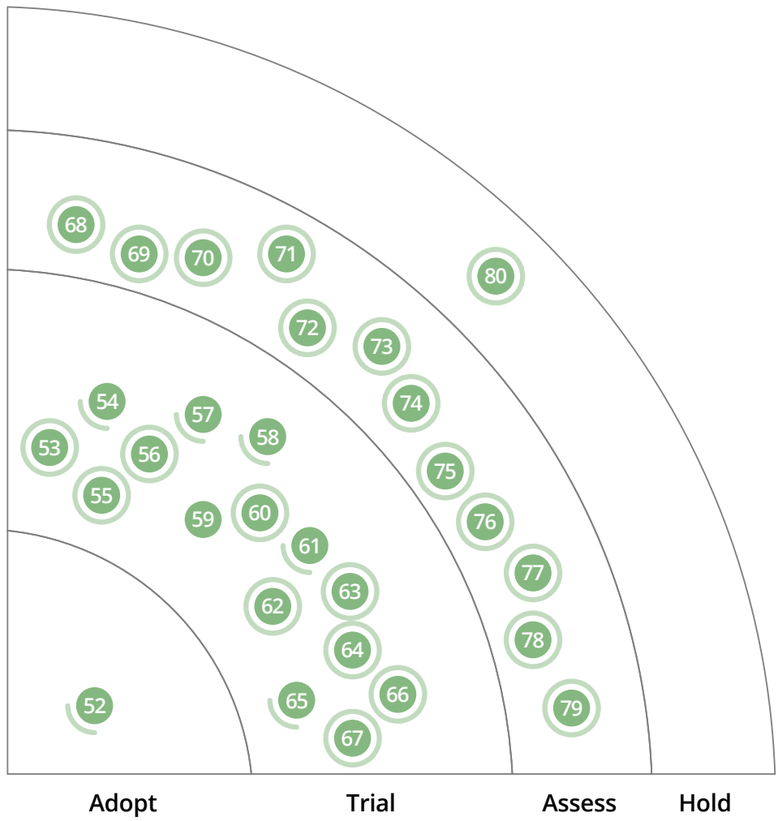
axe-core
Trial

Making the web inclusive requires serious attention to ensure accessibility is considered *and* validated at all stages of software delivery. Many of the popular accessibility testing tools are designed for testing after a web application is complete; as a result, issues are detected late and often are harder to fix, accumulating as debt. In our recent internal work on Thoughtworks websites, we included the open-source accessibility (a11y) testing engine axe-core as part of our build processes. It provided team members with early feedback on adherence to accessibility rules, even during early increments. Not every issue can be found through automated inspection, though.

Extending the functionality of axe-core is the commercially available axe DevTools, including functionality that guides team members through exploratory testing for a majority of accessibility issues.

dbt
Trial

Since we last wrote about dbt, we've used it in a few projects and like what we've seen. For example, we like that dbt makes the transformation part of ELT pipelines more accessible to consumers of the data as opposed to just the data engineers building the pipelines. It does this while encouraging good engineering practices such as versioning, automated testing and deployment. SQL continues to be the lingua franca of the data world (including databases, warehouses, query engines, data lakes and analytical platforms) and most of these systems support it to some extent. This allows dbt to be used against these systems for transformations by just building adaptors. The number of native connectors has grown to include Snowflake, BigQuery, Redshift and Postgres, as has the range of community plugins. We see tools like dbt helping data platforms become more "self service" capable.



Adopt
52. Sentry

Trial
53. axe-core
54. dbt
55. esbuild
56. Flipper
57. Great Expectations
58. k6
59. MLflow
60. OR-Tools
61. Playwright
62. Prowler
63. Pyright
64. Redash
65. Terratest
66. Tuple
67. Why Did You Render

Assess
68. Buildah and Podman
69. GitHub Actions
70. Graal Native Image
71. HashiCorp Boundary
72. imgcook
73. Longhorn
74. Operator Framework
75. Recommender
76. Remote - WSL
77. Spectral
78. Yelp detect-secrets
79. Zally

Hold
80. AWS CodePipeline

Tools

OR-Tools is an open-source software suite for solving combinatorial optimization problems. These optimization problems have a very large set of possible solutions, and tools like this are quite helpful in seeking the best solution.

(OR-Tools)

esbuild
Trial

We've always been keen to find tools that can shorten the software development feedback cycle; esbuild is such an example. As the front-end codebase grows larger, we usually face a packaging time of minutes. As a JavaScript bundler optimized for speed, esbuild can reduce this time by a factor of 10 to 100. It is written in Golang and uses a more efficient approach in the process of parsing, printing and source map generation which significantly surpasses build tools such as Webpack and Parcel in building time. esbuild may not be as comprehensive as those tools in JavaScript syntax transformation; however, this doesn't stop many of our teams from switching to esbuild as their default.

Flipper
Trial

Flipper is an extensible mobile application debugger. Out of the box it supports profiling, interactive layout inspection, log viewer and a network inspector for iOS, Android and React Native applications. Compared to other debugging tools for mobile apps, we find Flipper to be lightweight, feature rich and easy to set up.

Great Expectations
Trial

We wrote about Great Expectations in the previous edition of the Radar. We continue to like it and have moved it to Trial in this edition. Great Expectations is a framework that enables you to craft built-in controls that flag anomalies or quality issues in data pipelines. Just as unit tests run in a build pipeline, Great Expectations makes assertions during execution of a data pipeline. We like its simplicity and ease of use — the rules stored in JSON can be modified by our data domain experts without necessarily needing data engineering skills.

k6
Trial

We've had a bit more experience performance testing with k6 since we first covered it in the Radar, and with good results. Our teams have enjoyed the focus on the developer experience and flexibility of the tool. Although it's easy to get started with k6 all on its own, it really shines with its ease of integration into a developer ecosystem. For example, using the Datadog adapter, one team was quickly able to visualize performance in a distributed system and identify significant concerns before releasing the system to production. Another team, with the commercial version of k6, was able to use the Azure pipelines marketplace extension to wire performance tests into their CD pipeline and get Azure DevOps reporting with little effort. Since k6 supports thresholds that allow for automated testing assertions out of the box, it's relatively easy to add a stage to your pipeline that detects performance degradation of new changes, adding a powerful feedback mechanism for developers.

MLflow
Trial

MLflow is an open-source tool for machine-learning experiment tracking and lifecycle management. The workflow to develop and continuously evolve a machine-learning model includes a series of experiments (a collection of runs), tracking the performance of these experiments (a collection of metrics) and tracking and tweaking models (projects). MLflow facilitates this workflow nicely by supporting existing open standards and integrates well with many other tools in the ecosystem. MLflow as a managed service by Databricks on the cloud, available in AWS and Azure, is rapidly maturing, and we've used it successfully in our projects. We find MLflow a great tool for model management and tracking, supporting both UI-based and API-based interaction models. Our only growing concern is that MLflow is attempting to deliver too many conflating concerns as a single platform, such as model serving and scoring.

OR-Tools
Trial

OR-Tools is an open-source software suite for solving combinatorial optimization problems. These optimization problems have a very large set of possible solutions, and tools like OR-Tools are quite helpful in seeking the best solution. You can model the problem in any one of the supported languages — Python, Java, C# or C++ — and choose the solvers from several supported open-source or commercial solvers. We've successfully used OR-Tools in multiple optimization projects with integer and mixed-integer programming.

Playwright
Trial

Playwright allows you to write Web UI tests for Chromium and Firefox as well as WebKit, all through the same API. The tool

has gained some attention for its support of all the major browser engines which it achieves by including patched versions of Firefox and Webkit. We continue to hear positive experience reports with Playwright, in particular its stability. Teams have also found it easy to migrate from Puppeteer, which has a very similar API.

Prowler
Trial

We welcome the increased availability and maturity of infrastructure configuration scanning tools: Prowler helps teams scan their AWS infrastructure setups and improve security based on the results. Although Prowler has been around for a while, it has evolved a lot over the past few years, and we've found it very valuable to enable teams to take responsibility for proper security with a short feedback loop. Prowler categorizes AWS CIS benchmarking checks into different groups (Identity and Access Management, Logging, Monitoring, Networking, CIS Level 1, CIS Level 2, EKS-CIS), and it includes many checks that help you gain insights into your PCI DSS and GDPR compliance.

Pyright
Trial

While duck typing is certainly seen as a feature by many Python programmers, sometimes — especially for larger codebases — type checking can be useful, too. For that reason a number of type annotations are proposed as Python Enhancement Proposals (PEPs), and Pyright is a type checker that works with these annotations. In addition, it provides some type inference and guards that understand conditional code flow constructs. Designed with large codebases in mind, Pyright is fast, and its watch mode checks happen incrementally as files are changed to further shorten the feedback cycle. Pyright can be used directly on the command line, but integrations for VS Code, Emacs, vim, Sublime, and possibly other editors are available, too. In our experience, Pyright is preferable to alternatives like mypy.

Redash
Trial

Adopting a "you build it, you run it" DevOps philosophy means teams have increased attention on both technical and business metrics that can be extracted from the systems they deploy. Often we find that analytics tooling is difficult to access for most developers, so the work to capture and present metrics is left to other teams — long after features are shipped to end users. Our teams have found Redash to be very useful for querying product metrics and creating dashboards in a way that can be self-served by general developers, shortening feedback cycles and focusing the whole team on the business outcomes.

Terratest
Trial

Terratest caught our attention in the past as an interesting option for infrastructure testing. Since then, our teams have been using it, and they're very excited about it because of its stability and the experience it provides. Terratest is a Golang library that makes it easier to write automated tests for infrastructure code. Using infrastructure-as-code tools such as Terraform, you can create real infrastructure components (such as servers, firewalls, or load balancers) to deploy applications on them and then validate the expected behavior using Terratest. At the end of the test, Terratest can undeploy the apps and clean up resources. This makes it largely useful for end-to-end tests of your infrastructure in a real environment.

Tuple
Trial

Tuple is a relatively new tool optimized for remote paired programming, designed to fill the gap Slack left in the marketplace after abandoning Screenhero. Although it still exhibits some growing pains — platform availability is limited to Mac OS for now (with Linux support coming soon), and it has some UI quirks to work through — we've had good experience using it within those constraints. Unlike general-purpose video- and screen-sharing tools like Zoom, Tuple supports dual control with two mouse cursors, and unlike options such as Visual Studio Live Share, it isn't tied to an IDE. Tuple supports voice and video calls, clipboard sharing, and lower latency than general-purpose tools; and its ability to let you draw and erase in your pair's screen with ease makes Tuple a very intuitive and developer-friendly tool.

Why Did You Render
Trial

When working with React, we often encounter situations where our page is very slow because some components are re-rendering when they shouldn't be. Why Did You Render is a library that helps detect why

Tools

Tuple is a relatively new remote pair programming tool, designed to fill the gap Slack left in the marketplace after abandoning Screenhero.

(Tuple)

Tools

imgcook is a SaaS product from Alibaba, which can intelligently transform various design files—Sketch, PSD, even static images—into front-end code.

(imgcook)

a component is re-rendering. It does this by monkey patching React. We've used it in a few of our projects to debug performance issues with great effect.

Buildah and Podman

Assess

Even though Docker has become the sensible default for containerization, we're seeing new players in this space that are catching our attention. That is the case for Buildah and Podman, which are complementary projects to build images (Buildah) and run containers (Podman) using a rootless approach in multiple Linux distributions. Podman introduces a daemonless engine for managing and running containers which is an interesting approach in comparison to what Docker does. The fact that Podman can use either Open Container Initiative (OCI) images built by Buildah or Docker images makes this tool even more attractive and easy to use.

GitHub Actions

Assess

CI servers and build tools are some of the oldest and most widely used in our kit. They run the gamut from simple cloud-hosted services to complex, code-defined pipeline servers that support fleets of build machines. Given our experience and the wide range of options already available, we were initially skeptical when GitHub Actions were introduced as another mechanism to manage the build and integration workflow. But the opportunity for developers to start small and easily customize behavior means that GitHub Actions are moving toward the default category for smaller projects. It's hard to argue with the convenience of having the build tool integrated directly into the source code repository. An enthusiastic community has emerged around this feature and that means a wide range of user-contributed tools and workflows are available to get started. Tools vendors are also getting on board via the GitHub Marketplace. However, we still recommend you proceed with caution. Although code and Git history can be exported into alternative hosts, a development workflow based on GitHub Actions can't. Also, use your best judgment to determine when a project is large or complex enough to warrant an independently supported pipeline tool. But for getting up and running quickly on smaller projects, it's worth considering GitHub Actions and the ecosystem that is growing around them.

Graal Native Image

Assess

Graal Native Image is a technology that compiles Java code into an operating system's native binary — in the form of a statically linked executable or a shared library. A native image is optimized to reduce the memory footprint and startup time of an application. Our teams have successfully used Graal native images, executed as small Docker containers, in the serverless architecture where reducing start time matters. Although designed for use with programming languages such as Go or Rust that natively compile and require smaller binary sizes and shorter start times, Graal Native Image can be equally useful to teams that have other requirements and want to use JVM-based languages.

Graal Native Image Builder, *native-image*, supports JVM-based languages — such as Java, Scala, Clojure and Kotlin — and builds executables on multiple operating systems including Mac OS, Windows and multiple distributions of Linux. Since it requires a closed-world assumption, where all code is known at compile time, additional configuration is needed for features such as *reflection* or *dynamic class loading* where types can't be deduced at build time from the code alone.

HashiCorp Boundary

Assess

HashiCorp Boundary combines the secure networking and identity management capabilities needed for brokering access to your hosts and services in one place and across a mix of cloud and on-premise resources if needed. Key management can be done by integrating the key management service of your choice, be it from a cloud vendor or something like HashiCorp Vault. HashiCorp Boundary supports a growing number of identity providers and can be integrated with parts of your service landscape to help define permissions, not just on host but also on a service level. For example, it enables you to control fine-grained access to a Kubernetes cluster, and dynamically pulling in service catalogs from various sources is on the roadmap. All of this stays out of the way of the engineering end users who get the shell experience they're used to, securely connected through Boundary's network management layer.

imgcook

Assess

Remember the research project pix2code that showed how to automatically generate code from GUI screenshots? Now there is a productized version of this technique —

imgcook is a SaaS product from Alibaba that can intelligently transform various design files (Sketch/PSD/static images) into front-end code. Alibaba needs to customize a large number of campaign pages during the Double Eleven shopping festival. These are usually one-time pages that need to be developed quickly. Through the deep-learning method, the UX's design is initially processed into front-end code and then adjusted by the developer. Our team is evaluating this tech: although the image processing takes place on the server side while the main interface is on the web, imgcook provides tools that could integrate with the software design and development lifecycle. imgcook can generate static code as well as some data-binding component code if you define a DSL. The technology is not perfect yet; designers need to refer to certain specifications to improve the accuracy of code generation (which still needs to be adjusted by developers afterward). We've always been cautious about magic code generation, because the generated code is usually difficult to maintain in the long run, and imgcook is no exception. But if you limit the usage to a specific context, such as one-time campaign pages, it's worth a try.

Longhorn

Assess

Longhorn is a distributed block storage system for Kubernetes. There are many persistent storage options for Kubernetes; unlike most, however, Longhorn is built from the ground up to provide incremental snapshots and backups, thereby easing the pain of running a replicated storage for non–cloud-hosted Kubernetes. With the recent experimental support for ReadWriteMany (RWX) you can even mount the same volume for read and write access across many nodes. Choosing the right storage system for Kubernetes is a nontrivial task, and we recommend you assess Longhorn based on your needs.

Operator Framework

Assess

Operator Framework is a set of open-source tools that simplifies building and managing the lifecycle of Kubernetes operators. The Kubernetes operator pattern, originally introduced by CoreOS, is an approach to encapsulate the knowledge of operating an application using Kubernetes native capabilities; it includes *resources* to be managed and *controller* code that ensures the resources are matching their target state. This approach has been used to extend Kubernetes to manage many applications, particularly the stateful ones, natively. Operator Framework has three components: Operator SDK, which simplifies building, testing and packaging Kubernetes operators; Operator lifecycle manager to install, manage and upgrade the operators; and a catalog to publish and share third-party operators. Our teams have found Operator SDK particularly powerful in rapidly developing Kubernetes-native applications.

Recommender

Assess

The number of services offered by the big cloud providers keeps growing, but so does the convenience and maturity of tools that help you use them securely and efficiently. Recommender is a service on Google Cloud that analyzes your resources and gives you recommendations on how to optimize them based on your actual usage. The service consists of a range of "recommenders" in areas such as security, compute usage or cost savings. For example, the IAM Recommender helps you better implement the principle of least privilege by pointing out permissions that are never actually used and therefore are potentially too broad.

Remote - WSL

Assess

Over the past few years Windows Subsystem for Linux (WSL) has come up a few times in our discussions. Although we liked what we saw, including the improvements in WSL 2, it never made it into the Radar. In this edition we want to highlight an extension for Visual Studio Code that greatly improves the experience working with WSL. Although Windows-based editors could always access files on a WSL file system, they were unaware of the isolated Linux environment. With the Remote - WSL extension, Visual Studio Code becomes aware of WSL, allowing developers to launch a Linux shell. This also enables debugging of binaries running inside WSL from Windows. Jetbrains' IntelliJ too has seen steady improvement in its support for WSL.

Spectral

Assess

One of the patterns we've seen repeat itself in this publication is that static error- and style-checking tools emerge quickly after a new language gains popularity. These tools are generically known as linters — after the classic and

Tools

Recommender is a service on Google Cloud that analyzes cloud resources and provides a recommendation for optimization based on actual usage.

(Recommender)

Tools

Zally is a minimalist OpenAPI linter that helps ensure an API conforms to the team's API style guide.

(Zally)

beloved Unix utility lint, which statically analyzes C code. We like these tools because they catch errors early, before code even gets compiled. The latest instance of this pattern is Spectral, a linter for YAML and JSON. Although Spectral is a generic tool for these formats, its main target is OpenAPI (the evolution of Swagger) and AsyncAPI. Spectral ships with a comprehensive set of out-of-the-box rules for these specs that can save developers headaches when designing and implementing APIs or event-driven collaboration. These rules check for proper API parameter specifications or the existence of a license statement in the spec, among other things. While this tool is a welcome addition to the API development workflow, it does raise the question of whether a non-executable specification should be so complex as to require an error-checking technique designed for programming languages. Perhaps developers should be writing code instead of specs?

Yelp detect-secrets

Assess

Yelp detect-secrets is a Python module for detecting secrets within a codebase; it scans files within a directory looking for secrets. It can be used as a Git pre-commit hook or to perform a scan in multiple places within the CI/CD pipeline. It comes with a default configuration that makes it very easy to use but can be modified to suit your needs. You can also install custom plugins to add to its default heuristic searches. Compared to similar offerings, we found that this tool detects more types of secrets with its out-of-the-box configuration.

Zally

Assess

As the API specification ecosystem matures, we're seeing more tools built to automate style checks. Zally is a minimalist OpenAPI linter that helps to ensure an API conforms to the team's API style guide. Out of the box, it will validate against a rule set developed for Zalando's API style guide, but it also supports a Kotlin extension mechanism to develop custom rules. Zally includes a web UI that provides an intuitive interface for understanding style violations and includes a CLI that makes it easy to plug into your CD pipeline.

AWS CodePipeline

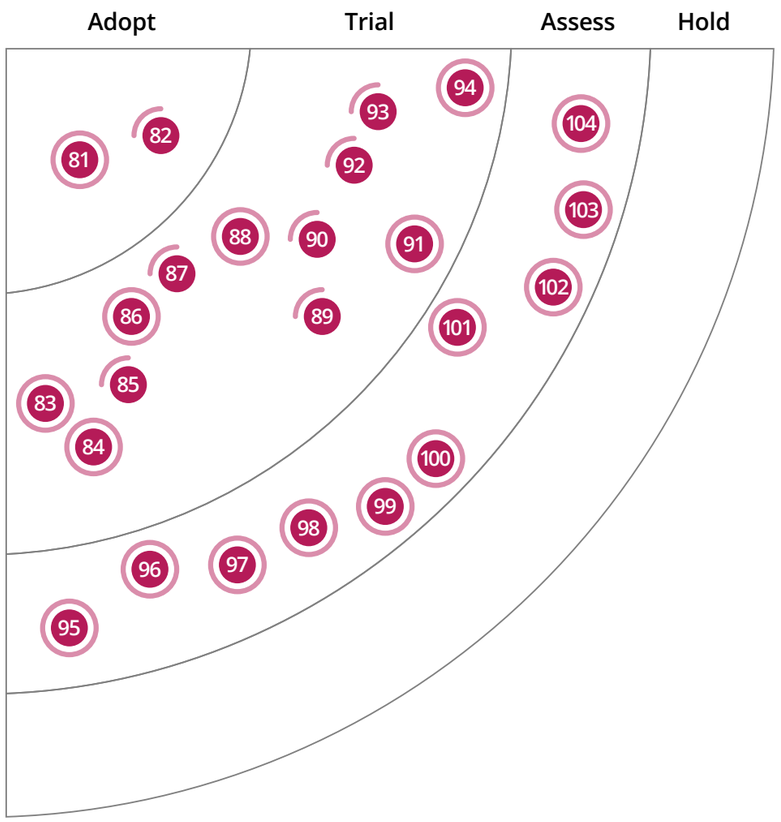
Hold

Based on the experiences of multiple Thoughtworks teams we suggest approaching AWS CodePipeline with caution. Specifically, we've found that once teams move beyond simple pipelines, this tool can become hard to work with. While it may seem like a "quick win" when first starting out with AWS, we suggest taking a step back and checking whether AWS CodePipeline will meet your longer-term needs, for example, pipeline fan-out and fan-in or more complex deployment and testing scenarios featuring nontrivial dependencies and triggers.

TECHNOLOGY RADAR

Languages & Frameworks

Languages & Frameworks



Combine
Adopt

A long time ago we placed ReactiveX — a family of open-source frameworks for reactive programming — into the Adopt ring of the Radar. In 2017, we mentioned the addition of RxSwift, which brought reactive programming to iOS development using Swift. Since then, Apple has introduced its own take on reactive programming in the form of the Combine framework. Combine has become our default choice for apps that support iOS 13 as an acceptable deployment target. It's easier to learn than RxSwift and integrates really well with SwiftUI. If you're planning to convert an existing application from RxSwift to Combine or work with both in the same project, you might want to look at RxCombine.

LeakCanary
Adopt

Our mobile teams now view LeakCanary as a good default choice for Android development. It detects annoying memory leaks in Android apps, is extremely simple to hook up and provides notifications with a clear trace-back to the cause of the leak. LeakCanary can save you tedious hours troubleshooting out-of-memory errors on multiple devices, and we recommend you add it to your toolkit.

Angular Testing Library
Trial

As we continue developing web applications in JavaScript, we continue enjoying the Testing Library approach of testing applications; and carry on exploring and gaining experience with its packages — beyond that of React Testing Library. Angular Testing Library brings all the benefits of its family when testing UI components in a user-centric way, pushing for more maintainable tests focused primarily on behavior rather than testing UI implementation details. Although it falls short in documentation, Angular Testing Library does provide good sample tests that helped us in getting started faster for various cases. We've had great success with this testing library in our Angular projects and advise you to trial this solid testing approach.

AWS Data Wrangler
Trial

AWS Data Wrangler is an open-source library that extends the capabilities of Pandas to AWS by connecting data frames

Adopt
81. Combine
82. LeakCanary

Trial
83. Angular Testing Library
84. AWS Data Wrangler
85. Blazor
86. FastAPI
87. io-ts
88. Kotlin Flow
89. LitElement
90. Next.js
91. On-demand modules
92. Streamlit
93. SWR
94. TrustKit

Assess
95. .NET 5
96. bUnit
97. Dagster
98. Flutter for Web
99. Jotai and Zustand
100. Kotlin Multiplatform Mobile
101. LVGL
102. React Hook Form
103. River
104. Webpack 5 Module Federation

Hold

Languages & Frameworks

FastAPI is a modern, high-performance web framework for building APIs with Python 3.6 and later.

(FastAPI)

to AWS data-related services. In addition to Pandas, this library leverages Apache Arrow and Boto3 to expose several APIs to load, transform and save data from data lakes and data warehouses. An important limitation is that you can't do large distributed data pipelines with this library. However, you can leverage the native data services — like Athena, Redshift and Timestream — to do the heavy lifting and pull data in order to express complex transformations that are well suited for data frames. We've used AWS Data Wrangler in production and like that it lets you focus on writing transformations without spending too much time on the connectivity to AWS data services.

Blazor
Trial

Although JavaScript and its ecosystem is dominant in the web UI development space, new opportunities are opening up with the emergence of WebAssembly. Blazor continues to demand our attention; it's producing good results with our teams building interactive rich user interfaces using C# on top of WebAssembly. The fact that our teams can use C# on the frontend too allows them to share code and reuse existing libraries. That, along with the existing tooling for debugging and testing, such as bUnit, make this open-source framework worth trying.

FastAPI
Trial

We're seeing more teams adopting Python as the preferred language to build solutions, not just for data science but for back-end services too. In these scenarios, we're having good experiences with FastAPI — a modern, fast (high-performance), web framework for building APIs with Python 3.6 or later. Additionally, this framework and its ecosystem include features such as API documentation using OpenAPI that allow our teams to focus on the business functionalities and quickly create REST APIs, which makes FastAPI a good alternative to existing solutions in this space.

io-ts
Trial

We've really enjoyed using TypeScript for a while now and love the safety that the strong typing provides. However, getting data into the bounds of the type system — from, for example, a call to a back-end service — can lead to run-time errors. One library that helps solve this problem is io-ts. It bridges the gap between compile-time type-checking and run-time consumption of external data by providing encode and decode functions. It can also be used as a custom type guard. As we gain more experience with io-ts in our work, our initially positive impressions are confirmed, and we still like the elegance of its approach.

Kotlin Flow
Trial

The introduction of coroutines to Kotlin opened the door for several innovations — Kotlin Flow is one of them, directly integrated into the coroutines library. It's an implementation of Reactive Streams on top of coroutines. Unlike RxJava, flows are a native Kotlin API similar to the familiar sequence API with methods that include map and `filter`. Like sequences, flows are *cold*, meaning that the values of the sequence are only constructed when needed. All of this makes writing multithreaded code much simpler and easier to understand than other approaches. The `toList` method, predictably, converts a flow into a list which is a common pattern in tests.

LitElement
Trial

Steady progress has been made since we first wrote about Web Components in 2014. LitElement, part of the Polymer Project, is a simple library that you can use to create lightweight web components. It's really just a base class that removes the need for a lot of the common boilerplate, making writing web components a lot easier. We've had success using it on projects, and as we see the technology maturing and the library being well liked, LitElement is becoming more commonly used in our Web Components-based projects.

Next.js
Trial

We've had a bit more experience using Next.js for React codebases since the last time we wrote about it. Next.js is an opinionated, zero-configuration framework that includes simplified routing, automatic compilation and bundling with Webpack and Babel, fast hot reloading for a convenient

developer workflow among other features. It provides server-side rendering by default, improves search engine optimization and the initial load time and supports incremental static generation. We've had positive experience reports from teams using Next.js and, given its large community, continue to be excited about the evolution of the framework.

On-demand modules
Trial

On-demand modules for Android is a framework that allows tailored APKs containing only required functionality to be downloaded and installed for a suitably structured app. This could be worth trialing for larger apps where download speed might be an issue, or if a user is likely only to use some functionality on initial installation. It can also simplify the handling of multiple devices without requiring different APKs. A similar framework is available for iOS.

Streamlit
Trial

Streamlit is an open-source application framework in Python used by data scientists for building interactive data applications. Tuning machine-learning models takes time; instead of going back and forth on the main application (the one that uses these models), we've found value in quickly building standalone prototypes in Streamlit and gathering feedback during experimentation cycles. Streamlit stands out from competitors such as Dash because of its focus on rapid prototyping and support for a wide range of visualization libraries, including Plotly and Bokeh. We're using it in a few projects and like how we can put together interactive visualizations with very little effort.

SWR
Trial

When used in appropriate circumstances, our teams have found that the React Hooks library SWR can result in cleaner code and much improved performance. SWR implements the stale-while-revalidate HTTP caching strategy, first returning data from cache (stale), then sending the fetch request (revalidate) and finally refreshing the values with the up-to-date response. We caution teams to only use the SWR caching strategy when an application is supposed to return stale data. Note that HTTP requires that caches respond to a request with the most up-to-date response; only in *carefully considered circumstances* is a stale response allowed to be returned.

TrustKit
Trial

SSL public key pinning is tricky. If you select the wrong policy or don't have a backup pin, your application will stop working unexpectedly. This is where TrustKit is useful — it's an open-source framework that makes SSL public key pinning easier for iOS applications. There is an equivalent framework for Android as well. Picking the correct pinning strategy is a nuanced topic, and you can find more details about it in the TrustKit Getting Started guide. We've used TrustKit in several projects in production, and it has worked out well.

.NET 5
Assess

We don't call out every new .NET version in the Radar, but .NET 5 represents a significant step forward in bringing .NET Core and .NET Framework into a single platform. Organizations should start to develop a strategy to migrate their development environments — a fragmented mix of frameworks depending on the deployment target — to a single version of .NET 5 or 6 when it becomes available. The advantage of this approach will be a common development platform regardless of the intended environment: Windows, Linux, cross-platform mobile devices (via Xamarin) or the browser (using Blazor). While polyglot development will remain the preferred approach for companies with the engineering culture to support it, others will find it more efficient to standardize on a single platform for .NET development. For now, we want to keep this in the Assess ring to see how well the final unified framework performs in .NET 6.

bUnit
Assess

bUnit is a testing library for Blazor that makes it easy to create tests for Blazor components in existing unit testing frameworks such as NUnit, xUnit or MSUnit. It provides a facade around the component allowing it to be run and tested within the familiar unit test paradigm, thus allowing very fast feedback and testing of the component in isolation. If you're developing for Blazor, we recommend that you add bUnit to your list of tools to try out.

Languages & Frameworks

Streamlit is an open-source application framework in Python used by data scientists for building interactive data visualizations.

(Streamlit)

Languages & Frameworks

These state management libraries for React aim to be small and simple to use. Perhaps not by complete coincidence, both names are translations of the word state into Japanese and German, respectively.

(Jotai and Zustand)

Dagster
Assess

Dagster is an open-source data orchestration framework for machine learning, analytics and plain ETL data pipelines. Unlike other task-driven frameworks, Dagster is aware of data flowing through the pipeline and can provide type-safety. With this unified view of pipelines and assets produced, Dagster can schedule and orchestrate Pandas, Spark, SQL or anything else that Python can invoke. The framework is relatively new, and we recommend that you assess its capabilities for your data pipelines.

Flutter for Web
Assess

So far, Flutter has primarily supported native iOS and Android applications. However, the Flutter team's vision is to support building applications on every platform. Flutter for Web is one step in that direction — it allows us to build apps for iOS, Android and the browser from the same codebase. It has been available for over a year now on the "Beta" channel, but with the recent Flutter 2.0 release, Flutter for Web has hit the stable milestone. In the initial release of web support, the Flutter team is focusing on progressive web apps, single-page apps and expanding existing mobile apps to the web. The application and framework code (all in Dart) are compiled to JavaScript instead of ARM machine code, which is used for mobile applications. Flutter's web engine offers a choice of two renderers: an HTML renderer, which uses HTML, CSS, Canvas and SVG, and a CanvasKit renderer that uses WebAssembly and WebGL to render Skia paint commands to the browser canvas. A few of our teams have started using Flutter for Web and like the initial results.

Jotai and Zustand
Assess

In the previous Radar, we commented on the beginning of a phase of experimentation with state management in React applications. We moved Redux back into the Trial ring, documenting that it is no longer our default choice, and we mentioned Facebook's Recoil. In this volume we want to highlight Jotai and Zustand: Both are state management libraries for React; both aim to be small and simple to use; and, perhaps not by complete coincidence, both names are translations of the word state into Japanese and German, respectively. Beyond these similarities, however, they differ in their design. Jotai's design is closer to that of Recoil in that state consists of atoms stored within the React component tree, whereas Zustand stores the state outside of React in a single state object, much like the approach taken by Redux. The authors of Jotai provide a helpful checklist to decide when to use which.

Kotlin Multiplatform Mobile
Assess

Following the trend of cross-platform mobile development, Kotlin Multiplatform Mobile (KMM) is a new entry in this space. KMM is an SDK provided by JetBrains that leverages the multiplatform capabilities in Kotlin and includes tools and features designed to make the end-to-end experience of building mobile cross-platform applications more enjoyable and efficient. With KMM you write code once for business logic and the app core in Kotlin and then share it with both Android and iOS applications. Write platform-specific code only when necessary, for example, to take advantage of native UI elements; and the specific code is kept in different views for each platform. Although still in Alpha, Kotlin Multiplatform Mobile is evolving rapidly. We'll certainly keep an eye on it, and you should too.

LVGL
Assess

With the increasing popularity of smart home and wearable devices, demand for intuitive graphical user interfaces (GUIs) is increasing. However, if you're engaged in embedded device development, rather than Android/iOS, GUI development may take a lot of effort. As an open-source embedded graphics library, LVGL has become increasingly popular. LVGL has been adapted to mainstream embedded platforms such as NXP, STM32, PIC, Arduino, and ESP32. It has a very small memory footprint: 64 kB flash and 8 kB RAM is enough to make it work, and it can run smoothly on various Cortex-M0 low-power MCUs. LVGL supports input types such as touchscreen, mouse and buttons and contains more than 30 controls, including TileView suitable for smart watches. The MIT license it chose doesn't restrict enterprise and commercial use. Our teams' feedback on this tool has been positive and one of our projects using LVGL is already in production, more specifically in small batch manufacturing.

React Hook Form
Assess

Building forms for the web remains one of the perennial challenges of front-end development, in particular with React. Many of our teams working with React have been using Formik to make this easier, but some are now assessing React Hook Form as a potential alternative. React Hooks already existed when React Hook Form was created, so it could use them as a first-class concept: the framework is registering and tracking form elements as uncontrolled components via a hook, thereby significantly reducing the need for re-rendering. It's also quite lightweight in size and in the amount of boilerplate code needed.

River
Assess

At the heart of many approaches to machine learning lies the creation of a model from a set of training data. Once a model is created, it can be used over and over again. However, the world isn't stationary, and often the model needs to change as new data becomes available. Simply re-running the model creation step can be slow and costly. Incremental learning addresses this issue, making it possible to learn from streams of data incrementally to react to change faster. As a bonus the compute and memory requirements are lower and predictable. In our implementations we've had good experience with the River framework, but so far we've added checks, sometimes manual, after updates to the model.

Webpack 5 Module Federation
Assess

The release of the Webpack 5 Module Federation feature has been highly anticipated by developers of micro frontend architectures. The feature introduces a more standardized way to optimize how module dependencies and shared code are managed and loaded. Module federation allows for the specification of shared modules, which helps with the deduplication of dependencies across micro frontends by loading code used by multiple modules only once. It also lets you distinguish between local and remote modules, where the remote modules are not actually part of the build itself but loaded asynchronously. Compared to build-time dependencies like npm packages, this can significantly simplify the deployment of a module update with many downstream dependencies. Be aware, though, that this requires you to bundle all of your micro frontends with Webpack, as opposed to approaches such as import maps, which might eventually become part of the W3C standard.

Languages & Frameworks

Machine-learning models often need to change as new data becomes available. Incremental learning makes it possible to learn from streams of data incrementally to react to change faster. In our implementations we've had good experience with River, a Python library for online machine learning.

(River)

/thoughtworks

We are a software consultancy and community of passionate purpose-led individuals, 9,000+ people strong across 48 offices in 17 countries. Over our 27+ year history, we have helped our clients solve complex business problems where technology is the differentiator. When the only constant is change, we prepare you for the unpredictable.

Want to stay up to date with all Radar-related news and insights?

Follow us on your favorite social channel or become a subscriber.

subscribe now

/thoughtworks

thoughtworks.com/radar

#TWTechRadar