



Technology Radar

An opinionated guide to
technology frontiers

About the Radar	3
Radar at a glance	4
Contributors	5
Themes	6
The Radar	8
Techniques	11
Platforms	20
Tools	27
Languages and Frameworks	35

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 and 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 “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.



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 17 senior technologists at Thoughtworks. The TAB meets twice a year face-to-face and biweekly virtually. 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. This edition of the Thoughtworks Technology Radar is based on a meeting of the TAB in Barcelona in September 2022.



Rebecca Parsons (CTO)



Martin Fowler (Chief Scientist)



Bharani Subramaniam



Birgitta Böckeler



Brandon Byars



Camilla Falconi Crispim



Erik Dörnenburg



Fausto de la Torre



Hao Xu



Ian Cartwright



James Lewis



Marisa Hoenig



Mike Mason



Neal Ford



Perla Villarreal



Scott Shaw



Shangqi Liu

Themes

The mainstreaming of ML

Machine learning (ML) was once a realm reserved for only a lucky few who had the tools and resources to build cool things. Fortunately, we see a gradual mainstreaming of ML as computational power grows on devices of all sizes, open-source tools arrive and more stringent requirements and awareness around privacy and personalized information all converge to create a burgeoning ecosystem. Techniques such as [federated machine learning](#) allow for ML models that provide privacy for sensitive information. The field of [TinyML](#) allows models to execute on resource-constrained devices, moving inference to the edge which both frees resources and improves privacy for sensitive data. [Feature Stores](#) provide analogous benefits to the Model-View-Controller design pattern for application development, allowing a cleaner separation of concerns between data curation, model training and inference. Publicly available models such as [Stable Diffusion](#) highlight both the amazing capabilities of machine learning and the concerns around source data and ethics. ML components are also easier than ever to wire together, making it possible to build ML experiences and solutions with creative composition of custom business models and highly capable generic models. We applaud the new capabilities in this space and eagerly await future advancements.

The power of platforms as a product

The word “platform” continues to be one of the most used words during our Radar meetings because the concept is so pervasive in the industry. It pops up in many different manifestations, including business or domain-focused platforms but also infrastructure or developer experience platforms. Fundamentally, the root cause of many of the problems and disappointments that organizations experience with all platforms is the failure to properly treat them as products. For example, many platforms intended to be consumed by developers lack the user research and contextual analysis we expect in other types of products. Platform owners need to validate their assumptions about developers’ needs and respond to actual usage patterns. And like any good product, a platform needs ongoing support. It must evolve and adapt in response to the developer’s changing needs. Additionally, roles like project managers and business analysts often have different scopes than in traditional applications. The “platform as a product” metaphor only works when fully embraced as a practice rather than a trendy phrase.



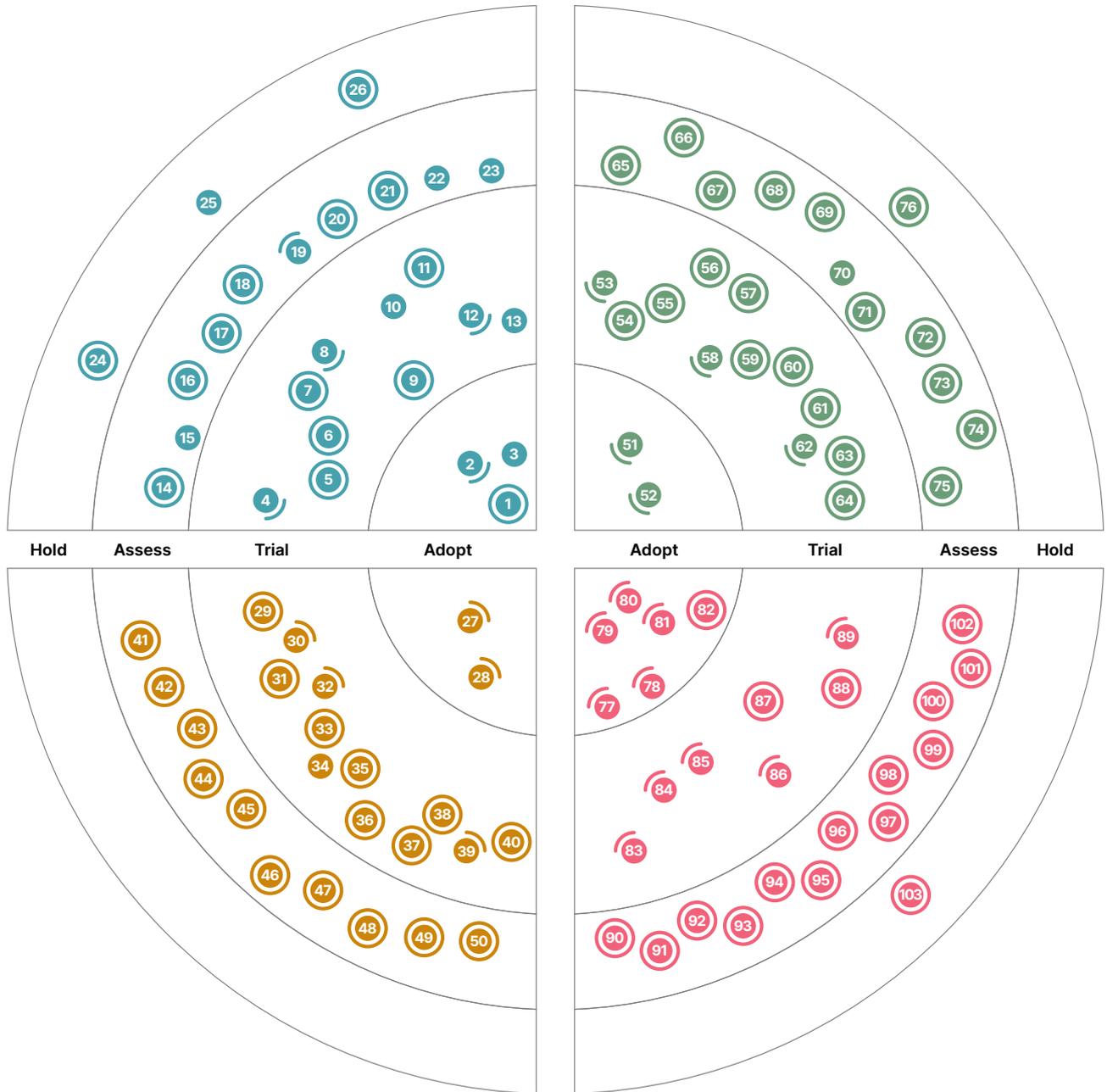
Moving data ownership to the edges

As we all too painfully know, centralization of any kind opens up the possibility of constriction, bottlenecks or unnecessary exposure. Thus, we constantly strive to find novel ways to break centralized coupling points, highlighted by several blips in our Radar. Based on research into conflict-free replicated data types (CRDTs), which enable data-based applications without a centralized database, the technique of local-first software/applications encourages developers to think about building around peer-to-peer data rather than using a centralized database. Moving data ownership to the edges also allows developers to take advantage of increased capabilities on devices, as showcased in the Mainstreaming of ML theme. For example, many capabilities such as facial recognition can occur on the edge, keeping the underlying data on the device forever.

Mobile should be modular, too

Software engineers have learned the value of structuring the architecture of an application primarily around domain concepts and business functionality. Technical concerns — a separation of UI from domain logic — are still important but play a secondary role. As mobile apps mature they often get larger, sometimes growing into so-called super apps, which comprise many services and can be seen as platforms in their own right. Apps that aren't quite as large but have picked up many capabilities over the years can usually be decomposed into modules, too, and companies find that mobile apps benefit from the same approach to modularity. Modular apps lend themselves to be written by multiple autonomous teams, which brings many well-documented benefits. Adding to the complexity is the requirement to deploy via an app store and the need to support native iOS and Android versions plus a web-based version, with subtle changes to accommodate each. We see better framework support for the unique tensions inherent in mobile development, but on the whole, despite the benefits, many organizations struggle to bring a modular approach to mobile development.

The Radar



 New  Moved in/out  No change

The Radar

Techniques

Adopt

1. Path-to-production mapping
2. Team cognitive load
3. Threat modeling

Trial

4. BERT
5. Component visual regression testing
6. Design tokens
7. Fake SMTP server to test mail-sending
8. Federated machine learning
9. Incremental developer platform
10. Micro frontends for mobile
11. Observability for CI/CD pipelines
12. SLSA
13. Software Bill of Materials

Assess

14. Carbon efficiency as an architectural characteristic
15. CUPID
16. GitHub push protection
17. Local-first application
18. Metrics store
19. Server-driven UI
20. SLIs and SLOs as code
21. Synthetic data for testing models
22. TinyML
23. Verifiable credentials

Hold

24. Satellite workers without “remote native”
25. SPA by default
26. Superficial cloud native

Platforms

Adopt

27. Backstage
28. Delta Lake

Trial

29. AWS Database Migration Service
30. Colima
31. Databricks Photon
32. DataHub
33. DataOps.live
34. eBPF
35. Feast
36. Monte Carlo
37. Retool
38. Seldon Core
39. Teleport
40. VictoriaMetrics

Assess

41. Bun
42. Databricks Unity Catalog
43. Dragonfly
44. Edge Impulse
45. GCP Vertex AI
46. Gradient
47. IAM Roles Anywhere
48. Keptn
49. OpenMetadata
50. OrioleDB

Hold

—

The Radar

Tools

Adopt

- 51. Great Expectations
- 52. k6

Trial

- 53. Apache Superset
- 54. AWS Backup Vault Lock
- 55. AWS Control Tower
- 56. Clumio Protect
- 57. Cruft
- 58. Excalidraw
- 59. Hadolint
- 60. Kaniko
- 61. Kusto Query Language
- 62. Spectral
- 63. Styra Declarative Authorization Service
- 64. xbar for build monitoring

Assess

- 65. Clasp
- 66. Databricks Overwatch
- 67. dbtvault
- 68. git-together
- 69. Harness Cloud Cost Management
- 70. Infracost
- 71. Karpenter
- 72. Mizu
- 73. Soda Core
- 74. Teller
- 75. Xcode Cloud

Hold

- 76. Online services for formatting or parsing code

Languages and Frameworks

Adopt

- 77. io-ts
- 78. Kotest
- 79. NestJS
- 80. React Query
- 81. Swift Package Manager
- 82. Yjs

Trial

- 83. Azure Bicep
- 84. Camunda
- 85. Gradle Kotlin DSL
- 86. Jetpack Media3
- 87. Ladle
- 88. Moshi
- 89. Svelte

Assess

- 90. Aleph.js
- 91. Astro
- 92. BentoML
- 93. Carbon Aware SDK
- 94. Cloudscape
- 95. Connect
- 96. Cross device SDK
- 97. Cypress Component Testing
- 98. JobRunr
- 99. Million
- 100. Soketi
- 101. Stable Diffusion
- 102. Synthetic Data Vault

Hold

- 103. Carbon

Techniques

Adopt

1. Path-to-production mapping
2. Team cognitive load
3. Threat modeling

Trial

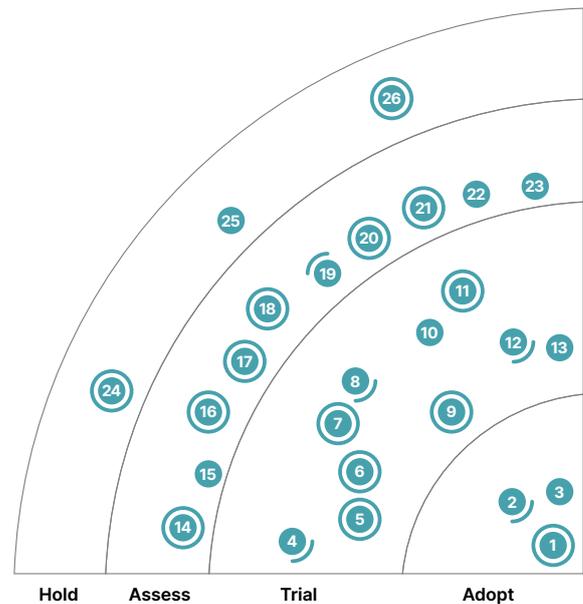
4. BERT
5. Component visual regression testing
6. Design tokens
7. Fake SMTP server to test mail-sending
8. Federated machine learning
9. Incremental developer platform
10. Micro frontends for mobile
11. Observability for CI/CD pipelines
12. SLSA
13. Software Bill of Materials

Assess

14. Carbon efficiency as an architectural characteristic
15. CUPID
16. GitHub push protection
17. Local-first application
18. Metrics store
19. Server-driven UI
20. SLIs and SLOs as code
21. Synthetic data for testing models
22. TinyML
23. Verifiable credentials

Hold

24. Satellite workers without "remote native"
25. SPA by default
26. Superficial cloud native



○ New ◐ Moved in/out ● No change



1. Path-to-production mapping

Adopt

Although path-to-production mapping has been a near-universal practice at Thoughtworks since codifying [Continuous Delivery](#), we often come across organizations unfamiliar with the practice. The activity is most often done in a workshop with a cross-functional group of people — that includes everyone involved in designing, developing, releasing and operating the software — around a shared whiteboard (or virtual equivalent). First, the steps in the process are listed in order, from the developer workstation all the way to production. Then, a facilitated session is used to capture further information and pain points. The most common technique we see is based on [value-stream mapping](#), although plenty of [process map](#) variants are equally valuable. The activity is often eye-opening for many of the participants, as they identify delays, risks and inconsistencies and continue to use the visual representation for the continuous improvement of the build and deploy process. We consider this technique so foundational that we were surprised to discover we hadn't blipped it before.

2. Team cognitive load

Adopt

Team interaction is a key concept when redesigning an organization for business agility and speed. These interactions will be reflected in the software being built (see [Conway's Law](#)) and indicate how effectively teams can autonomously deliver value to their customers. Our advice is to be intentional about how teams are designed and how they interact. Because we believe that organizational design and team interactions evolve over time, we think it's particularly important to measure and keep track of the team cognitive load, which indicates how easy or difficult teams find building, testing and maintaining their services. We've been using a [template](#) to assess team cognitive load that is based on ideas by the authors of the [Team Topologies](#) book.

We continue to be impressed by the positive impact of applying this book's concepts when communicating to clients and redesigning organizations. The authors recommend a simple but powerful approach to organizational design, identifying just four types of teams and three modes of interaction; this helps reduce ambiguity within the organization and provides a common vocabulary for teams, stakeholders and leadership to describe and design a team's work. To implement an org design change, we design the ideal to-be team topologies structure, apply any technical/staffing constraints (i.e., not enough employees) and then end up with the final to-be structure. That allows us to better advise clients and anticipate whether we're indeed improving cognitive load by comparing the as-is/to-be team structures.

3. Threat modeling

Adopt

We continue to recommend that teams carry out [threat modeling](#) — a set of techniques to help you identify and classify potential threats during the development process — but we want to emphasize that this is not a one-off activity only done at the start of projects; teams need to avoid the [security sandwich](#). This is because throughout the lifetime of any software, new threats will emerge and existing ones will continue to evolve thanks to external events and ongoing changes to requirements and architecture. This means that threat modeling needs to be repeated periodically — the frequency of repetition will depend on the circumstances and will need to consider factors such as the cost of running the exercise and the potential risk to the business. When used in conjunction with other techniques, such as establishing cross-functional security requirements to address common risks in the project's technologies and using automated security scanners, threat modeling can be a powerful asset.



4. BERT

Trial

Since we last talked about [BERT](#) (Bidirectional Encoder Representations from Transformers) in the Radar, our teams have successfully used it in a few natural language processing (NLP) projects. In one of our engagements, we observed significant improvements when we switched from the default BERT tokenizer to a domain-trained word-piece tokenizer for queries that contain nouns like brand names or dimensions. Although NLP has several new transformer models, BERT is well understood with good documentation and a vibrant community, and we continue to find it effective in an enterprise NLP context.

5. Component visual regression testing

Trial

[Visual regression testing](#) is a useful and powerful tool to have in your toolbox, but it has a significant cost given it's done for the entire page. With the rise of component-based frameworks such as [React](#) and [Vue](#), we've also seen the rise of component visual regression testing. This technique strikes a good balance between value and cost to ensure that no undesired visuals have been added to the application. In our experience, component visual regression testing presents fewer false positives and promotes a good architectural style. By using it with tools such as [Vite](#) and the webpack feature [Hot Module Replacement \(HMR\)](#), it could be seen as a paradigm shift for applying test-driven development to front-end development.

6. Design tokens

Trial

When faced with the challenge of using a [design system](#) consistently across many form factors and platforms, the team at Salesforce came up with the concept of [design tokens](#). The tokens store values, such as colors and fonts, in one central place. This makes it possible to [separate options from decisions](#), and it significantly improves [collaboration between teams](#). Design tokens are not new, but with the introduction of tools like [Tailwind CSS](#) and [Style Dictionary](#), we see design tokens being used more often.

7. Fake SMTP server to test mail-sending

Trial

Using test email accounts or entire test SMTP (Single Mail Transfer Protocol) servers remains a common software testing practice. However, using a real server carries the risk that [test emails will be sent to real people](#) and often complicates automated integration testing. We've seen success using a fake SMTP server to test mail sending, which records a request to send an email without actually sending it. Multiple open-source tools exist in this space, including [fake-smtp-server](#), which renders emails in a web UI for visual testing, and [mountebank](#), which exposes the sent emails through a REST API for integration testing. We recommend exploring this technique to reduce risk and improve testing efficiency.

8. Federated machine learning

Trial

We're now seeing client projects that use federated machine learning (ML). Traditionally, ML model training has required data to be placed in a centralized location where the relevant training algorithm can be run. From a privacy point of view, this is problematic, especially when the training data



contains sensitive or personally identifiable information; users might be reluctant to share data or local data protection legislation may prevent us from moving data to a central location. Federated ML is a decentralized technique for training on a large and diverse set of data that allows the data to remain remote — for example, on a user’s device. Network bandwidth and the computational limitations of devices still present significant technical challenges, but we like the way federated ML leaves users in control of their own personal information.

9. Incremental developer platform

Trial

We’ve been writing about developer platforms and how to build them in almost every edition of the Radar since 2017. In the meantime, the [Team Topologies](#) book has also done a great job of describing the ideal of a platform that supports developers with “self-service APIs, tools, services and knowledge.” However, we often see teams shooting for too much of that platform vision too fast. Instead, building an incremental developer platform is key.

[Team Topologies](#) recommends to always strive for what they call the “Thinnest Viable Platform” necessary at any given stage, where the first version could even be just a set of documentation on a wiki. The next increment could increase the service level by providing templates or allowing teams to create pull requests. Further increments could then introduce self-service APIs, but only if valuable. In short, even though we’ve cautioned against fully [ticket-driven platform operating models](#), going from zero to self-service is the other extreme. Pace yourself, [treat your platform as a product](#) and build it up incrementally.

10. 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 apps as well. When an app becomes sufficiently large and complex, it becomes necessary to distribute the development over multiple teams. This presents a number of challenges around team autonomy, repository structures and integration frameworks. In the past we’ve mentioned [Atlas and BeeHive](#), but these frameworks failed to gain traction and are no longer in active development. More recent approaches include [Tuist](#) or the [Swift Package Manager](#) for integrating the work of multiple teams into a single app. But in our experience, teams often end up implementing their own framework for integration. While we definitely see a need for modularity in scaling up mobile development teams, the case for micro frontends is less certain. This is because while micro frontends imply a direct correspondence between teams and pages or components, this structure could end up blurring responsibilities for business domain contexts, thereby increasing [team cognitive load](#). Our advice is to follow the basics of good, clean application design, embrace modularity when scaling up to multiple teams and adopt a micro frontend architecture only when the modules and the business domain are strongly aligned.

11. Observability for CI/CD pipelines

Trial

Observability practices have shifted the conversation from monitoring for well-understood problems to helping troubleshoot unknown problems in distributed systems. We’ve seen success taking that perspective outside of the traditional production environment by applying observability for CI/CD



pipelines to help optimize testing and deployment bottlenecks. Complex pipelines create developer friction when they run too slow or suffer from nondeterminism, reducing important feedback loops and hindering developer effectiveness. Additionally, their role as critical deployment infrastructure creates stress points during periods of rapid deployments, as happened to several organizations responding to the recent log4shell vulnerability. The concept of traces translates nicely to pipelines: instead of capturing the cascade of service calls, child spans capture information about each stage of the build. The same waterfall charts used to analyze a call flow in a distributed architecture can also be effective in helping us to identify bottlenecks in pipelines, even complex ones with fan-in and fan-out. This enables far more focused optimization efforts. While the technique should work with any tracing tool, [Honeycomb](#) supports a tool called [buildevents](#) that helps capture pipeline trace information. An alternative approach of capturing information already exposed by CI/CD platforms, taken by the open-source [buildviz](#) (built and maintained by a Thoughtworker), allows for a similar investigation without changing the step configurations themselves.

12. SLSA

Trial

As software continues to grow in complexity, the threat vector of software dependencies becomes increasingly challenging to guard against. Supply chain Levels for Software Artifacts, or [SLSA](#) (pronounced “salsa”), is a consortium-curated set of guidance for organizations to protect against supply chain attacks, evolved from internal guidance Google has been using for years. We appreciate that SLSA doesn’t promise a “silver bullet,” tools-only approach to securing the supply chain, but it does provide a checklist of concrete threats and practices along a maturity model. The [threat model](#) is easy to follow with real-world examples of attacks, and the [requirements](#) provide guidance to help organizations prioritize actions based on levels of increasing robustness to improve their supply chain security posture. Since we first mentioned it in the Radar, SLSA has added more detail around [software attestations](#) with examples to track concerns like [build provenance](#). Our teams have found SLSA to strike a nice balance between implementation guidance and higher-level awareness around supply chain threats.

13. Software Bill of Materials

Trial

With continued pressure to keep systems secure and no reduction in the general threat landscape, a machine-readable Software Bill of Materials (SBOM) may help teams stay on top of security problems in the libraries that they rely on. Since the original [Executive Order](#) was published, the industry has gained clarity and understanding of what an SBOM is and how to create one; the National Institute of Standards and Technology (NIST), for example, now has more [specific advice](#) on how to comply with the order. We’ve had production experience using SBOMs on projects ranging from small companies to large multinationals and even government departments, and we’re convinced they provide a benefit. More organizations and governments should consider requiring SBOMs for the software they use. The technique will be strengthened by the new tools that continue to emerge, such as the [Firebase Android BOM](#) that automatically aligns an application’s library dependencies to those listed in the BOM.



14. Carbon efficiency as an architectural characteristic

Assess

Sustainability is a topic that demands the attention of enterprises. In the software development space its importance has increased, and we're now seeing [different ways](#) to approach this topic. Looking at the carbon footprint of building software, for example, we recommend assessing carbon efficiency as an architectural characteristic. An architecture that takes into consideration carbon efficiency is one where design and infrastructure choices have been made in order to minimize energy consumption and therefore carbon emissions. The measurement tooling and advice in this space is maturing, making it feasible for teams to consider carbon efficiency alongside other factors such as performance, scalability, financial cost and security. Like almost everything in software architecture, this should be considered a trade-off; our advice is to think about this as one additional characteristic in a whole set of relevant [quality attributes](#) that are driven and prioritized by organizational goals and not left to a small cadre of experts to ponder in a siloed manner.

15. CUPID

Assess

How do you approach writing good code? How do you judge if you've written good code? As software developers, we're always looking for catchy rules, principles and patterns that we can use to share a language and values with each other when it comes to writing simple, easy-to-change code.

Daniel Terhorst-North has recently made a new attempt at creating such a checklist for good code. He argues that instead of sticking to a set of rules like [SOLID](#), using a set of properties to aim for is more generally applicable. He came up with what he calls the [CUPID](#) properties to describe what we should strive for to achieve "joyful" code: Code should be composable, follow the Unix philosophy and be predictable, idiomatic and domain based.

16. GitHub push protection

Assess

The accidental publication of secrets seems to be a perennial issue with tools such as [Talisman](#) popping up to help with the problem. Before now, GitHub Enterprise Cloud users with an Advanced Security License could enable security scanning on their accounts, and any secrets (API keys, access tokens, credentials, etc.) that were accidentally committed and pushed would trigger an alert. [GitHub push protection](#) takes this one step further, and brings it one step earlier in the development workflow, by blocking changes from being pushed at all if secrets are detected. This needs to be configured for the organization and applies, of course, only to license holders, but additional protection from publishing secrets is to be welcomed.

17. Local-first application

Assess

In a centralized application, the data on the server is the single source of truth — any modification to the data must go through the server. Local data is subordinate to the server version. This seems like a natural and inevitable choice to enable collaboration among multiple users of the software. Local-first application, or [local-first software](#), is a set of principles that enables both collaboration and local



data ownership. It prioritizes the use of local storage and local networks over servers in remote data centers or the cloud. Techniques like conflict-free replicated data types (CRDTs) and peer-to-peer (P2P) networks have the potential to be a foundational technology for realizing local-first software.

18. Metrics store

Assess

Metrics store, sometimes referred to as headless business intelligence (BI), is a layer that decouples metrics definitions from their usage in reports and visualizations. Traditionally, metrics are defined inside the context of BI tools, but this approach leads to duplication and inconsistencies as different teams use them in different contexts. By decoupling the definition in the metrics store, we get clear and consistent reuse across BI reports, visualizations and even embedded analytics. This technique is not new; for example, Airbnb introduced [Minerva](#) a year ago. However, we're now seeing considerable traction in the data and analytics ecosystem with more tools supporting metrics stores out of the box.

19. Server-driven UI

Assess

Server-driven UI continues to be a hot topic of discussion in mobile circles because it offers the potential for developers to take advantage of faster change cycles without falling foul of an app store's policies around revalidation of the mobile app itself. Server-driven UI separates the rendering into a generic container in the mobile app while the structure and data for each view is provided by the server. This means that changes that once required a round trip to an app store can now be accomplished via simple changes to the responses the server sends. While some very large mobile app teams have had great success with this technique, it also requires a substantial investment in building and maintaining a complex proprietary framework. Such an investment requires a compelling business case. Until the case is made, it might be best to proceed with caution; indeed, we've experienced some horrendous, overly configurable messes that didn't actually deliver on the promised benefits. But with the backing of behemoths such as Airbnb and Lyft, we may very well see some useful frameworks emerge that help tame the complexity. Watch this space.

20. SLIs and SLOs as code

Assess

Since Google first popularized service-level indicators (SLIs) and service-level objectives (SLOs) as part of their site reliability engineering (SRE) practice, observability tools like [Datadog](#), [Honeycomb](#) and [Dynatrace](#) started incorporating SLO monitoring into their toolchains. [OpenSLO](#) is an emerging standard that allows defining SLIs and SLOs as code, using a declarative, vendor-neutral specification language based on the YAML format used by [Kubernetes](#). While the standard is still quite new, we're seeing some encouraging momentum, as with Sumo Logic's contribution of the [slogen](#) tool to generate monitoring and dashboards. We're excited by the promise of versioning SLI and SLO definitions in code and updating observability tooling as part of the CI/CD pipeline of the service being deployed.



21. Synthetic data for testing models

Assess

During our discussions for this edition of the Radar, several tools and applications for synthetic data generation came up. As the tools mature, we've found that using synthetic data for testing models is a powerful and broadly useful technique. Although not intended as a substitute for real data in validating the discrimination power of machine-learning models, synthetic data can be used in a variety of situations. For example, it can be used to guard against catastrophic model failure in response to rarely occurring events or to test data pipelines without exposing personally identifiable information. Synthetic data is also useful for exploring edge cases that lack real data or for identifying model bias. Some helpful tools for generating data include [Faker](#) or [Synth](#), which generate data that conforms to desired statistical properties, and tools like [Synthetic Data Vault](#) that can generate data that mimics the properties of an input data set.

22. TinyML

Assess

We continue to be excited by the [TinyML](#) technique and the ability to create machine learning (ML) models designed to run on low-powered and mobile devices. Until recently, executing an ML model was seen as computationally expensive and, in some cases, required special-purpose hardware. While creating the models still broadly sits within this classification, they can now be created in a way that allows them to be run on small, low-cost and low-power consumption devices. If you've been considering using ML but thought it unrealistic because of compute or network constraints, then this technique is worth assessing.

23. Verifiable credentials

Assess

When we first included it in the Radar two years ago, verifiable credentials (VC) was an intriguing standard with some promising potential applications, but it wasn't widely known or understood outside the community of enthusiasts. This was particularly true when it came to the credential-granting institutions, such as state governments, who would be responsible for implementing the standards. Two years and one pandemic later, the demand for cryptographically secure, privacy-respecting and machine-verifiable electronic credentials has grown and, as a result, governments are starting to wake up to VC's potential. We're now starting to see VC crop up in our work for public-sector clients. The [W3C standard](#) puts credential holders at the center, which is similar to our experience when using physical credentials: users can put their verifiable credentials in their own digital wallets and show them to anyone at any time without the permission of the credentials' issuer. This decentralized approach also enables users to better manage and selectively disclose their own information which greatly improves data privacy protection. For example, powered by zero-knowledge proof technology, you can construct a verifiable credential to prove that you're an adult without revealing your birthday. It's important to note that although many VC-based [decentralized identity](#) solutions rely on blockchain technology, blockchain is not a prerequisite for all VC implementations.



24. Satellite workers without “remote native”

Hold

The term “remote team setup” does not just describe one setup; it encompasses multiple **patterns and flavors**. And many teams have been changing patterns recently. They’re coming out of the “everybody always remote” mode that was forced on them by a pandemic and moving into a pattern of (often rotating) satellite workers, where part of the team is co-located and part of the team is remote. We see many of them failing to properly consider what this means for their ways of working. Satellite workers without “remote native” ways of working is a slip back into privileging co-located practices. In a setup with satellite workers, it’s important to still **use “remote native” processes and approaches by default**. For example, if the co-located part of the team joins a meeting together, they should still all be on their individual laptops to participate in digital collaboration or meeting chat. Teams need to be aware of the risk of excluding their satellite workers and creating silos and feelings of exclusion. If you know that you’ll always have at least one satellite team member, the default ways of working should assume remoteness.

25. SPA by default

Hold

The prevalence of teams choosing a single-page application (SPA) when they need a website continues. We remain concerned that people aren’t properly recognizing SPAs as an architectural style to begin with; instead they’re immediately jumping into framework selection. SPAs incur complexity that simply doesn’t exist with traditional server-based websites: issues such as search engine optimization, browser history management, web analytics and first page load time all need to be addressed. Proper analysis and consideration of the trade-offs is required to determine if that complexity is warranted for business or user experience reasons. Too often teams are skipping that trade-off analysis, blindly accepting the complexity of SPAs by default even when business needs don’t justify it. We still see some developers who aren’t aware of an alternative approach because they’ve spent their entire career in a framework like React. We believe that many websites will benefit from the simplicity of server-side logic, and we’re encouraged by techniques like **Hotwire** that help close the gap on user experience.

26. Superficial cloud native

Hold

The term “cloud native” was originally used to describe architectures with characteristics that took maximum advantage of public cloud hosting. Examples include distributed architectures composed of many small, stateless and collaborating processes, and systems with high levels of automation for building, testing and deploying applications. However, we’ve noticed a growing trend toward superficial cloud native designs that simply use a lot of a cloud vendor’s proprietary services and stop there without revisiting the fundamentally monolithic, brittle or toil-intensive nature of the application. It’s important to remember that serverless functions by themselves don’t make an application more resilient or easier to maintain and that cloud native is really a matter of design rather than a set of implementation choices.

Platforms



Adopt

- 27. Backstage
- 28. Delta Lake

Trial

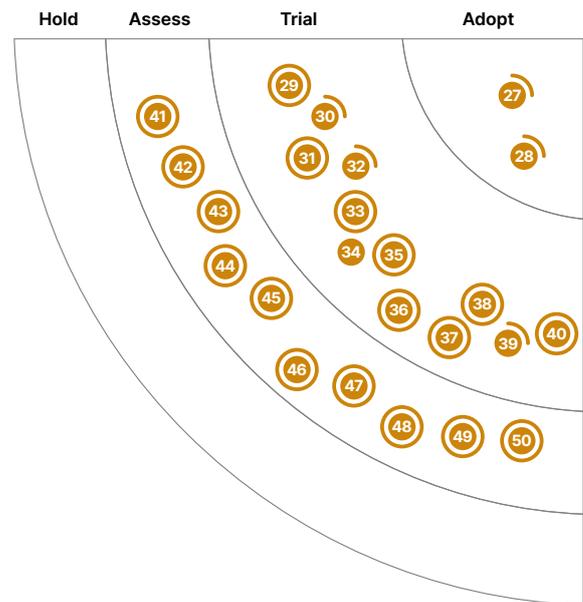
- 29. AWS Database Migration Service
- 30. Colima
- 31. Databricks Photon
- 32. DataHub
- 33. DataOps.live
- 34. eBPF
- 35. Feast
- 36. Monte Carlo
- 37. Retool
- 38. Seldon Core
- 39. Teleport
- 40. VictoriaMetrics

Assess

- 41. Bun
- 42. Databricks Unity Catalog
- 43. Dragonfly
- 44. Edge Impulse
- 45. GCP Vertex AI
- 46. Gradient
- 47. IAM Roles Anywhere
- 48. Keptn
- 49. OpenMetadata
- 50. OrioleDB

Hold

—



○ New ● Moved in/out ● No change



27. Backstage

Adopt

In an increasingly digital world, improving developer effectiveness in large organizations is often a core concern of senior leaders. We've seen enough value with developer portals in general and **Backstage** in particular that we're happy to recommend it in Adopt. Backstage is an open-source developer portal platform created by Spotify that improves discovery of software assets across the organization. It uses Markdown **TechDocs** that live alongside the code for each service, which nicely balances the needs of centralized discovery with the need for distributed ownership of assets. Backstage supports software templates to accelerate new development and a plugin architecture that allows for extensibility and adaptability into an organization's infrastructure ecosystem.

Backstage Service Catalog uses YAML files to track ownership and metadata for all the software in an organization's ecosystem; it even lets you track third-party SaaS software, which usually requires tracking ownership.

28. Delta Lake

Adopt

Delta Lake is an **open-source storage layer**, implemented by Databricks, that attempts to bring ACID transactions to big data processing. In our Databricks-enabled **data lake** or **data mesh** projects, our teams prefer using Delta Lake storage over the direct use of file storage types such as **AWS S3** or **ADLS**. Until recently, Delta Lake has been a closed proprietary product from Databricks, but it's now open source and accessible to non-Databricks platforms. However, our recommendation of Delta Lake as a default choice currently extends only to Databricks projects that use **Parquet** file formats. Delta Lake facilitates concurrent data read/write use cases where file-level transactionality is required. We find Delta Lake's seamless integration with Apache Spark **batch** and **micro-batch** APIs very helpful, particularly features such as **time travel** (accessing data at a particular point in time or commit reversion) as well as **schema evolution** support on write.

29. AWS Database Migration Service

Trial

Many of our teams have successfully used **AWS Database Migration Service** (DMS) to migrate data to and from AWS. In one of our Digital Transformation engagements, we achieved nearly zero downtime cut-over to the new system as we migrated data from Microsoft SQL Server to an AWS Relational Database Service (RDS) PostgreSQL instance. Such transformations involve many moving parts that require planning and coordination across multidisciplinary teams, but for data migration we're quite happy with DMS. It automatically manages the deployment, management and monitoring of all required resources. Over the years DMS has matured to support several **source** and **target** databases, and we continue to like it.

30. Colima

Trial

Colima is becoming a popular open alternative to Docker Desktop. It provisions the **Docker** container run time in a Lima VM, configures the Docker CLI on macOS and handles port-forwarding and volume mounts. Colima uses **containerd** as its run time, which is also the run time on most managed



Kubernetes services — improving the important dev-prod parity. With Colima you can easily use and test the latest features of containerd, such as lazy loading for container images. We've been having good results with Colima in our projects. When in the Kubernetes space, we also use **nerdctl**, a Docker-compatible CLI for containerd. Since Kubernetes has deprecated Docker as container run time and most managed-services (EKS, GKE, etc) are following its lead, more people will be looking to containerd native tools, hence the importance of tools like nerdctl. In our opinion, Colima is realizing its strong potential and becoming a go-to option as an alternative to Docker Desktop.

31. Databricks Photon

Trial

Starting with Databricks 9.1 LTS (Long Term Support), a new run time became available called **Databricks Photon**, an alternative that was rewritten from the ground up in C++. Several of our teams have now used Photon in production and have been pleased with the performance improvements and corresponding cost savings. Actual improvements and changes in costs will depend upon multiple factors such as data set size and transaction types. We recommend trialing against a realistic workload to gather data for a comparison before making any decision on Photon's use.

32. DataHub

Trial

Since we first mentioned **data discoverability** in the Radar, LinkedIn has evolved **WhereHows** to **DataHub**, the next generation platform that addresses data discoverability via an extensible metadata system. Instead of crawling and pulling metadata, DataHub adopts a push-based model where individual components of the data ecosystem publish metadata via an API or a stream to the central platform. This push-based integration shifts ownership from the central entity to individual teams, making them accountable for their metadata. As a result, we've used DataHub successfully as an organization-wide metadata repository and entry point for multiple autonomously maintained data products. When taking this approach, be sure to keep it lightweight and avoid the slippery slope leading to centralized control over a shared resource.

33. DataOps.live

Trial

DataOps.live is a data platform that automates environments in **Snowflake**. Inspired by **DevOps** practices, DataOps.live lets you treat the data platform like any other web platform by embracing continuous integration and continuous delivery (CI/CD), automated testing, observability and code management. You can roll back changes immediately without impacting the data or recover from complete failures and rebuild a fresh Snowflake tenant in minutes or hours instead of days. Our teams had good experiences with DataOps.live, because it allowed us to iterate quickly when building data products on top of Snowflake.

34. eBPF

Trial

For several years now, the Linux kernel has included the extended Berkeley Packet Filter (**eBPF**), a virtual machine that provides the ability to attach filters to particular sockets. But eBPF goes far beyond packet filtering and allows custom scripts to be triggered at various points within the kernel



with very little overhead. By allowing you to run sandboxed programs within the operating system kernel, application developers can run eBPF programs to add additional capabilities to the operating system at run time. Some of our projects require troubleshooting and profiling at the system call level, and our teams found that tools like [bcc](#) and [bpftrace](#) have made their jobs easier. Observability and network infrastructure also benefit from eBPF — for example, the [Cilium](#) project can implement traffic load balancing and observability [without sidecar overhead](#) in [Kubernetes](#), and [Hubble](#) provides further security and traffic observability on top of it. The [Falco](#) project uses eBPF for security monitoring, and the [Katran](#) project uses eBPF to build more efficient L4 load balancing. The eBPF community is growing rapidly, and we're seeing more and more synergy with the field of observability.

35. Feast

Trial

[Feast](#) is an open-source [Feature Store](#) for machine learning. It has several useful properties, including generating point-in-time correct feature sets — so error-prone future feature values do not leak to models during training — and supporting both streaming and batch data sources. However, it currently only supports timestamped structured data and therefore may not be suitable if you work with unstructured data in your models. We've successfully used Feast at a significant scale as an offline store during model training and as an online store during prediction.

36. Monte Carlo

Trial

[Monte Carlo](#) is a data observability platform. Using machine learning models, it infers and learns about data, identifying issues and notifying users when they arise. It allows our teams to maintain data quality across ETL pipelines, data lakes, data warehouses and business intelligence (BI) reports. With features such as monitoring dashboards as code, a central data catalog and field-level lineage, our teams find Monte Carlo to be an invaluable tool for overall data governance.

37. Retool

Trial

In previous editions, we've recommended assessing [bounded low-code platforms](#) as a method for applying low-code solutions to specific use cases in very limited domains. We've seen some traction in this space, specifically with [Retool](#), a low-code platform that our teams use to build solutions for internal users, predominantly to query and visualize data. It allows them to produce non-business-critical read-only solutions faster. The main reported benefits of Retool are its UI components and its ability to be integrated quickly and easily with common data sources.

38. Seldon Core

Trial

[Seldon Core](#) is an open-source platform to package, deploy, monitor and manage machine learning models in [Kubernetes](#) clusters. With out-of-the-box support for several machine-learning frameworks, you can easily containerize your models using [prepackaged inference servers](#), [custom inference servers](#) or [language wrappers](#). With distributed tracing through [Jaeger](#) and model explainability via [Alibi](#), Seldon Core addresses several last-mile delivery challenges with machine learning deployments, and our data teams like it.



39. Teleport

Trial

Teleport is a tool for **zero trust** network access to infrastructure. Traditional setups require complex policies or jump servers to restrict access to critical resources. Teleport, however, simplifies this with a unified access plane and with fine-grained authorization controls that replace jump servers, VPNs or shared credentials. Implemented as a single binary with out-of-the-box support for several protocols (including SSH, RDP, **Kubernetes** API, MySQL, **MongoDB** and PostgreSQL wire protocols), Teleport makes it easy to set up and manage secured access across Linux, Windows or Kubernetes environments. Since we first mentioned it in the Radar, a few teams have used Teleport and our overall positive experience prompted us to highlight it.

40. VictoriaMetrics

Trial

Modern observability relies on collecting and aggregating an exhaustive set of granular metrics to fully understand, predict and analyze system behavior. But when applied to a cloud native system composed of many redundant and cooperating processes and hosts, the cardinality (or number of unique time series) becomes unwieldy because it grows exponentially with each additional service, container, node, cluster, etc. When dealing with high-cardinality data, we've found that **VictoriaMetrics** performs well. VictoriaMetrics is particularly useful for operating **Kubernetes**-hosted **microservice** architectures, and the VictoriaMetrics operator makes it easy for teams to implement their own monitoring in a self-service way. We also like its componentized architecture and ability to continue collecting metrics even when the central server is unavailable. Although our team has been happy with VictoriaMetrics, this is a rapidly evolving area, and we'd recommend keeping an eye on other high-performance, **Prometheus**-compatible time series databases such as **Cortex** or **Thanos**.

41. Bun

Assess

Bun is a new JavaScript runtime, similar to **Node.js** or **Deno**. Unlike Node.js or Deno, however, Bun is built using WebKit's JavaScriptCore instead of Chrome's V8 engine. Designed as a drop-in replacement for Node.js, Bun is a single binary (written in **Zig**) that acts as a bundler, transpiler and package manager for JavaScript and **TypeScript** applications. Bun is currently in beta, so expect bugs or compatibility issues with a few Node.js libraries. However, it's been built from the ground up with several optimizations, including fast startup and improved server-side rendering, and we believe it's worthwhile to assess.

42. Databricks Unity Catalog

Assess

Databricks Unity Catalog is a data governance solution for assets such as files, tables or machine learning models in a **lakehouse**. Although you'll find several platforms in the enterprise data governance space, if you're already using other Databricks solutions, you should certainly assess Unity Catalog. We want to highlight that while these governance platforms usually implement a centralized solution for better consistency across workspaces and workloads, the responsibility to govern should be federated by enabling individual teams to govern their own assets.



43. Dragonfly

Assess

Dragonfly is a new in-memory data store with compatible **Redis** and Memcached APIs. It leverages the new Linux-specific **io_uring** API for I/O and implements **novel algorithms and data structures** on top of a multithreaded, shared-nothing architecture. Because of these clever choices in implementation, Dragonfly achieves impressive results in performance. Although Redis continues to be our default choice for in-memory data store solutions, we do think Dragonfly is an interesting choice to assess.

44. Edge Impulse

Assess

In previous Radars, we've written about **TinyML** — the practice of running trained models on small devices with onboard sensors to make decisions or extract features without a roundtrip to the cloud. **Edge Impulse** has made the process of collecting sensor data and then training and deploying a model as simple as possible. Edge Impulse is an end-to-end hosted platform for developing models optimized to run on small edge devices such as microcontrollers. The platform guides the developer through the entire pipeline, including the task of collecting and labeling training data. They've made it easy to get started using your mobile phone for both data collection and running the classifier while the model training and refining happens in the more powerful, cloud-hosted environment. The resulting recognition algorithms can also be optimized, compiled and uploaded to a wide range of microcontroller architectures. Although Edge Impulse is a commercial venture, the platform is free for developers and makes the entire process fun and engaging even for those who are new to machine learning. The low barrier of entry to creating a working application means that we'll be seeing more edge devices with smart decisioning built in.

45. GCP Vertex AI

Assess

GCP Vertex AI is a unified artificial intelligence platform that allows teams to build, deploy and scale machine-learning (ML) models. Vertex AI includes pretrained models, which can be used directly, fine-tuned or combined with **AutoML**, as well as infrastructure such as feature stores and pipelines for ML models. We like Vertex AI's integrated capabilities, which help to make it feel like a coherent AI platform.

46. Gradient

Assess

Gradient is a platform for building, deploying and running machine-learning applications, very similar to Google's Colab. Notebooks can be created from templates, helping you to get started with **PyTorch** or **TensorFlow** or with applications like **Stable Diffusion**. In our experience, Gradient is well-suited for GPU-intensive models, and we like that the web-based environment is persistent.



47. IAM Roles Anywhere

Assess

[IAM Roles Anywhere](#) is a new service from AWS that lets you obtain temporary security credentials in IAM for workloads such as servers, containers and applications that run outside of AWS. We find it particularly useful in hybrid cloud setups where workloads are split across AWS and non-AWS resources. Instead of creating long-lived credentials, with IAM Roles Anywhere you can now create short-lived credentials to access AWS resources using X.509 certificates. We believe this approach streamlines the access pattern across the hybrid cloud and recommend you check it out.

48. Keptn

Assess

[Keptn](#) is a control plane for delivery and operations that relies on [CloudEvents](#) for instrumentation. Like one of the techniques we mentioned in [observability for CI/CD pipelines](#), Keptn visualizes its orchestration as traces. The declarative definition of the delivery pipeline aims to separate SRE intentions from the underlying implementation, relying on other observability, pipeline and deployment tooling to respond to the appropriate events. We're particularly excited by the idea of adding service-level objective (SLO) verifications as [architectural fitness functions](#) to CI/CD pipelines: Keptn lets you define service-level indicators (SLIs) as key-value pairs, with the value representing the query to your observability infrastructure. It will then evaluate the result against the defined SLOs as a [quality gate](#). Keptn takes the same approach to automated operations, allowing a declarative definition that specifies the intent of scaling a ReplicaSet in response to a degradation of average response time, for example. Created by Dynatrace, Keptn also integrates with [Prometheus](#) and Datadog.

49. OpenMetadata

Assess

Undoubtedly, [data discoverability](#) has become a very important focal point for companies since it's an enabler for data to be shared and used efficiently by different people. We've included platforms such as [DataHub](#) and [Collibra](#) in previous editions of the Radar. However, our teams are constantly assessing options in this space and have recently shown interest in [OpenMetadata](#), a platform dedicated to metadata management by using open standards. Our teams like this open-source platform because it improves the development experience due to its simple architecture, easy deployment with a focus on automation and strong focus on data discoverability.

50. OrioleDB

Assess

[OrioleDB](#) is a new storage engine for PostgreSQL. Our teams use PostgreSQL a lot, but its storage engine was originally designed for hard drives. Although there are several options to tune for modern hardware, it can be difficult and cumbersome to achieve optimal results. OrioleDB addresses these challenges by implementing a cloud-native storage engine with explicit support for solid-state drives (SSDs) and nonvolatile random-access memory (NVRAM). To try the new engine, first install the enhancement patches to the current [table access methods](#) and then install OrioleDB as a PostgreSQL extension. We believe OrioleDB has great potential to address several [long-pending issues in PostgreSQL](#), and we encourage you to carefully assess it.

Tools

Adopt

- 51. Great Expectations
- 52. k6

Trial

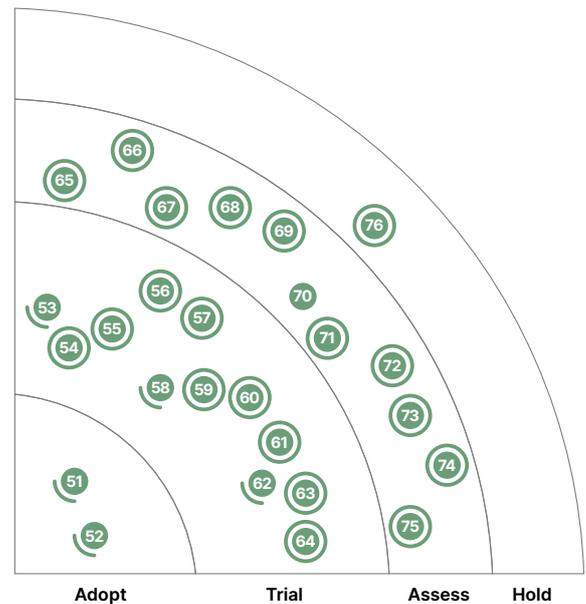
- 53. Apache Superset
- 54. AWS Backup Vault Lock
- 55. AWS Control Tower
- 56. Clumio Protect
- 57. Cruft
- 58. Excalidraw
- 59. Hadolint
- 60. Kaniko
- 61. Kusto Query Language
- 62. Spectral
- 63. Styra Declarative Authorization Service
- 64. xbar for build monitoring

Assess

- 65. Clasp
- 66. Databricks Overwatch
- 67. dbtvault
- 68. git-together
- 69. Harness Cloud Cost Management
- 70. Infracost
- 71. Karpenter
- 72. Mizu
- 73. Soda Core
- 74. Teller
- 75. Xcode Cloud

Hold

- 76. Online services for formatting or parsing code



● New ● Moved in/out ● No change



51. Great Expectations

Adopt

Great Expectations has become a sensible default for our teams in the data quality space, which is why we recommend adopting it — not only for the lack of better alternatives but also because our teams have reported great results in several client projects. Great Expectations is a framework that allows you to craft built-in controls that flag anomalies or quality issues in data pipelines. Just as unit tests run in a build pipeline, Great Expectations makes assertions during the execution of a data pipeline. We like its simplicity and ease of use — the rules stored in JSON can be modified by our data domain experts without necessarily needing data engineering skills.

52. k6

Adopt

Since we first mentioned it in the Radar, **k6** has become a go-to tool for performance testing. We continue to be fans of how easy it is to write JavaScript code for tests, but k6 also has a low-code **test builder** to make playing with the tool even easier. The documentation shows how easy it is to add performance testing to a pipeline across **multiple CI/CD tools**. Our teams find it easy to integrate **visualization tools** like **Grafana** and New Relic, which help them tune both infrastructure and applications. The developer friendliness and ecosystem make k6 a compelling option for investigating a system's behavior under heavy load.

53. Apache Superset

Trial

Apache Superset is a great business intelligence (BI) tool for data exploration and visualization to work with large data lake and data warehouse setups. It supports several **data sources** — including AWS Redshift, **BigQuery**, Azure MS SQL, **Snowflake** and **ClickHouse**. Moreover, you don't have to be a data engineer to use it; it's meant to benefit all engineers exploring data in their everyday work. For demanding use cases, we found it easy to scale Superset by deploying it in a **Kubernetes** cluster. Since we last talked about it in the Radar, Superset has graduated as an Apache product, and we've seen great success in several projects.

54. AWS Backup Vault Lock

Trial

When implementing robust, secure and reliable disaster recovery, it's necessary to ensure that backups can't be deleted or altered before their expiry, either maliciously or accidentally. Previously, with AWS Backup, these policies and guarantees had to be implemented by hand. Recently, AWS has added the Vault Lock feature to ensure backups are immutable and untamperable. **AWS Backup Vault Lock** enforces retention and deletion policies and prevents even those with administrator privileges from altering or deleting backup files. This has proved to be a valuable addition and fills a previously empty space.

55. AWS Control Tower

Trial

Multi-team account management is a challenge in AWS, especially in setup and governance; **AWS Control Tower** is an attempt to address this challenge. Our team has reported good results using it to manage accounts and access control for multiple teams in the organization through a single, centralized place.



56. Clumio Protect

Trial

We've had success with [Clumio Protect](#) for backing up AWS data, particularly S3. A commercial SaaS solution, Clumio Protect can also back up a range of other AWS services and stores the data offline where it is not accessible through the internet. Our teams responsible for handling data protection and recovery at massive scale found that Clumio Protect is easy to set up and maintain and far outperforms the native AWS Backup service when S3 buckets are particularly big.

57. Cruft

Trial

We've been talking about [tailored service templates](#) ever since we first identified [microservices](#) as a thing. If an organization sets out to create a collection of small services that can be developed, built, deployed and operated independently but consistently, it makes sense to give teams a solid starting point that aligns to the standard. However, one of the enduring problems with that approach is that as the template evolves over time in response to changing technical and business requirements, projects based on older versions of the template fall out of date. Retrofitting template improvements into an established project becomes a major pain. [Cruft](#) attempts to address this problem by providing tools to identify and patch differences between a local project and the current head of a master template repository. It combines the [Cookiecutter](#) templating engine with git hashes to identify and apply changes to the templates. Think of it as a package manager for a project boilerplate. Keeping templates up-to-date is a notoriously difficult and long-standing problem, so to us the solution Cruft provides sounds almost too good to be true. Based on early feedback from our team, however, Cruft actually works and makes life easier for service builders and maintainers. We're anxious to see how it performs over the long term, but for now it's worth taking a look at this potentially useful tool.

58. Excalidraw

Trial

We continue to hear enthusiastic reports about [Excalidraw](#) from our teams, but our previous caveat about security remains in place. Excalidraw is a simple yet powerful online drawing tool. Sometimes teams just need a quick picture instead of a formal diagram; for remote teams, Excalidraw provides a quick way to create and share diagrams. Our teams also like the "lo-fi" look of the diagrams it can produce, which is reminiscent of the whiteboard diagrams they would have produced when co-located. Regarding security, at the time of writing, anyone who has the link can see your diagrams; note, though, that the paid version of Excalidraw provides further authentication and options to run a server locally do exist.

59. Hadolint

Trial

We like spreading the word about linting tools that actually help you find issues rather than just shortcut style disputes in the team. [Hadolint](#) is one of those tools — it helps find common issues in Dockerfiles. We find it to be fast, [accurate](#) and with good documentation. It explains both how to fix an issue and why it's an issue in the first place, thus nudging Dockerfile authors toward good practices. Incidentally, Hadolint is built on top of [ShellCheck](#), which we recommend in its own right for checking your shell scripts.



60. Kaniko

Trial

Most of today's CI/CD pipeline tools and platforms are built on containers as runtimes. Many of our teams are using [Kaniko](#) to build container images from within those container-based pipelines. This comes as part of a trend away from [Docker](#) as the de facto standard for container runtimes. With Kaniko, you can build your images without using a Docker daemon. This helps avoid the security issue of Docker's "privileged" mode, which would be necessary for any "Docker-in-Docker" activity. Moreover, you don't have to assume that your pipeline has access to a Docker daemon in the first place, which cannot be taken for granted anymore and often requires extra configuration.

61. Kusto Query Language

Trial

As data work becomes more common, we continue to see tools that try to enhance the SQL language; [Kusto Query Language](#) (KQL) is one of them. KQL was created by Azure, and it brings modularity, encapsulation, composability, reusability, extensibility and dynamism to relational querying. Our teams quite like its interactivity: you can pipe a query to the render operator and see a chart instantly. You can also combine these charts into dashboards and get insights from logs to execs in minutes. Although the KQL language is currently limited to the [Azure Data Explorer](#), we anticipate the move to enhance SQL to achieve better data operability will not stop.

62. Spectral

Trial

[Spectral](#) is a JSON/YAML linter with an emphasis on OpenAPI and AsyncAPI specifications. It ships with a comprehensive set of out-of-the-box rules for these specs that can save developers headaches when designing and implementing APIs or event-driven collaboration. These rules check for proper API parameter specifications or the existence of a license statement in the spec, among other things. The [CLI](#) makes it easy to incorporate Spectral into both local development and CI/CD pipelines, and the [JavaScript API](#) supports more advanced use cases. The [GitHub site](#) links to publicly available real-world rule sets from companies like Adidas, giving teams a head start on adopting their own linting rules.

63. Styra Declarative Authorization Service

Trial

[Styra Declarative Authorization Service](#) (DAS) is a governance and automation tool for managing [Open Policy Agent \(OPA\)](#) at scale. Built by the creators of OPA, the tool allows us to deploy policies across "systems," including [Kubernetes](#) clusters, infrastructure code repositories, namespaces and more. Most importantly, it allows for real-time analysis of decisions made by an OPA agent, along with replayability for debugging and investigating what-if scenarios for policy changes. It also comes with an audit log that can help security teams with historical reporting.



64. xbar for build monitoring

Trial

On remote teams, we sorely lack having a **dedicated build monitor** in the room; unfortunately, newer continuous integration (CI) tools lack support for the old **CCTray** format. The result is that broken builds aren't always picked up as quickly as we'd like. To solve this problem, many of our teams have started using **xbar** for build monitoring. With xbar, one can execute a script to poll build status, displaying it on the menu bar. It can be further scripted to track other team metrics such as pending credential expiries or how far the production release lags behind the user acceptance testing (UAT) release. Of course, xbar is more general purpose, but it solves an immediate and emergent problem caused by remote working. **Rumps**, among other tools, can solve the same problem.

65. Clasp

Assess

Unfortunately, a big part of the world still runs on spreadsheets and will continue to do so. They're the ultimate tool to let anyone build those small custom tools tailored to their exact needs. However, when you want to enhance them with a level of logic that requires "real" code, the low-code nature of spreadsheets can then become a constraint. If you're with a company that, like Thoughtworks, uses Google's G-Suite, **Clasp** enables you to apply at least some **Continuous Delivery** practices to Apps Script code. You can write the code outside of the Apps Script project, which creates options for testing, source control and build pipelines; it even lets you use **TypeScript**. Clasp has been around for a while, and you shouldn't expect a programming environment with all of the usual comforts, but it can greatly improve the experience of using Apps Script.

66. Databricks Overwatch

Assess

Databricks Overwatch is a Databricks Labs project that enables teams to analyze various operational metrics of Databricks workloads around cost, governance and performance with support to run what-if experiments. It's essentially a set of data pipelines that populate tables in Databricks, which can then be analyzed using tools like notebooks. Overwatch is very much a power tool; however, it's still in its early stages and it may take some effort to set it up — our use of it required Databricks solution architects to help set it up and populate a price reference table for cost calculations — but we expect adoption to get easier over time. The level of analysis made possible by Overwatch is deeper than what is allowed by cloud providers' cost analysis tools. For example, we were able to analyze the cost of job failures — recognizing that failing fast saves money compared to jobs that only fail near the final step — and break down the cost by various groupings (workspace, cluster, job, notebook, team). We also appreciated the improved operational visibility, as we could easily audit access controls around cluster configurations and analyze operational metrics like finding the longest running notebook or largest read/write volume. Overwatch can analyze historical data, but its real-time mode allows for alerting which helps you to add appropriate controls to your Databricks workloads.



67. dbtvault

Assess

Data Vault 2.0 is a data modeling methodology and design pattern intended to improve the flexibility of data warehouses compared to other popular modeling approaches. Data Vault 2.0 can be applied to any data store such as **Snowflake** or **Databricks**. When implementing Data Vault warehouses, we've found the **dbtvault** package for **dbt** to be a helpful tool. dbtvault provides a set of **jinja** templates that generate and execute the ETL scripts necessary to populate a Data Vault warehouse. Although dbtvault has some rough edges — it lacks support for enforcing implied uniqueness or performing incremental loads — overall, it fills a niche and requires minimal configuration to get started.

68. git-together

Assess

We're always looking for ways to remove small frictions from pair programming, which is why we're excited by **git-together**, a tool written in Rust that simplifies git commit attribution during pairing. By aliasing `git-together` as `git`, the tool allows you to add extensions to `git config` that capture committer information, aliasing each committer by their initials. Changing pairs (or switching to soloing or mob programming) requires you to run `git with`, followed by the initials of the pair (for example: `git with bb cc`), allowing you to resume your regular git workflow afterward. Every time you commit, `git-together` will rotate through the pair as the official author that git stores, and it will automatically add any other authors to the bottom of the commit message. The configuration can be checked in with the repo, allowing `git-together` to work automatically after cloning a repo.

69. Harness Cloud Cost Management

Assess

Harness Cloud Cost Management is a commercial tool that works across all three of the major cloud providers and their managed **Kubernetes** clusters to help visualize and manage cloud costs. The product calculates a cost efficiency score by looking at idle resources as well as resources not allocated to any workload and uses historical trends to help optimize resource allocation. The dashboards highlight cost spikes and allow a user to register unexpected anomalies, which are then fed into their reinforcement learning algorithm around anomaly detection. Cloud Cost Management can recommend adjustments to limits for memory and CPU usage, with options to optimize for either cost or performance. "Perspectives" allows you to group costs based on organizationally defined filters (which could correspond to business units, teams or products) and automate report distribution to bring visibility into cloud spend. We believe Cloud Cost Management offers a compelling feature set to help organizations mature their FinOps practices.

70. Infracost

Assess

We continue to see organizations move to the cloud without properly understanding how they will track ongoing spend. We previously blipped **run cost as architecture fitness function**, and **Infracost** is a tool that aims to make these cloud cost trade-offs visible in Terraform pull requests. It's open-source software and available for macOS, Linux, Windows and Docker and supports pricing for AWS, GCP and Microsoft Azure out of the box. It also provides a public API that can be queried for current cost data. We remain excited by its potential, especially when it comes to gaining better cost visibility in the IDE.



71. Karpenter

Assess

One of the fundamental capabilities of [Kubernetes](#) is its ability to automatically launch new pods when additional capacity is needed and shut them down when loads decrease. This horizontal autoscaling is a useful feature, but it can only work if the nodes needed to host the pods already exist. While [Cluster Autoscaler](#) can do some rudimentary cluster expansion triggered by pod failures, it has limited flexibility; [Karpenter](#), however, is an open-source [Kubernetes Operator](#) autoscaler with more smarts built in: it analyzes the current workloads and the pod scheduling constraints to automatically select an appropriate instance type and then start or stop it as needed. Karpenter is an operator in the spirit of tools like [Crossplane](#) that can provision cloud resources outside the cluster. Karpenter is an attractive companion to the autoscaling services cloud vendors provide natively with their managed Kubernetes clusters. For example, AWS now supports Karpenter as a first-class alternative in their EKS Cluster Autoscaler service.

72. Mizu

Assess

[Mizu](#) is an API traffic viewer for [Kubernetes](#). Unlike other tools, Mizu does not require instrumentation or code changes. It runs as a [DaemonSet](#) to inject a container at the node level in your Kubernetes cluster and performs tcpdump-like operations. We find it useful as a debugging tool, as it can observe all API communications across multiple protocols (REST, gRPC, [Kafka](#), AMQP and [Redis](#)) in real time.

73. Soda Core

Assess

[Soda Core](#) is an open-source data quality and observability tool. We talked about [Great Expectations](#) previously in the Radar, and Soda Core is an alternative with a key difference — you express the data validations in a DSL called [SodaCL](#) (previously called [Soda SQL](#)) as opposed to Python functions. Once the validations are written, it can be executed as part of a [data pipeline](#) or [scheduled to run programmatically](#). As we become increasingly data-driven, it's critical to maintain data quality, and we encourage you to assess Soda Core.

74. Teller

Assess

[Teller](#) is an open-source universal secret manager for developers that ensures the correct environment variables are set when starting an application. However, it's not a vault itself — it's a CLI tool that connects to a variety of sources, ranging from cloud secrets providers to third-party solutions like [HashiCorp Vault](#) to local environment files. Teller has additional functionality to scan for vault-kept secrets in your code, to redact secrets from logs, to detect drift between secrets providers and to sync between them. Given the sensitivity of accessing secrets, we can't emphasize enough the need to secure the supply chain for open-source dependencies, but we appreciate how easy the CLI is to use in local development environments, CI/CD pipelines and deployment automation.



75. Xcode Cloud

Assess

Xcode Cloud is a CI/CD tool that is built into Xcode and used to build, test and deploy Apple apps. It provides an integrated experience with familiar tools for Apple developers like Xcode, App Store Connect and TestFlight. Based on our team's experience, it does a good job of simplifying the pipeline configuration and provisioning profiles and certificates. This tool is quite fresh and most of our mobile development teams are still using the more mature **Bitrise**. Still, we think it's worth assessing and tracking its progress.

76. Online services for formatting or parsing code

Hold

We previously called out **production data in test environments** and now want to highlight another common practice that needs to be approached with care or even stopped entirely: online services for formatting or parsing code. There are many useful sites for formatting or parsing formats such as JSON and YAML, as well as sites that assess code tutorials or produce online code metrics. Great care is needed when using these. Pasting a block of JavaScript, JSON or similar into an unknown website can easily create security and privacy issues and might unknowingly export personal data into a different jurisdiction. These sites should never be used with production data and should be approached with caution in all other circumstances.

Languages and Frameworks



Adopt

- 77. io-ts
- 78. Kotest
- 79. NestJS
- 80. React Query
- 81. Swift Package Manager
- 82. Yjs

Trial

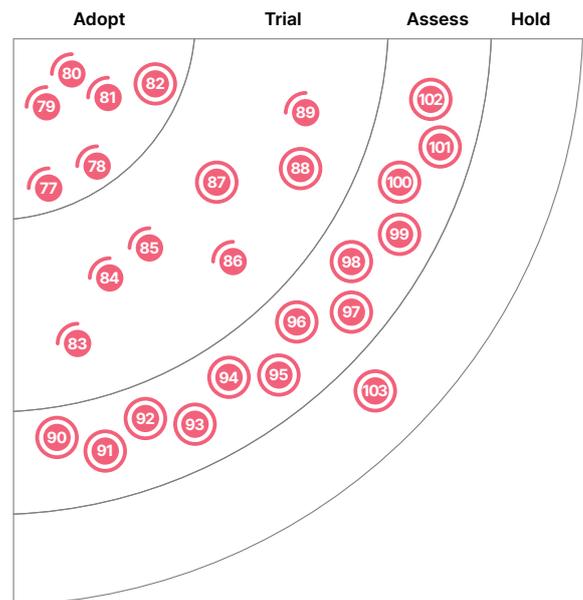
- 83. Azure Bicep
- 84. Camunda
- 85. Gradle Kotlin DSL
- 86. Jetpack Media3
- 87. Ladle
- 88. Moshi
- 89. Svelte

Assess

- 90. Aleph.js
- 91. Astro
- 92. BentoML
- 93. Carbon Aware SDK
- 94. Cloudscape
- 95. Connect
- 96. Cross device SDK
- 97. Cypress Component Testing
- 98. JobRunr
- 99. Million
- 100. Soketi
- 101. Stable Diffusion
- 102. Synthetic Data Vault

Hold

- 103. Carbon



● New ● Moved in/out ● No change



77. io-ts

Adopt

Our teams developing in [TypeScript](#) are finding [io-ts](#) invaluable, especially when interacting with APIs that ultimately result in the creation of objects with specific types. When working with TypeScript, getting data into the bounds of the type system (i.e., from the aforementioned APIs) can lead to run-time errors that can be hard to find and debug. io-ts bridges the gap between compile-time type checking and run-time consumption of external data by providing encode and decode functions. Given the experiences of our teams and the elegance of its approach, we think io-ts is worth adopting.

78. Kotest

Adopt

[Kotest](#) (previously KotlinTest) is a stand-alone testing tool for the [Kotlin](#) ecosystem that is widely used among our teams across various Kotlin implementations — native, JVM or JavaScript. Its key advantages are that it offers a variety of testing styles in order to structure test suites and that it comes with a comprehensive set of matchers, which allow for expressive tests in an elegant internal DSL. In addition to its support for [property-based testing](#), our teams like the solid IntelliJ plugin and the support community. Many of our developers consider it their first choice and recommend those who are still using JUnit in Kotlin consider switching to Kotest.

79. NestJS

Adopt

In the past, we've cautioned about [Node overload](#), and we're still cautious about the reasons to choose it. However, in scenarios where Node.js is required to build back-end applications, our teams are reporting that [NestJS](#) is a suitable option to enable developers to create testable, scalable, loosely coupled and easily maintainable applications in enterprises. NestJS is a [TypeScript](#)-first framework that makes the development of Node.js applications safer and less error-prone. NestJS is opinionated and comes with SOLID principles and an [Angular](#)-inspired architecture out of the box.

80. React Query

Adopt

[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. It works hand-in-hand with existing async data-fetching libraries like [axios](#), [Fetch](#) and [GraphQL](#) since they are built on promises. 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](#), also help developers new to the framework understand how it works. For React Native, you can use a [third-party developer tools plugin](#) utilizing [Flipper](#). In our experience, version 3 of React Query brought the stability needed to be used in production with our clients.



81. Swift Package Manager

Adopt

When introduced in 2014, Swift didn't come with a package manager. Later, [Swift Package Manager](#) was created as an official Apple open-source project, and this solution has continued to develop and mature. Our teams rely increasingly on SwiftPM because most packages can be included through it and the processes for both creators and consumers of packages have been streamlined. In the previous Radar, we recommended trialing, but we now believe it makes sense to select it as the default when starting new projects. For existing projects using tools like CocoaPods or [Carthage](#), it might be worth a quick experiment to gauge the level of effort to migrate and to check whether all dependencies are available.

82. Yjs

Adopt

Conflict-free replicated data type (CRDT) algorithms are proven to be able to automatically distribute and merge changes among peers without conflicts. But in practice, even for small enough data, these algorithms usually require a significant amount of memory to trace all the changes made by different peers, thus making them impractical. [Yjs](#) is a carefully optimized CRDT implementation that keeps memory consumption at a reasonable level for large data sets and millions of modifications. It also provides bindings for popular text editors, which greatly reduce the cost of building collaborative tools.

83. Azure Bicep

Trial

For those who prefer a more natural language than JSON for infrastructure code, [Azure Bicep](#) is a domain-specific language (DSL) that uses a declarative syntax and supports reusable parameterized templates for modular resource definitions. A [Visual Studio Code extension](#) provides instant type safety, intellisense and syntax checking, while the compiler allows bidirectional transpilation to and from Azure Resource Manager (ARM) templates. Bicep's resource-oriented DSL and native integration with the Azure ecosystem make it a compelling choice for Azure infrastructure development.

84. Camunda

Trial

Since we last mentioned [Camunda](#), we've seen many of our teams and clients use the platform, making it one of our preferred workflow engines in cases where a workflow engine is a good fit for the domain. Camunda offers workflow and decision engines that can be integrated as a library in your Java code. This makes it easy to test, version and refactor workflows, alleviating some of the downsides of other more low-code workflow engines. We've even seen Camunda used in environments with high performance requirements. Teams also like how easy it is to integrate with [Spring Boot](#) and its nice user interface.

85. Gradle Kotlin DSL

Trial

Previously, we blipped about the Android Gradle plugin Kotlin DSL, or Gradle Kotlin DSL, which added support for [Kotlin](#) Script as an alternative to [Groovy](#) for Android projects using [Gradle](#) build scripts. The goal of replacing Groovy with Kotlin is to provide better support for refactoring and simpler editing



in IDEs and, ultimately, to produce code that is easier to read and maintain. For teams already using Kotlin, it also means working on the build in a familiar language. We now suggest trialing Kotlin DSL as an alternative language to Groovy for Gradle projects in general, especially if you have large or complex Gradle build scripts. Many IDEs now include support for the migration of existing projects. Some caveats remain, and we suggest checking the [documentation](#) for the most up-to-date details, including the prerequisites. We had a team with an at least seven-year-old, 450-line build script migrate successfully within a few days.

86. Jetpack Media3

Trial

Android had several media APIs: Jetpack Media, also known as MediaCompat, Jetpack Media2 and ExoPlayer. Unfortunately, these libraries were developed independently, with different goals but overlapping functionality. Android developers not only had to choose which library to use, they also had to contend with writing adaptors or other connecting code when features from multiple APIs were needed. [Jetpack Media3](#) is an API that takes common areas of functionality from the existing APIs — including UI, playback and media session handling — and combines them into a merged and refined API. The player interface from ExoPlayer has also been updated, enhanced and streamlined to act as the common player interface for Media3. After an early access phase, Media3 is now in beta. Although its first release is forthcoming, we've already had positive experiences using it in apps.

87. Ladle

Trial

As [Storybook](#) grew in popularity, it became more and more of a behemoth. If all you really care about is isolating and testing your React UI components, then [Ladle](#) is the alternative. Ladle supports most of the Storybook API (MDX files are not supported yet) and can be used as a drop-in replacement. It is lightweight and has better integration with [Vite](#). It also provides simple and clean APIs that can be easily integrated with other testing frameworks.

88. Moshi

Trial

We're hearing that our [Kotlin](#)-based teams are seeking alternatives to Java frameworks such as GSON when handling JSON. Although it's been around for some time, [Moshi](#) has now emerged as a preferred framework for many of these teams. It's easy to migrate from GSON and Moshi provides native support for Kotlin non-nullable types and default parameters. Moshi makes working with JSON faster and easier. If you're currently using a Java framework from within Kotlin to handle JSON, we recommend giving Moshi a try.

89. Svelte

Trial

Among web component frameworks, [Svelte](#) stands out by moving reactivity out of the browser and into the compiler. Instead of optimizing DOM updates by using a virtual DOM and browser optimization tricks, Svelte compiles your code into vanilla framework-less JavaScript code that surgically updates the DOM directly. In addition to the run-time performance benefits, this also allows Svelte to optimize



the amount of code the browser has to download without sacrificing features for developers; moreover, it's proven to be performant and battery-friendly for mobile web applications as less code has to execute in the browser itself. Performance benefits aside, our teams have appreciated its friendly learning curve and the maintenance benefits that come from **writing less code**. Svelte itself is only the component framework, but **SvelteKit** adds features to build full web applications.

90. Aleph.js

Assess

There is certainly no shortage of frameworks to build web applications in JavaScript/**TypeScript**. We've featured many of them in the Radar, but what sets **Aleph.js** apart in this crowded field is that it's built to run on **Deno**, the new server-side run time created by the original developer of **Node**. This puts Aleph.js on a modern foundation that addresses several shortcomings and problems with Node. Aleph.js is still new — it's approaching the 1.0 release at the time of writing — but it already offers a solid developer experience, including hot module replacement. With Deno now way past its **1.0 release**, this is a modern choice for projects that can take the risk.

91. Astro

Assess

It's hard to believe, but in 2022, the developer community continues to pump out interesting new frameworks for building web applications. **Astro** is a recent, open-source, multi-page application framework that renders HTML on the server and minimizes the amount of JavaScript sent over the wire. Astro seems particularly well-suited to content-oriented websites that pull from many different sources. We like the fact that although Astro encourages sending only HTML, it still supports — when appropriate — select active components written in the front-end JavaScript framework of your choice. It does this through its **island architecture**. Islands are regions of interactivity within a single page where the necessary JavaScript is downloaded only when needed. Astro is relatively new but seems to support a growing ecosystem of developers and code. It's one to watch as it develops.

92. BentoML

Assess

BentoML is a python-first framework for serving machine-learning models in production at scale. The models it provides are agnostic of their environment; all model artifacts, source code and dependencies are encapsulated in a self-contained format called Bento. It's like having your model "as a service." Think of BentoML as the **Docker** for ML models: It generates VM images with pre-programmed APIs ready for deployment and includes features that make it easy to test these images. BentoML can help speed up the initial development effort by easing the start of projects which is why we included it in Assess.

93. Carbon Aware SDK

Assess

When looking at reducing the carbon footprint of an application — the carbon dioxide emissions caused indirectly by running the software — attention is usually directed at making the software more efficient. The thinking is clear: more efficient software needs less electricity and fewer servers,



reducing the emissions from electricity generation and manufacturing of the servers. An additional strategy is to make the application *carbon aware*. This is because the same workload does not always have the same carbon footprint. For example, when run in a data center in a cooler climate, less power for air conditioning is needed; or, when run at a time when more renewable energy is available (more sunshine, stronger winds), less electricity from carbon-based sources is required. With the [Carbon Aware SDK](#), software engineers can query data sources to discover less carbon-intensive options for a given workload and then move it to a different location or run it at a different time. This makes sense for large workloads that are neither time nor latency sensitive, such as training a machine-learning model. Although the SDK and available data sources are not very comprehensive yet, we believe it's time to start looking at how we can make our systems carbon aware.

94. Cloudscape

Assess

[Cloudscape](#) is an open-source design system that not only has a rich set of components but also 35 interaction and content representation patterns. In addition, it uses [design tokens](#) for theming and provides element wrappers for all components, which greatly simplifies unit testing. This makes it stand out from other design systems out there.

95. Connect

Assess

[Connect](#) is a family of libraries for building browser- and gRPC-compatible HTTP APIs. Similar to gRPC, you write Protocol Buffer schema and implement the application logic, and Connect generates code to handle marshaling, routing, compression and content type negotiation. However, Connect tries to improve on gRPC in several ways. This includes native support for gRPC-Web without a translating proxy; interoperability with third-party routers or middleware, because [connect-go](#) is built on top of net/http (unlike grpc-go); and fully generated type-safe clients with the ergonomics of hand-crafted code. We mostly prefer REST and are not a big fan of the RPC approach to building APIs. That said, Connect does seem to address some of our concerns with RPCs, and we encourage you to assess it.

96. Cross device SDK

Assess

As smart devices continue to embed themselves in our lives, we are starting to see new use cases emerge that span across multiple devices. The classic example is a text we start reading on a phone but prefer to finish on a tablet. Other examples include plotting a cycling route on a laptop and then transferring the data to a bike computer for easier navigation or using a mobile phone as a webcam. Such use cases require very specific kinds of features, like the discovery of nearby devices, secure communication and multi-device sessions. Apple started introducing such features a while ago to its own SDKs, and now Google has released the first preview of its [Cross device SDK](#). Although the preview has several limitations — for example, only phones and tablets are supported and only two devices at a time — the technology is exciting and can be utilized as it is rolled out over time.



97. Cypress Component Testing

Assess

Cypress Component Testing provides a testable component workbench to quickly build and test UI components. You can write component visual regression tests with the same API that you write end-to-end (E2E) UI tests. Although still in beta, component testing will be the most important feature in **Cypress** 10.

98. JobRunr

Assess

JobRunr is a library for background job processing in Java and an alternative to the Quartz scheduler. Our teams have enjoyed using JobRunr's built-in dashboard, which is easy to use and allows the monitoring and scheduling of background tasks. JobRunr is open source and free for commercial use; for features such as job migration and recovery, however, you need to get a paid license.

99. Million

Assess

Million is a new virtual DOM JavaScript library. Similar to **Svelte**, it leverages the compiler, **Vite**, to create small JavaScript bundles with exceptional rendering performance. The Million library ships as a single NPM package with several modules — including **router**, **jsx-runtime** and a module for **React compatibility** to create single-page applications. Although **React** popularized the virtual DOM a decade ago, it's fascinating to see new innovations in this space.

100. Soketi

Assess

Soketi is an open-source WebSockets server. If your application is compatible with the **Pusher** protocol, you can plug Soketi in directly as it fully implements the **Pusher Protocol v7**. We find the **beta support** for Cloudflare Workers particularly interesting because it opens the door to using WebSockets at the network edge.

101. Stable Diffusion

Assess

OpenAI's **DALL-E** caught everyone's attention with its ability to create **images from text prompts**. Now, **Stable Diffusion** offers the same capability but, critically, it's open source. Anyone with access to a powerful graphics card can experiment with the model, and anyone with **sufficient** compute resources can recreate the model themselves. The results are **astounding** but also raise significant questions. For example, the model is trained on image-text pairs obtained via a **broad scrape of the internet** and therefore will reflect societal biases, which means it could possibly produce content that is illegal, upsetting, or at the very least undesirable. Stable Diffusion now includes an AI-based **safety classifier**; however, given its open-source nature, people can disable the classifier. Finally, artists have noted that with the right prompts the model is adept at mimicking their artistic style.

This raises questions about the ethical and legal implications of an AI capable of imitating an artist.



102. Synthetic Data Vault

Assess

[Synthetic Data Vault \(SDV\)](#) is a synthetic data generation ecosystem of libraries that can learn the distribution of a data set to generate synthetic data with the same format and statistical properties as the source. In the past, we talked about the downsides of using [production data in test environments](#). However, the nuances of data distribution in production can hardly be replicated manually, resulting in defects and surprises. We believe SDV and similar tools can address this gap by generating production-like data for [single-table](#), [complex multi-table](#) and [multivariate timeseries](#) data. Although SDV isn't new, we quite like it and decided to highlight it.

103. Carbon

Hold

We're seeing some interest in the [Carbon](#) programming language. That doesn't come as a surprise: it has Google's backing and is presented as a natural successor to C++. In our opinion C++ can't be replaced fast enough as software engineers have shown, over the past decades, that writing safe and error-free C++ code is extremely difficult and time-consuming. While Carbon is an interesting concept with its focus on migration from C++, without a working compiler, it's clearly a long way from being usable and there are other modern programming languages that are good choices if you want to migrate from C++. It's too early to tell whether Carbon will become the natural successor to C++, but, from today's perspective, we recommend that teams look at [Rust](#) and [Go](#) rather than postponing a migration because they're waiting for Carbon to arrive.

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 12,000+ people strong across 50 offices in 18 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.

