Technology Radar

An opinionated guide to technology frontiers

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.

Hold   Assess   Trial   Adopt
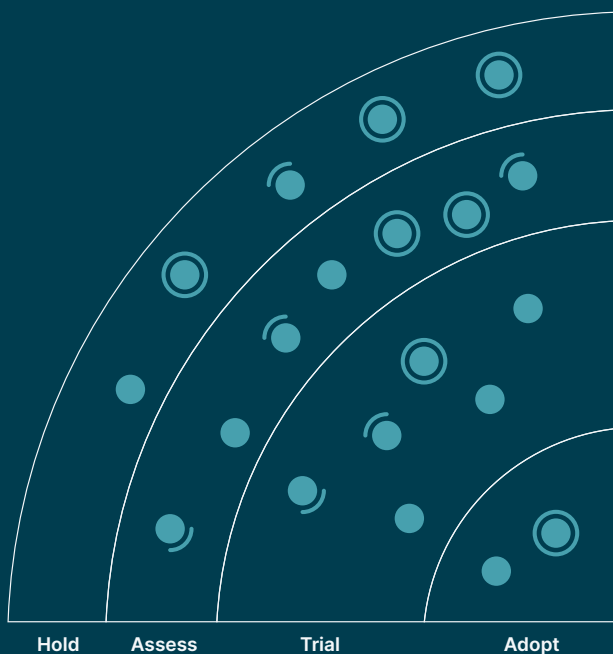
Adopt: We feel strongly that the industry should be adopting these items. We use them when appropriate in our projects.

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.

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.

Contributors

The Technology Advisory Board (TAB) is a group of 18 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 Falconi Crispim
Cassie Shum
Erik Doernenburg
Fausto de la Torre
Hao Xu
Ian Cartwright
James Lewis
Lakshminarasimhan Sudarshan
Mike Mason
Neal Ford
Perla Villarreal
Scott Shaw
Shangqi Liu
Zhamak Dehghani
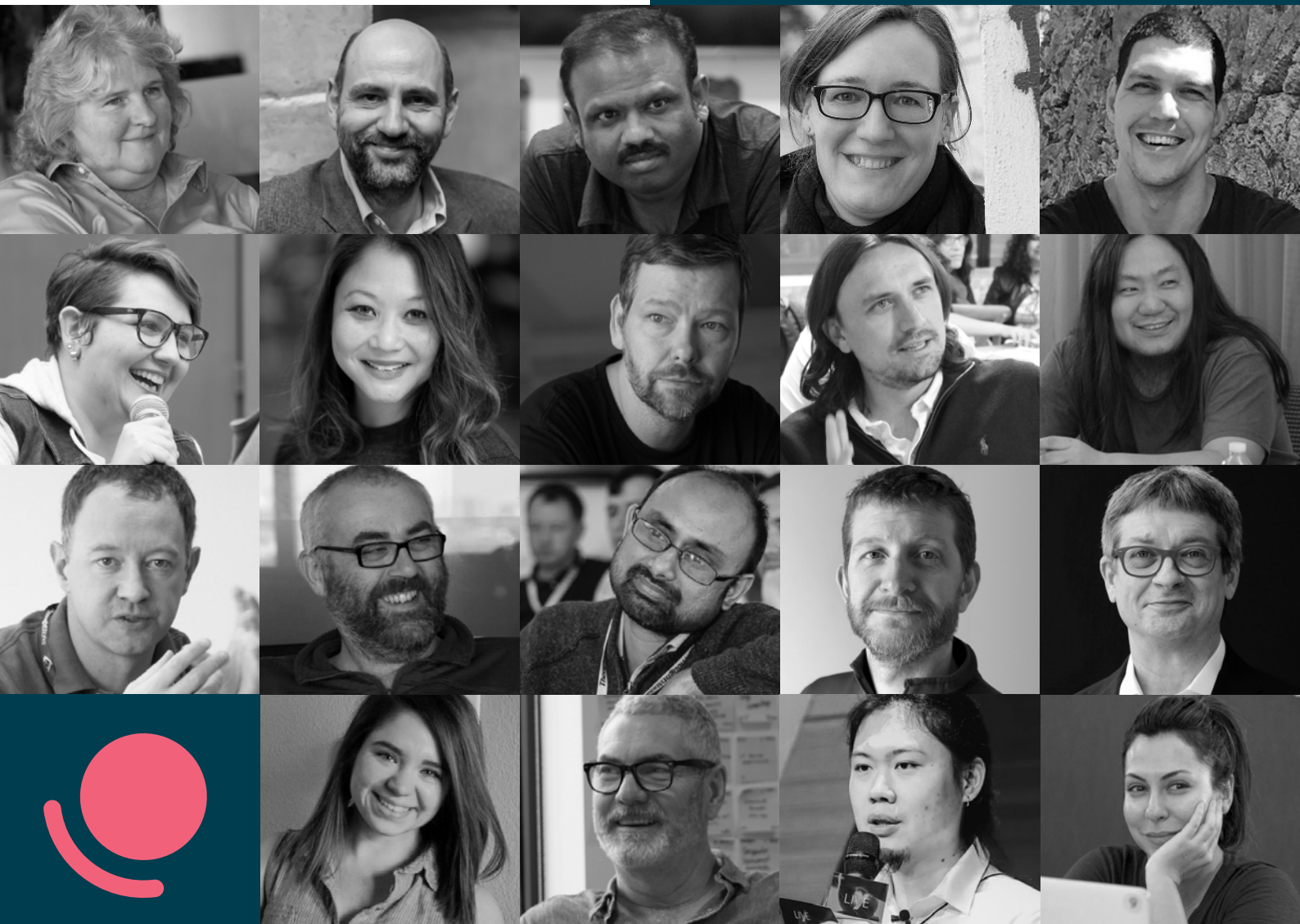
Themes

Adapting Kafka

We discussed several topics in this edition of the Radar (some of which eventually failed to make the final cut) where teams are employing tools to adapt to/from Kafka. Some of these tools allow more traditional interfaces to Kafka (such as **ksqlDB**, **Confluent Kafka REST Proxy**, and Nakadi), while others are designed to provide extra services such as GUI frontends and orchestration add-ons. We suspect that part of the underlying reason for this bounty of tools is the underlying sharp-edged complexity of some of Kafka's parts combined with increased presence in organizations that need to bend it to existing architectures and processes. Some teams end up treating **Kafka as a next-generation enterprise service bus** — one example of *The slippery slope of convenience* theme — but other teams use Kafka to provide universal access to business events as they happen. These organizations recognize that it is sometimes easier to have a centralized infrastructure with adaptation at the edges and try to avoid sprawl with careful design and governance. In any case, it shows that Kafka continues toward status as a de facto standard for asynchronous publish/subscribe messaging at volume.

The slippery slope of convenience

An antipattern as old as the Radar is the tendency for teams to place behavior within their ecosystem at convenient but improper nexus points that lead to long-term technical debt and worse. Examples abound, including using a database as an integration point, using **Kafka** as a global orchestrator, intermingling business logic with infrastructure code and so on. Modern software development offers many places for developers to stash behavior, and inexperienced or inconsiderate teams often entangle concerns by not carefully considering the long-term consequences of inappropriate coupling. Inappropriate team structures and other deviations from **Conway's Law** don't help either. As software systems become more complex, development teams must show diligence to both create and maintain thoughtful architecture and design, not slap-dash decisions for expediency. Often, thinking about the testability of a particular approach leads teams away from some of these potentially problematic decisions. Software tends toward complexity when left to its own devices. Careful design and, perhaps more importantly, ongoing governance works to ensure that schedule pressure or one of the other numerous disruptive forces doesn't cause teams to make convenient but improper decisions.

Conway's is still the law

Many architects cite **Conway's Law**, the 1960s observation that teams' communication structures influence design, to justify changes to team organization, and we discovered across several nominated blips in this edition that an organization's team structure remains a key enabler when handled well and a serious impediment when handled poorly. Examples we discuss include the need for product thinking around platform teams rather than treating them as order-takers; **Team Topologies** and the increasing recognition of **team cognitive load** in relation to effectiveness; and the new framework developed around programmer productivity called **SPACE**. Organizations spend enormous funds on tools, yet many find better productivity gains by paying attention to the people who build the software and what makes them effective within a particular organization.

Clever tech we shouldn't need

Many in the software world prize clever solutions to complex problems, yet often those clever solutions result from self-inflicted accidental complexity. Numerous examples of this phenomenon exist today, including the unfortunate but common practice of secreting orchestration or coordination code in an inappropriate location. For example, we see clever workflow management tools such as **Airflow** or **Prefect** that are overeagerly used to manage complex data pipelines through orchestration. We find a host of tools that work around the issues caused by monorepos, such as **Nx** and many more. Teams often don't realize they're doubling or tripling down on needless complexity without stepping back to look at the big picture and question whether the current solution is worse than the problem. Rather than jump to more technology to solve a problem, teams should do root cause analysis, address the underlying essential complexity and course correct. **Data mesh** is an example of an approach that addresses the underlying organizational and technical assumptions that have led to overly complex data pipelines and tooling.

Fewer technology platforms on the Radar

We found a serious drop in the number of platform-related blips in this edition of the Radar, which we attribute to the increased consolidation on some industry standards: most companies have already chosen their cloud vendors, and they've mostly standardized on **Kubernetes** for container orchestration and **Kafka** for high-performance messaging. Does this mean that platforms no longer matter? Or are we experiencing the equivalent of a business cycle of alternating periods of expansion and contraction — we've seen similar periods of rapid innovation followed by stasis (what Stephen Jay Gould called "punctuated equilibrium") in database technologies, for example. Perhaps the industry has entered a period of relative calm as organizations assimilate the tectonic shift to the cloud and await the next wave of disruptive innovation.

The Radar

Hold Assess Trial Adopt

Adopt Trial Assess Hold
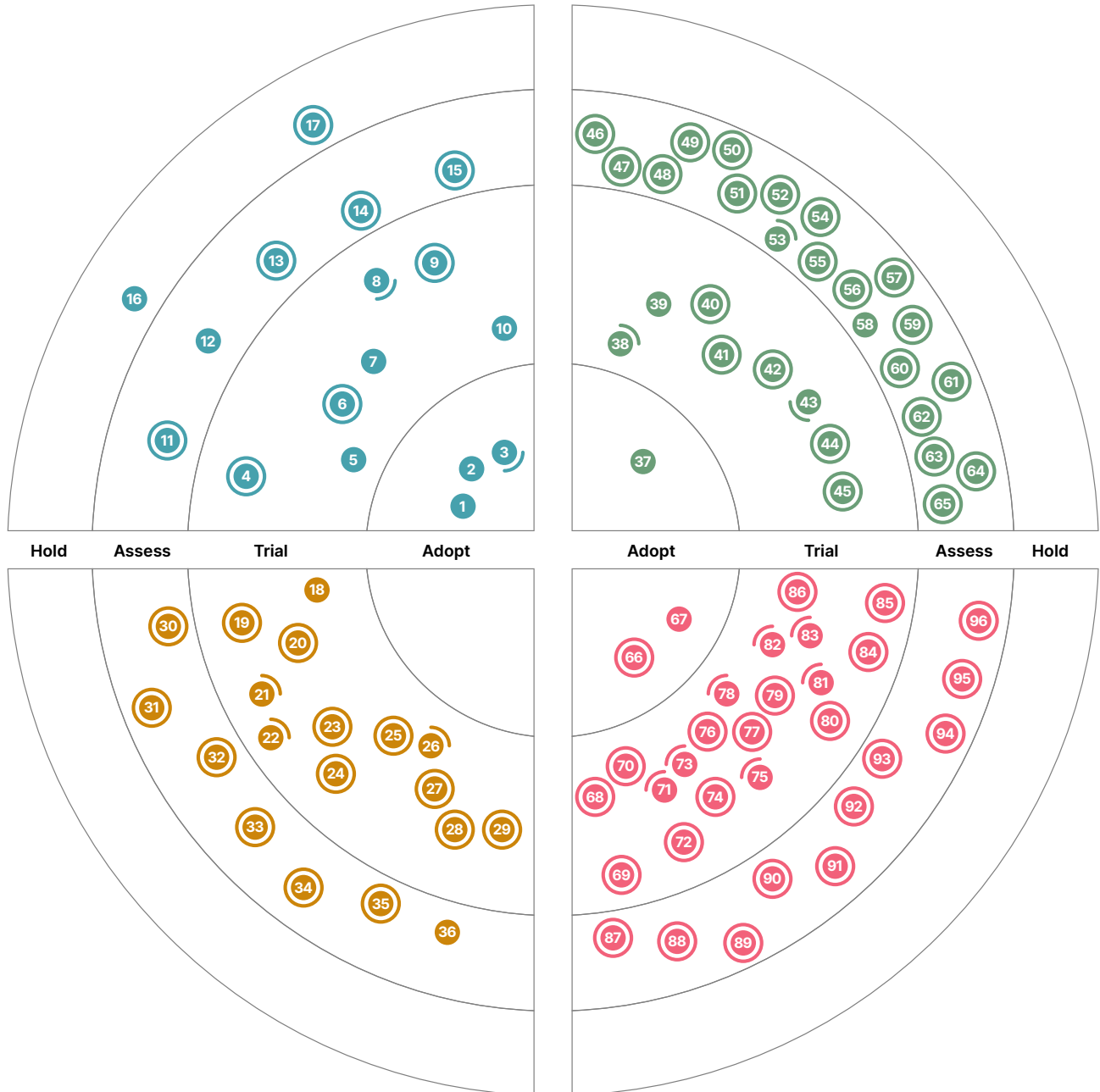
New   Moved in/out   No change

The Radar

Techniques

Adopt
1. Four key metrics
2. Platform engineering product teams
3. Zero trust architecture

Trial
4. CBOR/JSON bilingual protocols
5. Data mesh
6. Living documentation in legacy systems
7. Micro frontends for mobile
8. Remote mob programming
9. Single team remote wall
10. Team cognitive load

Assess
11. AR spatial anchors
12. Hotwire
13. Operator pattern for nonclustered resources
14. Remote spontaneous huddling
15. Software Bill of Materials

Hold
16. Peer review equals pull request
17. Production data in test environments

Platforms

Adopt
—

Trial
18. Backstage
19. ClickHouse
20. Confluent Kafka REST Proxy
21. GitHub Actions
22. K3s
23. Mambu
24. MirrorMaker 2.0
25. OPA Gatekeeper for Kubernetes
26. Pulumi
27. Sealed Secrets
28. Vercel
29. Weights & Biases

Assess
30. Azure Cognitive Search
31. Babashka
32. ExternalDNS
33. Konga
34. Milvus 2.0
35. Thought Machine Vault
36. XTDB

Hold
—

The Radar

Tools

Adopt
37. fastlane

Trial
38. Airflow
39. Batect
40. Berglas
41. Contrast Security
42. Dive
43. Lens
44. Nx
45. Wav2Vec 2.0

Assess
46. cert-manager
47. Cloud Carbon Footprint
48. Code With Me
49. Comby
50. Conftest
51. Cosign
52. Crossplane
53. gopass
54. Micoo
55. mob
56. Modern Unix commands
57. Mozilla Sops
58. Operator Framework
59. Pactflow
60. Prefect
61. Proxyman
62. Regula
63. Sourcegraph
64. Telepresence
65. Vite

Hold
—

Languages and Frameworks

Adopt
66. Jetpack Compose
67. React Hooks

Trial
68. Arium
69. Chakra UI
70. DoWhy
71. Gatsby.js
72. Jetpack Hilt
73. Kotlin Multiplatform Mobile
74. lifelines
75. Mock Service Worker
76. NgRx
77. pydantic
78. Quarkus
79. React Native Reanimated 2.0
80. React Query
81. Tailwind CSS
82. TensorFlow Lite
83. Three.js
84. ViewInspector
85. Vowpal Wabbit
86. Zap

Assess
87. Headless UI
88. InsightFace
89. Kats
90. ksqlDB
91. Polars
92. PyTorch Geometric
93. Qiankun
94. React Three Fiber
95. Tauri
96. Transloco

Hold
—

Techniques

Adopt
1. Four key metrics
2. Platform engineering product teams
3. Zero trust architecture

Trial
4. CBOR/JSON bilingual protocols
5. Data mesh
6. Living documentation in legacy systems
7. Micro frontends for mobile
8. Remote mob programming
9. Single team remote wall
10. Team cognitive load

Assess
11. AR spatial anchors
12. Hotwire
13. Operator pattern for nonclustered resources
14. Remote spontaneous huddling
15. Software Bill of Materials

Hold
16. Peer review equals pull request
17. Production data in test environments
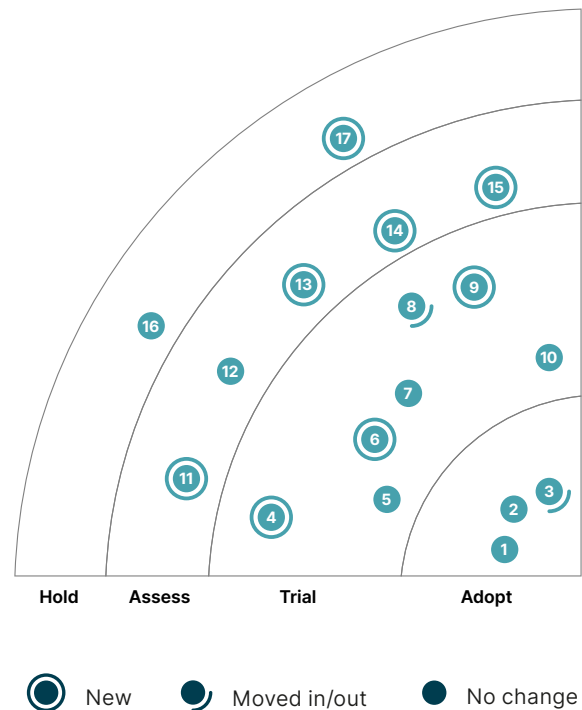
1. Four key metrics

Adopt

To measure software delivery performance, more and more organizations are turning to the four key metrics as defined by the **DORA research** program: change lead time, deployment frequency, mean time to restore (MTTR) and change fail percentage. This research and its statistical analysis have shown a clear link between high delivery performance and these metrics; they provide a great leading indicator for how a team, or even a whole delivery organization, is doing.

We're still big proponents of these metrics, but we've also learned some lessons since we first started monitoring them. And we're increasingly seeing misguided measurement approaches with tools that help teams measure these metrics based purely on their continuous delivery (CD) pipelines. In particular when it comes to the stability metrics (MTTR and change fail percentage), CD pipeline data alone doesn't provide enough information to determine what a deployment failure with real user impact is. Stability metrics only make sense if they include data about real incidents that degrade service for the users.

And as with all metrics, we recommend to always keep in mind the ultimate intention behind a measurement and use them to reflect and learn. For example, before spending weeks to build up sophisticated dashboard tooling, consider just regularly taking the **DORA quick check** in team retrospectives. This gives the team the opportunity to reflect on which **capabilities** they could work on to improve their metrics, which can be much more effective than overdetailed out-of-the-box tooling.

2. Platform engineering product teams

Adopt

We continue to see platform engineering product teams as a sensible default with the key insight being that they're just another **product team**, albeit one focused on internal platform customers. Thus it is critical to have clearly defined customers and products while using the same engineering disciplines and ways of working as any other (externally focused) product team; platform teams aren't special in this regard. We strongly caution against just renaming existing internal teams "platform teams" while leaving ways of working and organizational structures unchanged. 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.

3. Zero trust architecture

Adopt

We keep hearing about enterprises finding their security badly compromised due to an overreliance on the "secure" network perimeter. Once this external perimeter is breached, internal systems prove to be poorly protected with attackers quickly and easily able to deploy automated data extraction tools and ransomware attacks that all too often remain undetected for long periods. This leads us to recommend zero trust architecture (ZTA) as a now sensible default.

ZTA is a paradigm shift in security architecture and strategy. It's based on the assumption that a network perimeter is no longer representative of a secure boundary and no implicit trust should be granted to users or services based solely on their physical or network location. The number of

resources, tools and platforms available to implement aspects of ZTA keeps growing and includes enforcing **policies as code** based on the least privilege and as-granular-as-possible principles and continuous monitoring and automated mitigation of threats; using **service mesh** to enforce security control application-to-service and service-to-service; implementing **binary attestation** to verify the origin of the binaries; and including **secure enclaves** in addition to traditional encryption to enforce the three pillars of data security: in transit, at rest and in memory. For introductions to the topic, consult the **NIST ZTA** publication and Google's white paper on **BeyondProd**.

4. CBOR/JSON bilingual protocols
Trial

Although it's been around for a while, we're seeing more and more use cases where using the **CBOR** specification for data interchange makes sense — especially in environments containing multiple types of applications communicating with one another: service to service, browser to service, and so on. One thing we've found useful with **Borer**, a Scala implementation of a CBOR encoder/decoder, is the ability for clients to negotiate content between the binary representation and plain old JSON format. It's quite useful to have a text version viewable in a browser as well as the concise binary format. We foresee CBOR/JSON bilingual protocols picking up in popularity with the continuing rise of IoT and edge computing and other situations where the environment is tightly constrained.

5. Data mesh
Trial

Increasingly, we see a mismatch between what data-driven organizations want to achieve and what the current data architectures and organizational structures allow. Organizations want to embed data-driven decision-making, machine learning and analytics into many aspects of their products and services and how they operate internally; essentially they want to augment every aspect of their operational landscape with data-driven intelligence. Yet, we still have a ways to go before we can embed analytical data, access to it and how it is managed into the business domains and operations. Today, every aspect of managing analytical data is externalized outside of the operational business domains to the data team and to the data management monoliths: data lakes and data warehouses. **Data mesh** is a decentralized sociotechnical approach to remove the dichotomy of analytical data and business operation. Its objective is to embed sharing and using analytical data into each operational business domain and close the gap between the operational and analytical planes. It's founded on four principles: domain data ownership, data as a product, self-serve data platform and computational federated governance.

Our teams have been implementing the **data mesh architecture**; they've created new architectural abstractions such as the data product quantum to encapsulate the code, data and policy as an autonomous unit of analytical data sharing embedded into operational domains; and they've built self-serve data platform capabilities to manage the lifecycle of data product quanta in a declarative manner as described in *Data Mesh*. Despite our technical advances, we're still experiencing friction using the existing technologies in a data mesh topology, not to mention the resistance of business domains to embrace sharing and using data as a first-class responsibility in some organizations.

6. Living documentation in legacy systems

Trial

Living documentation, which comes from the behavior-driven development (BDD) community, is often considered a privilege for those well-maintained codebases with executable specifications. We found that this technique can also be applied to legacy systems. Lack of business knowledge is a common obstacle encountered by teams when doing system modernization. Code is usually the only trustworthy source of truth because staff turnover and existing documentation are outdated. Therefore it's very important to reestablish the association between the documentation and the code and spread the business knowledge among the team when we take over a legacy system. In practice, we would first try to go to the codebase and deepen our understanding of the business through simple cleanup and safe refactoring. During the process, we'll need to add annotations to the code so that we're able to automatically generate living documentation later. This is very different from doing BDD in green-field projects, but it's a good start in legacy systems. Based on the generated documentation, we would try to convert some of the specs into executable high-level automation tests. Do this iteratively, and eventually you could get living documentation in legacy systems that is closely associated with the code and partially executable.

7. Micro frontends for mobile

Trial

Since introducing them in the Radar in 2016, we've seen widespread adoption of **micro frontends** for web UIs. Recently, however, we've seen projects extend this architectural style to include micro frontends for mobile applications as well. When the application becomes sufficiently large and complex, it becomes necessary to distribute the development over multiple teams. This presents the challenge of maintaining team autonomy while integrating their work into a single app. Some teams write their own frameworks to enable this development style, and in the past we've mentioned **Atlas and Beehive** as possible ways to simplify the problem of integrating multiteam app development. More recently, we've seen teams using **React Native** to accomplish the same thing. Each React Native micro frontend is kept in its own repository where it can be built, tested and deployed separately. The team responsible for the overall application can then aggregate those micro frontends built by different teams into a single released app.

8. Remote mob programming

Trial

We continue to see many teams working and collaborating remotely; for these teams remote mob programming is a technique that is well worth trying. Remote mob programming allows 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 using their video conferencing tool of choice without having to connect to a big display, book a physical meeting room or find a whiteboard.

9. Single team remote wall

Trial

With the increased use of remote distributed teams, one of the things we hear people have missed having is the physical team wall. This is a single place where all the various story cards, tasks, status and progress can be displayed, acting as an information radiator and hub for the team. Often the

wall was an integration point with the actual data being stored in different systems. As teams have become remote, they've had to revert to looking into the individual source systems and getting an "at a glance" view of a project has become very difficult. A single team remote wall is a simple technique to reintroduce the team wall virtually. While there might be some overhead in keeping this up-to-date, we feel the benefits to the team are worth it. For some teams, updating the physical wall formed part of the daily "ceremonies" the team did together, and the same can be done with a remote wall.

10. 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.

11. AR spatial anchors
Assess

Many augmented reality (AR) applications depend on knowing the location and orientation of the user's device. The default is to use GPS-based solutions, but spatial anchors, a newer technique to address this requirement, are also worth considering. Spatial anchors work with the image recorded by the device's camera, using image features and their relative position in 3D space to recognize a real-world location. For this location a corresponding anchor is created in the AR space. Although spatial anchors can't replace all GPS and marker-based anchors, they do provide more accuracy than most GPS-based solutions and are more resilient to different viewing angles than marker-based anchors. Our experience is currently limited to Google's **Cloud Anchors for Android**, which worked well for us. Somewhat uncharacteristically Google also offers **Cloud Anchors for iOS** and with **Azure Spatial Anchors** Microsoft supports even more platforms.

12. Hotwire
Assess

After successfully launching their email application **HEY** as a server-side application, Basecamp **reported** migrating its flagship product, **Basecamp 3**, to **Hotwire** this summer. As organizations increasingly default to single-page applications (SPAs) for new web development, we continue to be excited by Hotwire swimming against the stream. Unlike SPAs, Hotwire applications keep most of the logic and navigation on the server, relying on a minimal amount of browser JavaScript. Hotwire modularizes HTML pages into a set of components (called **Turbo Frames**) that can be lazy loaded, provide independent contexts and send HTML updates to those contexts based on user actions. SPAs offer undeniable user responsiveness, but the simplicity of traditional server-side web programming combined with modern browser tooling provides a refreshing take on balancing developer effectiveness and user responsiveness.

13. Operator pattern for nonclustered resources
Assess

We're seeing increasing use of the **Kubernetes Operator** pattern for purposes other than managing applications deployed on the cluster. Using the operator pattern for nonclustered resources takes advantage of custom resource definitions and the event-driven scheduling mechanism implemented in the Kubernetes control plane to manage activities that are related to yet outside of the cluster. This technique builds on the idea of **Kube-managed cloud services** and extends it to other activities, such as continuous deployment or reacting to changes in external repositories. One advantage of this technique over a purpose-built tool is that it opens up a wide range of tools that either come with Kubernetes or are part of the wider ecosystem. You can use commands such as `diff`, `dry-run` or `apply` to interact with the operator's custom resources. Kube's scheduling mechanism makes development easier by eliminating the need to orchestrate activities in the proper order. Open-source tools such as **Crossplane**, **Flux** and **ArgoCD** take advantage of this technique and we expect to see more of these emerge over time.

14. Remote spontaneous huddling
Assess

We're seeing continued innovation in remote collaboration tools. The new **Huddles** feature in Slack provides a Discord-like experience of persistent audio calls that users can jump in and out of at any time. **Gather** provides a creative way to emulate a virtual office with avatars and video. IDEs provide direct collaboration features for pairing and debugging: we've previously blipped **Visual Studio Live Share** and included **JetBrains Code With Me** to the list in this edition. As tools continue to evolve modalities for collaboration in addition to video conferencing, we're increasingly seeing teams participating in remote spontaneous huddling, recreating the spontaneity of informal conversations over the intentionality of scheduling a Zoom or Microsoft Teams meeting. We don't expect to ever fully recreate the richness of face-to-face communication through digital tools, but we do see improved remote team effectiveness by giving teams multiple channels of collaboration rather than relying on one toolchain for everything.

15. Software Bill of Materials
Assess

In May 2021, the U.S. White House published its **Executive Order on Improving the Nation's Cybersecurity**. The document puts forward several technical mandates that relate to items we've featured in past Radars, such as **zero trust architecture** and automated compliance scanning using **security policy as code**. Much of the document is devoted to improving the security of the software supply chain. One item in particular that caught our attention was the requirement that government software should contain a machine-readable Software Bill of Materials (SBOM), defined as "a formal record containing the details and supply chain relationships of various components used in building software." In other words, it should detail not just the components shipped but also the tools and frameworks used to deliver the software. This order has the potential to usher in a new era of transparency and openness in software development. This will undoubtedly have an impact on those of us who produce software for a living. Many, if not all software products produced today contain open-source components or employ them in the build process. Often, the consumer has no way of knowing which version of which package might have an impact on the security of their

product. Instead they must rely on the security alerts and patches provided by the retail vendor. This executive order will ensure that an explicit description of all components is made available to consumers, empowering them to implement their own security controls. And since the SBOM is machine-readable, those controls can be automated. We sense that this move also represents a shift toward embracing open-source software and practically addressing both the security risks and benefits that it provides.

16. 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.

17. Production data in test environments
Hold

We continue to perceive production data in test environments as an area for concern. Firstly, many examples of this have resulted in reputational damage, for example, where an incorrect alert has been sent from a test system to an entire client population. Secondly, the level of security, specifically around protection of private data, tends to be less for test systems. There is little point in having elaborate controls around access to production data if that data is copied to a test database that can be accessed by every developer and QA. Although you can obfuscate the data, this tends to be applied only to specific fields, for example, credit card numbers. Finally, copying production data to test systems can break privacy laws, for example, where test systems are hosted or accessed from a different country or region. This last scenario is especially problematic with complex cloud deployments. Fake data is a safer approach, and tools exist to help in its creation. We do recognize there are reasons for specific elements of production data to be copied, for example, in the reproduction of bugs or for training of specific ML models. Here our advice is to proceed with caution.

Platforms

Adopt
—

Trial
18. Backstage
19. ClickHouse
20. Confluent Kafka REST Proxy
21. GitHub Actions
22. K3s
23. Mambu
24. MirrorMaker 2.0
25. OPA Gatekeeper for Kubernetes
26. Pulumi
27. Sealed Secrets
28. Vercel
29. Weights & Biases

Assess
30. Azure Cognitive Search
31. Babashka
32. ExternalDNS
33. Konga
34. Milvus 2.0
35. Thought Machine Vault
36. XTDB

Hold
—

Hold	Assess	Trial	Adopt
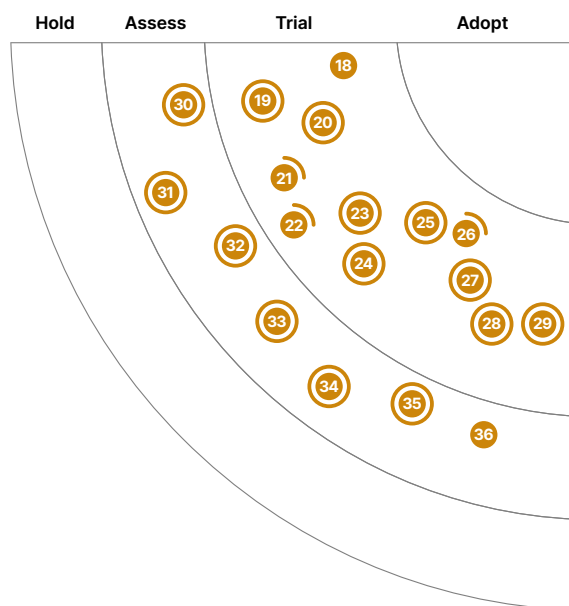
New | Moved in/out | No change

18. Backstage
Trial

As the focus on improving the developer experience and efficiency increases across organizations, we're seeing **Backstage** rise in popularity, alongside the adoption of developer portals. These organizations are looking 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 can 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 on 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. We'll be watching the new **Backstage Service Catalog**, currently in alpha, which keeps track of ownership and metadata for all the software in an organization's ecosystem.

19. ClickHouse
Trial

ClickHouse is an open-source, columnar online analytical processing (OLAP) database for real-time analytics. It started as an experimental project in 2009 and since has matured into a highly performant and linearly scalable analytical database. It's efficient query processing engine together with data compression makes it suitable to run interactive queries without pre-aggregation. We've used ClickHouse and are quite impressed with its performance.

20. Confluent Kafka REST Proxy
Trial

Kafka is a common default for event-driven architectures, but adapting it to legacy environments introduces an impedance mismatch. In a few cases, we've had success minimizing the legacy complexity using **Confluent Kafka REST Proxy**. The proxy allows developers to access Kafka through an HTTP interface, which is useful in environments that make using the native Kafka protocol difficult. For example, we were able to consume events emitted through SAP simply by having the SAP team invoke an HTTP POST command through a preconfigured SAP remote function call, avoiding the need to spin up a Java abstraction around SAP (and a team to manage it). The proxy is quite full-featured, although, as with any such adapter tool, we recommend caution and a clear-eyed view of the trade-offs involved. We believe the proxy is valuable when it enables legacy producers to send events, but would be careful creating event consumers through the proxy as the abstraction gets more complex. The proxy doesn't change the fact that Kafka consumers are stateful, which means that consumer instances created through the REST API are tied to a specific proxy, and the need to make an HTTP call to consume messages from a topic changes the standard semantics of Kafka eventing.

21. GitHub Actions
Trial

Despite our cautionary advice when we last blipped it, we've seen continued enthusiasm for **GitHub Actions**. What we said before still holds true: GitHub Actions is not yet a full-fledged CI/CD replacement for complex workflows. It cannot, for example, re-trigger a single job of a workflow, call other actions inside a composite action or support a shared library. Furthermore, while the ecosystem in the **GitHub Marketplace** offers obvious advantages, giving third-party GitHub Actions access to your build pipeline risks sharing secrets in insecure ways (we recommend following GitHub's advice

on **security hardening**). Despite those concerns, the convenience of creating your build workflow directly in GitHub next to your source code is a compelling option for some teams, and **act** helps you run GitHub Actions locally. As always, we recommend a clear-eyed assessment of the trade-offs, but some of our teams are happy with the simplicity of GitHub Actions.

22. K3s
Trial
K3s is a lightweight Kubernetes distribution built for IoT and edge computing. You get the benefits of a fully compliant Kubernetes but with reduced operational overhead. Its enhancements include lightweight storage backends (**sqlite3** as default instead of **etcd**), a single binary package with minimal OS dependencies and reduced memory footprint, all of which make K3s suitable for resource-constrained environments. We've used K3s in point-of-sale machines, and we're quite happy with our decision.

23. Mambu
Trial
Mambu is a SaaS cloud banking platform. It empowers customers to easily and flexibly build and change their banking and lending products. Unlike other out-of-box core banking platforms that you can only adapt with hard-coded integration, Mambu is designed for constantly changing financial offerings. It comes with an opinionated workflow, while also providing an API-driven approach to customize business logic, process and integrations. We currently have several projects using Mambu. With its cloud-based scalability and highly customizable capabilities, it's becoming one of the sensible default domain systems when building financial products.

24. MirrorMaker 2.0
Trial
MirrorMaker 2.0 (also known as MM2), built using the Kafka Connect framework, solves many tool shortcomings of previous Kafka replication approaches. It can successfully **geo-replicate** topic data and metadata across clusters, including offsets, consumer groups and authorization command lines (ACLs). MM2 preserves partitioning and detects new topics and partitions. We appreciated the ability to stage a cluster migration over time, an approach that can be useful in migrating from an on-prem cluster to a cloud cluster. After synchronizing the topics and consumer groups, we first migrated the clients to the new cluster location, then we migrated the producers to the new location and finally turned off MM2 and decommissioned the old cluster. We've also seen MM2 used in disaster recovery and high-availability scenarios.

25. OPA Gatekeeper for Kubernetes
Trial
OPA Gatekeeper for Kubernetes is a customizable admission webhook for **Kubernetes** that enforces policies executed by the **Open Policy Agent (OPA)**. We're using this extension of the Kubernetes platform to add a security layer to clusters, providing automated governance mechanisms that ensure applications are compliant with defined policies. Our teams like it because of its customization capability; using CustomResourceDefinitions (CRD) allows us to define ConstraintTemplates and Constraints which make defining rules and the objects (e.g., deployments, jobs, cron jobs) and namespaces under evaluation an easy task.

26. Pulumi
Trial

We've been seeing an increase in teams using **Pulumi** in various organizations. 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.

27. Sealed Secrets
Trial

Kubernetes natively supports a key-value object known as a secret. However, by default, Kubernetes secrets aren't really secret. They're handled separately from other key-value data so that precautions or access control can be applied separately. There is support for encrypting secrets before they are stored in **etcd**, but the secrets start out as plain text fields in configuration files. **Sealed Secrets** is a combination operator and command-line utility that uses asymmetric keys to encrypt secrets so that they can only be decrypted by the controller in the cluster. This process ensures that the secrets won't be compromised while they sit in the configuration files that define a Kubernetes deployment. Once encrypted, these files can be safely shared or stored alongside other deployment artifacts.

28. Vercel
Trial

Since we first evaluated **JAMstack**, we've seen more and more web applications of this style. However, when the infrastructure for building traditional dynamic websites and back-end services is too heavy for JAMstack, our teams choose **Vercel**. Vercel is a cloud platform for static site hosting. More importantly, it provides a seamless workflow for developing, previewing and shipping JAMstack sites. The configuration for the deployment is quite simple. By integrating with GitHub, each code commit or pull request could trigger a new website deployment that has a URL for preview, which greatly accelerates development feedback. Vercel also uses CDN to scale and speed up production sites. It's worth mentioning that the team behind Vercel also supports another popular framework, **Next.js**.

29. Weights & Biases
Trial

Weights & Biases is a machine learning (ML) platform for building models faster through experiment tracking, data set versioning, visualizing model performance and model management. You can integrate it with existing ML code and quickly get live metrics, terminal logs and system statistics streamed to the dashboard for further analysis. Our teams have used Weights & Biases, and we like its collaborative approach to model building.

30. Azure Cognitive Search
Assess

Azure Cognitive Search provides search as a service for applications that require text search over heterogeneous content. It provides push or pull-based APIs to upload and index images, unstructured text or structured document content, with **limitations on supported pull-based data source types**.

It provides APIs over REST and .NET SDK to execute search queries, either using a simple query language or more powerful **Apache Lucene** queries with field-scoped queries, fuzzy search, infix and suffix wildcard search and regular expression search, among other features. We've successfully used Azure Cognitive Search alongside other Azure services, including searching content uploaded from **Cosmos DB**.

31. Babashka
Assess

Even today, considering all the development and infrastructure tools at our disposal, we often reach a point where we need a script to glue several things together or to automate a recurring task. Current favorites for writing these scripts are bash and Python, but we're happy to report that there's a new, exciting option: Clojure. This is made possible with **Babashka**, a complete Clojure run time implemented with **GraalVM**. Babashka ships with libraries that cover most of the use cases for which you'd use a scripting tool, and loading of further libraries is possible, too. The use of GraalVM brings startup times within range of native tools, and it also makes Babashka one of the few options for a multithreaded scripting environment, for those rare cases when it's needed.

32. ExternalDNS
Assess

ExternalDNS synchronizes Kubernetes ingresses and services with external DNS providers, filling a hole previously filled by **kops dns-controller**, **Zalando's Mate** or **route53-kubernetes** — the last two of which have been deprecated in favor of ExternalDNS. The tool makes internal Kubernetes resources discoverable via public DNS servers, removing a sometimes manual step to update DNS records when an ingress host or service's IP address changes. It supports a huge list of DNS service providers out of the box with more being added via community support. As the old joke goes, **it's always DNS**.

33. Konga
Assess

Konga is an open-source UI for administering the **Kong API Gateway**, previously featured in the Radar in Trial. Our teams liked the quick setup and rich feature set that allowed them to experiment with and try out configurations easily. And the fact it's open-source software eases concerns about licensing costs.

34. Milvus 2.0
Assess

Milvus 2.0 is a cloud-native, open-source vector database built to search and manage embedding vectors generated by machine-learning models and neural networks. It supports several **vector indexes** for approximate nearest neighbors (ANN) search across embedding vectors of audio, video, image or any unstructured data. Milvus 2.0 is a relatively new database, and we recommend you assess it for your similarity search needs.

35. Thought Machine Vault
Assess

It's rare for us to feature commercial, off-the-shelf software in the Radar, much less a core banking platform. However, **Thought Machine Vault** (no connection to Thoughtworks) is an example of a product in this class designed to support good software engineering practices such as test-driven

development, continuous delivery and infrastructure as code. Developers define banking products in Vault by writing smart contracts in Python code. This is distinctly different from the standard no-code approach where customization is done through graphical interfaces or proprietary configuration files or both. Because products are defined in regular Python code, developers have access to a range of tools such as test frameworks and version control to ensure that their work is safe and accurate. We wish more financial services platforms were designed with developer effectiveness in mind.

36. XTDB
Assess

XTDB is an open-source document database with bitemporal graph queries. It natively supports two time axes for each record: valid time, when a fact occurs, and transaction time, when a fact is processed and recorded by the database. Support for bitemporality is beneficial in numerous scenarios, including analytical use cases executing time-aware queries; auditing historical changes to facts; supporting distributed data architectures that must guarantee globally consistent point-in-time queries such as **data mesh**; and preserving data immutability. XTDB takes information in document form, expressed in the Extensible Data Notation (EDN) format, a subset of the Clojure language. XTDB supports graph as well as SQL queries and is extensible through a REST API layer and Kafka Connect, among other modules. We're excited to see a growth in adoption of XTDB and the addition of features such as support for transactions and SQL.

Tools

Adopt
37. fastlane

Trial
38. Airflow
39. Batect
40. Berglas
41. Contrast Security
42. Dive
43. Lens
44. Nx
45. Wav2Vec 2.0

Assess
46. cert-manager
47. Cloud Carbon Footprint
48. Code With Me
49. Comby
50. Conftest
51. Cosign
52. Crossplane
53. gopass
54. Micoo
55. mob
56. Modern Unix commands
57. Mozilla Sops
58. Operator Framework
59. Pactflow
60. Prefect
61. Proxyman
62. Regula
63. Sourcegraph
64. Telepresence
65. Vite

Hold
—

Adopt   Trial   Assess   Hold
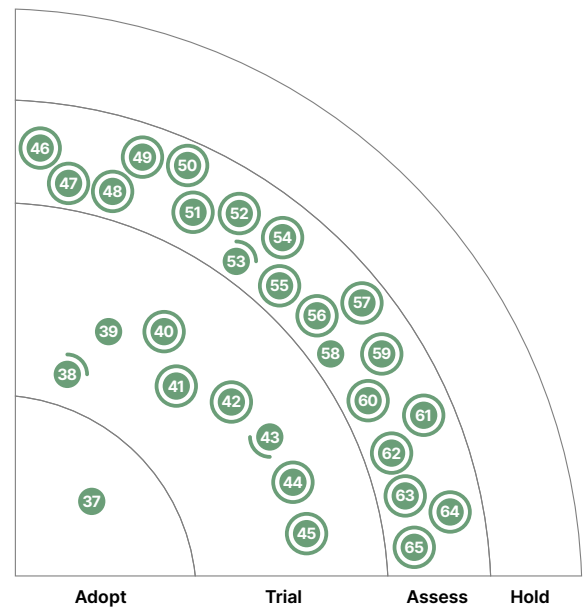
New   Moved in/out   No change

37. fastlane

Adopt

Releasing applications for iOS involves a code-signing step. Although supported by Apple's toolchain, the process can be cumbersome, error prone and full of unexpected surprises. We're happy to report that **fastlane**, already our tool of choice for automating the release process of mobile applications, provides a better solution: **match** is integrated into fastlane's smooth process, and it implements a **new approach** to manage code signing for teams. Instead of storing the signing keys in the developer's macOS keychain — the default approach — the new approach revolves around storing the keys and certificates in a Git repository. This not only makes it easier to on-board new team members and set up new development machines; in our experience, it also is the easiest method to integrate signing into continuous delivery pipelines.

38. Airflow

Trial

In recent years we've seen the rise of generic and domain-specific workflow management tools. The drivers behind this rise include the increased usage of data-processing pipelines and the automation of the machine-learning (ML) model development process. **Airflow** is one of the early open-source task orchestration tools that popularized the definition of directed acyclic graphs (DAGs) as code, an improvement over an XML/YAML pipeline configuration. Although Airflow remains one of the most widely adopted orchestration tools, we encourage you to evaluate other tools based on your unique situation. For example, you may want to choose **Prefect**, which supports dynamic data-processing tasks as a first-class concern with generic Python functions as tasks; or **Argo** if you prefer a tight integration with Kubernetes; or **Kubeflow** or **MLflow** for ML-specific workflows. Given the rise of new tools, combined with some of the shortfalls of Airflow (such as lack of native support for dynamic workflows and its centralized approach to scheduling pipelines), we no longer recommend Airflow as the default orchestration tool.

We believe that with the increased usage of streaming in analytics and data pipelines, as well as managing data through a **decentralized data mesh**, the need for orchestration tools to define and manage complex data-processing pipelines is reduced.

39. Batect

Trial

Batect continues to gain traction among our developers and is considered by many to be a default approach for configuring local development and test environments. This open-source tool (which happens to be developed by a Thoughtworker) makes it easy to set up and share a build environment based on **Docker**. Batect then becomes the entry point for your build system, replacing the ubiquitous go script as the basis for a "**check out and go**" approach. Batect continues to evolve in response to developer feedback and recently added support for Docker's BuildKit and shell tab completion.

40. Berglas

Trial

Berglas is a tool for managing secrets on **Google Cloud Platform (GCP)**. We've recommended **secrets as a service** as a technique to store and share secrets in modern distributed architectures in the past, and GCP offers **Secret Manager** for that purpose, and Berglas works well with Secret Manager. This is especially useful for those GCP services that don't have direct integration with

Secret Manager yet; the alternative in such cases would be to write custom code or scripts. Berglas ships as a command-line tool and as a library, and both also come in handy in use cases beyond secrets as a service. The author of Berglas, who also happens to be the original author of **HashiCorp Vault**, now works at Google; however, Berglass is not an official Google product.

41. Contrast Security

Trial

Contrast Security offers a security platform with multiple components, including static application security testing (SAST), interactive application security testing (IAST), open-source scanning and runtime application self-protection (RASP). It's been around for a few years now, and we've used it in multiple projects. One of the things we quite like about the Contrast platform is its run-time analysis of libraries; it helps identify libraries that are not used, which in turn helps our teams prioritize vulnerabilities and potentially get rid of unused libraries. This is particularly relevant given the increased importance of **securing the software supply chain**. We also quite like its IAST component; we've found it effective in our continuous delivery (CD) pipeline with reduced false positives, and it manages to catch a good range of vulnerabilities.

42. Dive

Trial

Dive is a tool for analyzing Docker images; it helps explore each layer in the image and identify what's changed in each layer. Dive estimates *image efficiency* and *wasted space* in an image and can be integrated into the continuous integration (CI) pipeline to fail the build based on the efficiency score or amount of wasted space. We've used it in a few projects, and it has proven to be a useful tool — particularly if you're building images with a very low tolerance for additional tools or space consumption.

43. Lens

Trial

Our teams continue to report good results when using **Lens** to visualize and manage their **Kubernetes** clusters. Billed as an "IDE for Kubernetes," Lens makes it possible to interact with the cluster without having to memorize commands or manifest file structures. Kubernetes can be complex, and we understand that a tool for visualizing cluster metrics and deployed workloads can save time and reduce some of the toil involved in maintaining a Kubernetes cluster. Instead of hiding complexity behind a simple point-and-click interface, Lens brings together the tools an administrator would run from the command line. But be cautious about interactively making changes to a running cluster via any mechanism. We generally prefer that infrastructure changes be **implemented in code** so they are repeatable, testable and less prone to human error. However, Lens does excel as a one-stop tool to interactively navigate through and comprehend your cluster status.

44. Nx

Trial

Over the years we've debated several times whether to feature monorepos in the Radar. Each time we ended up concluding that the trade-offs introduced by monorepos require a nuanced discussion and the technique is "too complex to blip." Now we're seeing increased interest in monorepos in the JavaScript community, for example, for building applications composed of micro frontends, as discussed in this **podcast episode**. Whether this is a good idea depends a lot on your situation, and

we certainly don't want to give a general recommendation. What we do want to comment on is the tooling. In our teams we see a shift away from **Lerna** and a strong preference to use **Nx** for managing JavaScript-based monorepos.

45. Wav2Vec 2.0
Trial

Wav2Vec 2.0 is a self-supervised learning framework for speech recognition. With this framework the model is trained in two phases. First, it begins in self-supervised mode using unlabeled data and tries to achieve the best possible speech representation. Then it uses supervised fine-tuning, during which labeled data teaches the model to predict particular words or phonemes. We've used Wav2Vec and find its approach quite powerful for building automatic speech recognition models for regional languages with limited availability of labeled data.

46. cert-manager
Assess

cert-manager is a tool to manage your X.509 certificates within your **Kubernetes** cluster. It models certificates and issuers as first-class resource types and provides certificates as a service securely to developers and applications working within the Kubernetes cluster. With built-in support for **Let's Encrypt**, **HashiCorp Vault** and Venafi, cert-manager is an interesting tool to assess for certificate management.

47. Cloud Carbon Footprint
Assess

Stakeholders increasingly expect businesses to account for the environmental externalities of their decisions, as evidenced by the rise of environmental, social and corporate governance (ESG) investing and employee activism around climate change. Migrating to the cloud offers the potential for more efficient energy usage — the cloud providers have much more scale to justify investment in green energy sources and R&D — but the downside of software abstractions for cloud users is that those abstractions also hide the energy impact as the actual data centers are hidden from view and financed by another company. **Cloud Carbon Footprint**, a new open-source tool, takes advantage of cloud APIs to provide visualizations of estimated carbon emissions based on usage across AWS, GCP and Azure. It uses heuristics like Etsy's **Cloud Jewels** to estimate energy usage and public data sources to convert energy usage into emissions based on the carbon intensity of the cloud region's underlying energy grid (GCP **publishes** this data already). The tool's dashboards act as information radiators, allowing decision makers to modify setups to cut costs and emissions at the same time. The linkage of cloud regions to carbon intensity of the underlying grid provides a nudge to switch dirty workloads to regions with greener energy sources.

48. Code With Me
Assess

JetBrains' collaborative coding tool, **Code With Me**, has been increasing in popularity as many teams use various JetBrains tools in this remote-first world. Along with other remote collaboration tools such as VSCode's **Visual Studio Live Share**, Code With Me gives development teams an improved experience with remote pairing and collaboration. Code With Me's abilities to invite teammates into the IDE projects and collaborate in real time are worth exploring. However, we've seen some limitations with regard to refactoring seamlessly and some issues in high-latency environments. We'll continue to watch this tool in this space.

49. Comby

Assess

This edition of the Radar introduces two tools that search and replace code using an abstract syntax tree (AST) representation. They occupy a similar space as **jscodeshift** but contain parsers for a wide range of programming languages. Although they share some similarities, they also differ in several ways. One of these tools, **Comby**, is unique in its simple, command-line interface designed in the spirit of Unix tools such as **awk** and **sed**. While the Unix commands are based on regular expressions operating matching text, Comby employs a pattern syntax that is specific to programming language constructs and parses the code before searching. This helps developers search large code bases for structural patterns. Like **sed**, Comby can replace the patterns it matches with new structures. This is useful for automating wholesale changes to large codebases or for making repetitive changes across a suite of microservice repositories. Since these tools are fairly new, we expect to see a range of creative uses that have yet to be discovered.

50. Conftest

Assess

Conftest is a tool for writing tests against structured configuration data. It relies on the **Rego language** from **Open Policy Agent** to write tests for **Kubernetes** configurations, **Tekton** pipeline definitions or even **Terraform** plans. Configurations are a critical part of the infrastructure, and we encourage you to assess Conftest to verify assumptions and get quick feedback.

51. Cosign

Assess

Cosign is a container signing and verification tool. Part of Sigstore — a project under the Cloud Native Computing Foundation (CNCF) umbrella aimed at simplifying software signing and transparency — Cosign supports not only Docker and Open Container Initiative (OCI) images but also other artifacts that can be stored in a container registry. We previously talked about **Docker Notary**, which also operates in this space; Notary v1, however, has some disadvantages: it's not registry native and needs a separate Notary server. Cosign avoids this problem and stores the signatures in the registry next to an image. It currently has integrations with **GitHub actions** and **Kubernetes** using a Webhook with further integrations in the pipeline. We've used Cosign in some of our projects and it looks quite promising.

52. Crossplane

Assess

Crossplane is another entry in the class of tools implemented by the **Kubernetes Operator pattern** but with side effects that extend beyond the Kubernetes cluster. In our last Radar we mentioned **Kube-managed cloud services** as a technique, and Crossplane does just that. The idea is to leverage the Kubernetes control plane to provision cloud services on which your deployment is dependent, even if they aren't deployed on the cluster itself. Examples include managed database instances, load balancers or access control policies. This tool is noteworthy for two reasons. First, it demonstrates the powerful and flexible execution environment of the underlying Kubernetes control plane. There is no real limit to the range of supported custom resources. Second, Crossplane provides an alternative to the usual options of **Terraform**, **CDK** or **Pulumi**. Crossplane comes with a set of predefined providers for the major cloud services that cover the most commonly provisioned services. It isn't trying to be a general-purpose infrastructure-as-code (IaC) tool but rather a companion to workloads

being deployed in Kubernetes. Often associated with the practice of **GitOps**, Crossplane stands on its own and allows you to stay within the Kubernetes ecosystem when it's necessary to manage external cloud resources. However, Crossplane doesn't help with provisioning Kubernetes itself; you'll need at least one other IaC tool to bootstrap the cluster.

53. gopass
Assess

gopass is a password manager for teams, built on GPG and Git. It's a descendant of **pass** and adds several features, including interactive search and multiple password stores in a single tree. Since we first mentioned gopass, our teams have used it on several projects, sometimes stretching it beyond its limits. A sorely missed feature was the ability to deprecate secrets. Discoverability was already an issue, but not being able to mark secrets as no longer in use compounded this problem. The biggest issue, though, was scale. When you have teams with 50+ people using the same repository for several years, we found that the repository could grow to multiple gigabytes in size. Re-encrypting the secrets when onboarding new members could take more than half an hour. The underlying issue seems to be that in our teams everything changes all the time: people come and go, secrets are rotated, the architecture evolves, new secrets are added, old ones are no longer needed. gopass seems to work well, even for large numbers of users, when there's less change.

54. Micoo
Assess

Micoo is a new entrant into the crowded space of **visual regression tools**; it's an open-source solution and is self-contained, providing Docker images to enable an easy and quick environment setup. It also provides different clients for Node.js, Java and Python as well as a Cypress plugin so it can be easily integrated with most of the common frontend UI automation testing frameworks or solutions. Although Micoo doesn't provide all the functionality of some of the SaaS-based or other commercial solutions, our teams have been using it extensively and have had positive experiences. They've especially called out that it works for mobile and desktop apps as well as the web.

55. mob
Assess

Sometimes you come across a tool that you didn't realize you needed until you do; **mob** is just such a tool. Living as we do in a world where remote pair programming has become the norm for many teams, having a tool that allows for seamless handover either between pairs or a wider group as part of a **mob programming** session is super useful. mob hides all the version control paraphernalia behind a command-line interface that makes participating in mob programming sessions simpler. It also provides specific advice around how to participate remotely, for example, to "steal the screenshare" in Zoom rather than ending a screenshare, ensuring the video layout doesn't change for participants. A useful tool and thoughtful advice, what's not to like?

56. Modern Unix commands
Assess

There are many reasons to love Unix, but the one that has profoundly affected our industry is the Unix philosophy of building applications that "do one thing and do it well." Unix commands embody this philosophy. A set of small functions that can be piped together to create more complex solutions. In recent years, programmers have contributed to a growing set of modern Unix commands. These

modern versions attempt to be smaller and faster, often written in **Rust**. They include additional features such as syntax highlighting and utilize features of modern terminals. They aim to support programmers natively by integrating nicely with `git` and recognizing source code files. For example, **bat** is a replacement for `cat` with paging and syntax highlighting; **exa** is a replacement for `ls` with extended file information and **ripgrep** is a faster `grep` replacement that by default ignores gitignore, binary and hidden files. The **Modern Unix** repository has a reference to some of these commands. We've been enjoying using these Unix commands. You should try them in improving your command-line experience. However, we caution against using them in scripts as replacements for the standard command-line utilities that are shipped in default OS distributions, because they reduce the scripts' portability running on other machines.

57. Mozilla Sops
Assess

Plaintext secrets checked into source control (usually Github) are one of the most pervasive security mistakes developers make. For this reason we thought it useful to feature **Mozilla Sops**, a tool for encrypting secrets in text files that our developers find useful in situations where it is impossible to remove secrets from legacy code repositories. We've mentioned many tools of this type before (**Blackbox**, **git-crypt**), but Sops has several features that set it apart. For example, Sops integrates with cloud-managed keystores such as AWS and GCP Key Management Service (KMS) or Azure Key Vault as sources of encryption keys. It also works cross-platform and supports PGP keys. This enables fine-grained access control to secrets on a file-by-file basis. Sops leaves the identifying key in plain text so that secrets can still be located and diffed by git. We're always supportive of anything that makes it easier for developers to be secure; however, remember that you don't have to keep secrets in source control to begin with. See **Decoupling secret management from source code** in our November 2017 issue.

58. Operator Framework
Assess

We continue to see the adoption of Kubernetes in new and novel scenarios. For example, we see Kubernetes is being extended to **manage resources running outside of its cluster** or **across multiple infrastructure providers**, or it is used in managing stateful applications beyond Kubernetes's original scope. These extensions are possible using the **Kubernetes Operator** pattern: building Kubernetes controllers that have the domain-specific knowledge of the custom resource they manage. For example, an operator that manages a stateful application can use the Kubernetes primitives to automate an application's specific tasks beyond its deployment, such as restore, backup and upgrade its database.

Operator Framework is a set of open-source tools that simplifies building and managing the lifecycle of **Kubernetes operators**. Although there are **multiple frameworks** to help you build Kubernetes operators, Operator Framework remains a good choice. It supports rich operator lifecycle management using its **Operator Lifecycle Manager** module; it supports multiple languages to build the operator code itself using its **Operator SDK**; and it provides a **catalog** for publishing and sharing the operators. If you're planning to build Kubernetes operators, we recommend giving the Operator Framework a try to accelerate your development reliably.

59. Pactflow

Assess

For organizations with larger and more complex API ecosystems, especially those who are already using **Pact**, we think it's worth assessing whether **Pactflow** could be useful. Pactflow manages the workflow and continuous deployment of tests written in Pact, lowering the barrier to **consumer-driven contract testing**. The complexity of coordination between multiple producers and various disparate consumers can become prohibitive. We've seen some teams invest significant effort in hand-crafting solutions to this problem and think it's worth assessing whether Pactflow can look after this for you.

60. Prefect

Assess

Prefect is a data workflow management tool that makes it easy to add semantics such as retries, dynamic mapping, caching and failure notifications to data pipelines. You can mark Python functions as tasks and chain them together through function calls to build the data flow. The Python API combined with a collection of predefined tasks for common data operations makes Prefect a noteworthy option to assess for your data pipeline needs.

61. Proxyman

Assess

It may not be a tool that you need everyday, but when you're in the weeds trying to diagnose a nasty network problem, it's very useful to be able to reach for a feature-rich HTTP debugging proxy. **Proxyman** is just such a tool. Quite a few of our teams have been using it for a while now as a macOS-specific drop-in replacement for **Charles** and really like its streamlined interface and cert management.

62. Regula

Assess

One of the key tenets of infrastructure as code (IaC) is automated testing. If we have a solid test pyramid with good code-level coverage at the bottom, we can produce a better and more secure infrastructure. Unfortunately, tools to assist in this space have been sparse. **Conftest** is frequently used to test Terraform JSON and HCL code, but it is a general-purpose tool. **Regula** is an attractive alternative. Similar to Conftest, Regula checks for compliance of infrastructure code by applying rules written in Open Policy Agent's Rego language, but it also provides a set of primitives specifically for validating infrastructure configurations. Because both tools are based on the Rego language, Regula rules can be run by Conftest. However, Regula comes with its own command-line tool for running tests as part of a pipeline with no dependence on Conftest or OPA. Our developers have found that Regula saves time and produces much more readable, maintainable and succinct test code. Still, both tools only validate the infrastructure code. A complete suite should also test the infrastructure to ensure the code is being accurately interpreted.

63. Sourcegraph

Assess

Another abstract syntax tree–based code search tool that received our attention is **Sourcegraph**. In contrast to **Comby**, which is open source, Sourcegraph is a commercial tool (with a 10-user free tier). Sourcegraph is particularly suited for searching, navigating or cross-referencing in large codebases.

The cloud-hosted version can be accessed through Sourcegraph's website and is designed to search publicly available open-source repositories. Whereas Comby is a lightweight command-line tool for automating repetitive tasks, Sourcegraph's emphasis is on interactive developer tools for understanding and navigating large code bases. Unlike Comby's **sed**-like interface, Sourcegraph's automated code rewriting capability is driven from a UI, allowing users to review changes before they're made. Because Sourcegraph is a hosted service, it also has the ability to continuously monitor code bases and send alerts when a match occurs.

64. Telepresence
Assess
Telepresence is a tool that helps shorten the feedback loop of changes that usually require a deployment for proper testing. Developers can use it to plug a process that is running on their local machines into a remote Kubernetes cluster. This gives the local process access to the remote cluster's services and features, and the local service can also temporarily replace one of the cluster services.

In situations where the service integration setup has become somewhat unwieldy, Telepresence can boost developer productivity and enable more effective local testing. However, if you get into the habit of using a clever tool like this, you may have bigger problems. For example, if you use Telepresence because it has become impossible to set up all necessary dependencies for local development, you may want to investigate the complexity of your setup and architecture. If it becomes the only way for you to do service integration tests, consider looking into **consumer-driven contract testing** or other automated ways of integration testing.

65. Vite
Assess
Fast feedback is crucial for a good developer experience. Nothing breaks the flow of development more than having to wait a minute or two before getting feedback on the last code changes. Unfortunately, with applications growing in size and complexity, the popular build tools for front-end pipelines are often not fast enough anymore. Previously, we featured **esbuild**, which offers a significant performance improvement, because it's implemented in a compile-to-native language rather than JavaScript. **Vite**, which is built on top of esbuild, delivers **significant improvements** over other tools. It consists of two major parts: a dev server that provides rich feature enhancements over native ES modules, such as extremely fast Hot Module Replacement (HMR), and a build command that bundles your code with Rollup. Vite relies on ES modules, and unlike most older tools, it doesn't provide shimming or polyfills, which means it's not compatible with older browsers that don't support ES modules. In cases where older browsers had to be supported, some of our teams used Vite during development and other tools for production builds.

Languages and Frameworks
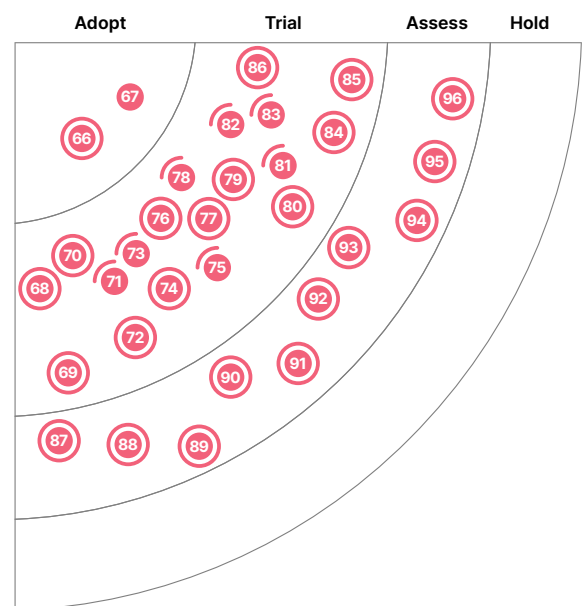
Adopt
66. Jetpack Compose
67. React Hooks

Trial
68. Arium
69. Chakra UI
70. DoWhy
71. Gatsby.js
72. Jetpack Hilt
73. Kotlin Multiplatform Mobile
74. lifelines
75. Mock Service Worker
76. NgRx
77. pydantic
78. Quarkus
79. React Native Reanimated 2.0
80. React Query
81. Tailwind CSS
82. TensorFlow Lite
83. Three.js
84. ViewInspector
85. Vowpal Wabbit
86. Zap

Assess
87. Headless UI
88. InsightFace
89. Kats
90. ksqlDB
91. Polars
92. PyTorch Geometric
93. Qiankun
94. React Three Fiber
95. Tauri
96. Transloco

Hold
—

| Adopt | Trial | Assess | Hold |

New   Moved in/out   No change

66. Jetpack Compose
Adopt

In a move that mirrors Apple's introduction of **SwiftUI**, Google introduced **Jetpack Compose** as a new and quite different approach to building user interfaces for modern Android applications. Compose brings more powerful tools and an intuitive Kotlin API. In most cases less code is needed, and it has become easier to create user interfaces at runtime rather than defining a static UI that can be filled with data. With **Compose Multiplatform** and **Kotlin Multiplatform** developers now have a unified toolkit to build desktop, web and native Android apps. Wear OS 3.0+ is included, too, and with support for iOS already present in **Kotlin Multiplatform Mobile**, it's likely that iOS will be supported by Compose in the future.

67. React Hooks
Adopt

React Hooks have introduced a new approach to managing stateful logic; given React components have always been closer to functions than classes, Hooks have embraced this and brought state to the functions, instead of using classes to take function to the state with methods. Another staple of state management in React applications is **Redux**, and we've already noted that it has come under scrutiny, suggesting that sometimes the complexity of Redux isn't worth it and in such cases a simple approach using Hooks is preferable. Rolling this completely on your own can quickly become tricky; therefore we recommend considering a combination of **React Context** and the useContext and useReducer hooks, along the lines explained in this **blog post**.

68. Arium
Trial

Arium is an automated testing framework for 3D applications written in Unity. Functional tests are an important part of a healthy test pyramid. Arium, which is built as a wrapper on the **Unity Test framework**, lets you write functional tests for 3D apps on multiple platforms. We've used it successfully in a few of our projects.

69. Chakra UI
Trial

Chakra UI is a UI component library for **React.js** that is designed for accessibility. We like it, especially for its accessibility features, including dark mode and compatibility with the Web Accessibility Initiative – Accessible Rich Internet Applications (WAI-ARIA) guidelines. Moreover, it is easy to test and customize which makes for a good development experience, accelerating the development process of UI solutions in production environments.

70. DoWhy
Trial

DoWhy is a Python library to perform end-to-end causal inference and analysis. Although machine-learning models can make predictions based on factual data, exploiting the correlation of variables that were present at the time, they're insufficient in scenarios where we need to ask *What if* and *Why* questions: *What if a variable changed? What would be the impact on the outcome?* Causal inference is an approach to answer such questions. It estimates the causal effect, that is, the magnitude by

which an outcome would change, if we changed one of the causal variables. This approach is applied when we can't arrive at the answer through observations and collecting data from A/B testing — due to the cost of experiments or limitations. The DoWhy library estimates the causal effect based on a process that uses the past collected facts and data as well as assumptions one can make knowing the domain. It uses a four-step process of modeling the causal relationships graph based on assumptions, identifying a cause for an outcome, estimating the causal effect and finally challenging those assumptions by refuting the result. We've used this library successfully in production, and it's one of the commonly used libraries in causal estimation use cases.

71. Gatsby.js
Trial

Although several frameworks promise the same ease of development and scalability typical of static site generators, we continue to have good experiences with **Gatsby.js**. In particular we've used it to build and deploy websites that scale to very large numbers of users without having to worry about capacity planning or deployment infrastructure. Our developers have also been impressed by the focus on accessibility and support for old browsers and that they could reuse their **React.js** experience. All in all, we feel Gatsby has matured well and is a solid choice in this space.

72. Jetpack Hilt
Trial

Jetpack Hilt has recently reached version 1.0, and we can report that we've had good experiences with it. Jetpack Hilt offers extensions for integrating Hilt with various other AndroidX libraries, such as WorkManager and Navigation. It further expands the reach of Hilt, to provide developers with a standard way of incorporating **Dagger** dependency injection into Android applications. We've featured **Koin** as a Kotlin-native dependency injection framework in the Radar before, and we would advise against attempting to replace Koin in a large existing codebase. However, when starting a new project, Hilt, it seems, is now the way to go.

73. Kotlin Multiplatform Mobile
Trial

For many organizations, cross-platform mobile development is becoming a strong option especially as the end-to-end experience of building mobile cross-platform applications becomes more enjoyable and efficient. **Kotlin Multiplatform Mobile** (KMM) is an SDK provided by JetBrains that leverages the **multiplatform capabilities** in **Kotlin** and includes tools and features designed to streamline the developer experience. 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. You 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. We're moving KMM to Trial as it is **evolving rapidly** and we're seeing a few organizations use this as their default.

74. lifelines
Trial

lifelines is a library for survival analysis in Python. Originally developed for birth and death events, it has evolved into a complete survival analysis library to predict any duration of time. Beyond medical use cases (such as answering, *How long does this population live for?*), we've used it in retail and manufacturing to answer questions like *How long users are subscribed to a service?* or *When should we do the next preventive maintenance?*

75. Mock Service Worker

Trial

Web applications, especially those for internal use in enterprises, are usually written in two parts. The user interface and some business logic run in the web browser while business logic, authorization and persistence run on a server. These two halves normally communicate via JSON over HTTP. The endpoints shouldn't be mistaken for a real API; they're simply an implementation detail of an application that is split across two run-time environments. At the same time, they provide a valid seam to test the pieces individually. When testing the JavaScript part, the server side can be stubbed and mocked at the network level by a tool such as **Mountebank**. **Mock Service Worker** offers an alternative approach of intercepting requests in the browser. This simplifies manual tests as well. Like Mountebank, Mock Service Worker is run outside the browser as a Node.js process for testing network interactions. In addition to REST interactions, it mocks GraphQL APIs — a bonus because GraphQL can be complex to mock manually at the network level.

76. NgRx

Trial

State management in React applications has been a recurring topic in the Radar, and we've recently clarified our position on **Redux**, a popular framework in this space. **NgRx** is, in essence, Redux for **Angular**. It's a framework for building reactive applications with Angular, providing ways to manage state and to isolate side effects. Our teams report that picking up NgRx was straightforward, not the least because it is built with **RxJS**, and they highlight a trade-off similar to the one we know from Redux: adding reactive state management comes with added complexity that only pays off in larger applications. The developer experience is enhanced by schematics, a scaffolding library and a set of tools that enable visual tracking of state and time-travel debugging.

77. pydantic

Trial

Originally type annotations were added to Python to support static analysis. However, considering how widely type annotations, and annotations in general, are used in other programming languages, it was only a matter of time before developers would begin to use Python's type annotations for other purposes. **pydantic** falls into this category. It allows you to use type annotations for data validation and settings management at run time. When data arrives as, say, a JSON document and needs to be parsed into a complex Python structure, pydantic ensures that the incoming data matches the expected types or reports an error if it doesn't. Although you can use pydantic directly, many developers have used it as part of **FastAPI**, one of the most popular Python web frameworks. In fact, using pydantic in FastAPI is considered so indispensable that a recently proposed change to Python, aimed at reducing the cost of loading annotated code into memory, was **reconsidered** because it would have broken the use of type annotations at run time.

78. Quarkus

Trial

We assessed **Quarkus** two years ago, and now our teams have more experience with it. Quarkus is a Kubernetes-native Java stack tailored for OpenJDK HotSpot and **GraalVM**. Over the past two years, Quarkus has wired those best-of-breed libraries in the Java world and streamlined the code configuration, giving our teams a pretty good developer experience. Quarkus has a very fast boot

time (tens of milliseconds) and a low RSS memory footprint; this is because of its **container-first** building approach: it uses ahead-of-time compilation techniques to do dependency injection at compile time and thus avoids the run-time costs of reflection. Our team has also had to endure the trade-offs: it takes nearly 10 minutes for Quarkus to build on our pipeline; some features that rely on annotations and reflection (such as ORM and serializer) are also limited. Part of these trade-offs are the result of using GraalVM. So if your application is not running for FaaS, using Quarkus with HotSpot is also a good choice.

79. React Native Reanimated 2.0
Trial

If we want animations in **React Native** applications, **React Native Reanimated 2.0** is the way to go. We previously had Reanimated 1.x, but it had issues related to the complexity of the Reanimated declarative language and also had some additional performance costs related to initialization and communication between the React Native JavaScript thread and the UI thread. Reanimated 2.0 is an attempt at reimagining how to run animations in the UI thread; it allows us to code the animations in JavaScript and run them on the UI thread using a new API called **animation worklets**. It does this by spawning a secondary JavaScript context on the UI thread that then is able to run JavaScript functions. We're using it in our React Native projects which need animations and like it a lot.

80. React Query
Trial

React Query is often described as the missing data-fetching library for React. Fetching, caching, synchronizing and updating server state is a common requirement in many React applications, and although the requirements are well-understood, getting the implementation right is notoriously difficult. React Query provides a straightforward solution using hooks. As an application developer you simply pass a function that resolves your data and leave everything else to the framework. We like that it works out-of-the-box but still offers a lot of configuration when needed. The developer tools, unfortunately not yet available for React Native, do help with understanding of how the framework works, which benefits developers new to it. In our experience, version 3 of the framework brought the stability needed to be used in production with our clients.

81. Tailwind CSS
Trial

Our developers have continued to be productive with **Tailwind CSS** and are impressed with its ability to scale with large teams and codebases. Tailwind CSS offers an alternative approach to CSS tools and frameworks that reduces complexity through lower-level utility CSS classes. The Tailwind CSS classes can easily be customized to suit any customer's visual identity. We've also found that it pairs particularly well with **Headless UI**. Tailwind CSS allows you to avoid writing any classes or CSS on your own which leads to a more maintainable codebase in the long term. It seems that Tailwind CSS offers the right balance between reusability and customization to create visual components.

82. TensorFlow Lite
Trial

Since we first mentioned **TensorFlow Lite** in the Radar in 2018, we've used it in several products and are happy to report that it works as advertised. The standard use case is to integrate pretrained models into mobile apps, but TensorFlow Lite also supports on-device learning which opens

further areas of application. Numerous examples on the website showcase many common areas of application, such as image classification and object detection, but also hint at new ways of interaction using, for example, pose estimation and gesture recognition.

83. Three.js
Trial

We first mentioned **Three.js** in the Radar in Assess back in 2017. Since then, this 3D rendering library for the web has evolved and improved rapidly. The standard WebGL APIs have improved, and Three.js has added support for WebXR, turning it into a viable tool for creating immersive experiences. At the same time, browser support for 3D rendering and WebXR device APIs has improved, making the web an increasingly attractive platform for 3D content. Although there are other 3D rendering libraries, our teams have come to prefer Three.js, especially when paired with **React Three Fiber** to abstract away some of the low-level details. We've found that developers still need to be conscious of performance issues and will sometimes need to restructure data to optimize rendering speed.

84. ViewInspector
Trial

When creating a user interface with **SwiftUI**, the idea is to build a view model that can be mapped easily to the elements of the user interface. In such cases, most of the testing can be done on the model, using the standard unit testing frameworks which makes these tests straightforward to write and fast to run. To test the bindings between the model and the views, developers usually reach for **XCUITest**, a test automation framework that launches the full application and remote controls the interface. It works, tests are reasonably stable, but they take a long time to run.

For a faster approach to writing unit tests for SwiftUI, try **ViewInspector**, an open-source framework that uses Swift's public reflection API to access the underlying views created by SwiftUI. With ViewInspector, a test simply instantiates a SwiftUI view, locates the interface elements that need to be tested and then makes assertions against them. Basic interactions such as taps can be tested, too. Like many UI testing frameworks, it provides an API to locate interface elements, either by specifying a path through the view hierarchy or by using a set of finder methods. These tests are usually simpler than XCUITests, and they run much faster. As a word of caution, though, given the ease with which tests can be written using ViewInspector, you might be tempted to over-test the interface. Testing simple one-to-one mappings is just double-entry bookkeeping. And even though ViewInspector makes it easier to test the SwiftUI code, remember to keep most of the logic in the model.

85. Vowpal Wabbit
Trial

Vowpal Wabbit is a general-purpose machine-learning library. Even though it was originally created at Yahoo! Research over a decade ago, we still want to mention it to highlight that it continues to be the place where many of the newest machine-learning techniques get added first. If you're interested in machine learning, you may want to keep an eye on the innovations in Vowpal Wabbit. Note also that Microsoft has shown a deeper interest in Vowpal Wabbit in recent years, employing a main contributor and integrating it into their Azure offerings, for example in their **machine-learning designer** and in **Personalizer**.

86. Zap

Trial

Zap is a super performant structured logging library for GoLang which is faster than the standard Log implementation and other logging libraries. Zap has both a "pretty" logger, providing a structured and `printf`-style interface, as well as an (even) faster implementation with just the structured interface. Our teams have used it extensively at scale and are happy to recommend it as their go-to solution.

87. Headless UI

Assess

Headless UI is an unstyled component library for either **React.js** or **Vue.js** from the same people that created **Tailwind CSS**. Our developers like that they don't have to customize or work around the default styles that other component libraries come with. The components' rich functionality and full accessibility, combined with the frictionless styling, allows developers to focus more productively on the business problem and user experience. Unsurprisingly, Headless UI also pairs well with Tailwind CSS classes.

88. InsightFace

Assess

InsightFace is an open source 2D and 3D deep face analysis toolbox, mainly based on **PyTorch** and MXNet. InsightFace uses some of the most recent and accurate methods for face detection, face recognition and face alignment. We're particularly interested in it, because it has one of the best implementations for ArcFace, a cutting-edge machine-learning model that detects the similarities of two images. InsightFace with ArcFace received a 99.83% accuracy score on the **Labeled Faces in the Wild (LFW)** data set. We're experimenting with it in the context of facial deduplication and have seen promising results.

89. Kats

Assess

Kats is a lightweight framework for performing time series analyses, recently released by Facebook Research. Time series analysis is an important area in data science; it encompasses the problem domains of forecasting, detection (including the detection of seasonalities, outliers and change points), feature extraction and multivariate analysis. Typically we tend to have different libraries for different techniques in a time series analysis. Kats though aims to be a one-stop shop for time series analyses and provides a set of algorithms and models for all the time series analysis problem domains. Previously we mentioned **Prophet**, also by Facebook Research, which is one of the models Kats implements for forecasting. We're looking forward to trying Kats in problems involving time series analyses.

90. ksqlDB

Assess

If you're using **Apache Kafka** and building stream-processing applications, **ksqlDB** is a great framework for writing simple applications using SQL-like statements. ksqlDB is not a traditional SQL database. However, it allows you to use lightweight SQL-like statements to build new Kafka **streams** or **tables** on top of the existing Kafka topics. The queries can pull data, similar to reading from a

traditional database, or push results to the application when incremental changes occur. You can choose to run it as a **standalone server** natively as part of your existing Apache Kafka installation or as a fully managed service on Confluent Cloud. We're using ksqlDB in simple data-processing use cases. In more complex use cases, where an application requires programming code beyond algebraic SQL queries, we continue to use data-processing frameworks such as **Apache Spark** or **Apache Flink** on top of Kafka. We recommend experimenting with ksqlDB in scenarios where the simplicity of the application allows it.

91. Polars

Assess

Polars is an in-memory data frame library implemented in **Rust**. Unlike other data frames (such as Pandas), Polars is multithreaded and safe for parallel operations. The in-memory data is organized in the **Apache Arrow** format for efficient analytic operations and to enable interoperability with other tools. If you're familiar with Pandas, you can quickly get started with Polars' Python bindings. We believe Polars, with Rust implementation and Python bindings, is a performant in-memory data frame to assess for your analytical needs.

92. PyTorch Geometric

Assess

PyTorch Geometric is a geometric deep learning extension library for **PyTorch**. Geometric deep learning aims to build neural networks that can learn from non-Euclidean data like graphs. Graph network-based machine-learning approaches have been of increasing interest in social network modeling and in biomedical fields, specifically in drug discovery. PyTorch Geometric provides an easy-to-use library to design complicated graph network problems like protein structure representation. It has GPU and CPU support and includes a good collection of graph-based machine-learning algorithms based on recent research.

93. Qiankun

Assess

Micro frontends have continued to gain in popularity since they were first introduced. However, it's easy to fall into **micro frontend anarchy** if teams fail to maintain consistency across an application, from styling technique to state management. **Qiankun**, which means heaven and earth in Chinese, is a JavaScript library built to provide an out-of-the-box solution for this. Qiankun is based on **single-spa**, so it allows different frameworks to coexist in a single application. It also provides style isolation and JavaScript sandbox to ensure the style or state of microapplications do not interfere with each other. Qiankun has received some attention in the community; our teams are also assessing it, hoping that it can support more friendly debugging.

94. React Three Fiber

Assess

With the rising interest in — and viability of — 3D and extended reality (XR) applications in web browsers, our teams have been experimenting with React Three Fiber for developing 3D experiences on the web. **React Three Fiber** is a library that takes the React.js component and state model and translates it to 3D objects rendered with the **Three.js** library. This approach opens up 3D web programming to the wider group of developers already familiar with React.js and the rich ecosystem

of tools and libraries surrounding it. However, when developing applications with React Three Fiber, our teams often have to manipulate the 3D scene imperatively. This doesn't mix well with the reactive component paradigm provided by React. There is no escaping the need to understand the basic 3D rendering mechanisms. The jury is still out on whether React Three Fiber offers enough abstraction to warrant learning its idiosyncrasies or if it's better just to work with Three.js directly.

95. Tauri
Assess

Tauri is an **Electron** alternative for building desktop applications using a combination of **Rust** tools and HTML, CSS, and JavaScript rendered in System's WebView. Unlike Electron which bundles Chromium, the applications built with Tauri leverage the underlying WebView, that is, WebKit on macOS, WebView2 on Windows and WebKitGTK on Linux. This approach has interesting trade-offs — on one hand you get small and fast application binaries; on the other hand, you need to verify compatibility quirks across WebViews of different systems.

96. Transloco
Assess

Transloco is a library for Angular to build multilingual applications. It can be used in templates and offers a function to cover more complex use cases. Because the translations are loaded on-demand at run time, Transloco makes it easy to implement language switching in the web browser. It also covers localization of numbers, dates and more using template pipes.

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 is a global technology consultancy that integrates strategy, design and engineering to drive digital innovation. We are 10,000+ people strong across 48 offices in 17 countries. Over the last 25+ years, we've delivered extraordinary impact together with our clients by helping them solve complex business problems with technology as the differentiator.

/thoughtworks