



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	38

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.

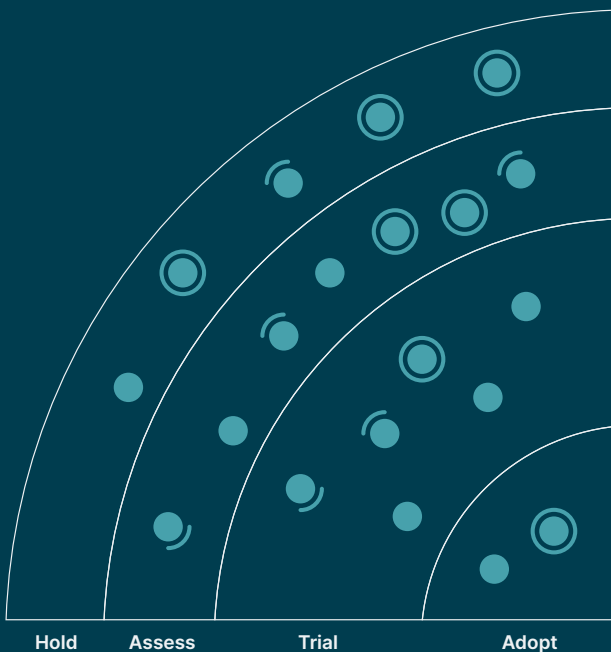
For more background on the Radar, see thoughtworks.com/radar/faq.



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 21 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 remotely in March 2023.



Rebecca Parsons (CTO)



Martin Fowler (Chief Scientist)



Bharani Subramaniam



Birgitta Böckeler



Brandon Byars



Camilla Falconi Crispim



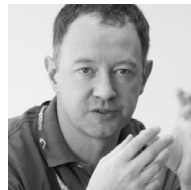
Erik Dörnenburg



Fausto de la Torre



Hao Xu



Ian Cartwright



James Lewis



Marisa Hoenig



Maya Ormaza



Mike Mason



Neal Ford



Pawan Shah



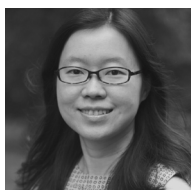
Scott Shaw



Selvakumar Natesan



Shangqi Liu



Sofia Tania



Vanya Seth

Themes

The meteoric rise of practical AI

No, this theme text wasn't written by [ChatGPT](#). Artificial intelligence has been quietly bubbling away in specialized areas for decades, and tools like [GitHub Copilot](#) have been around (and gradually seeing adoption) for a few years. However, over the last few months, tools like ChatGPT have completely reoriented everyone to what's possible and made the tools widely available. Several blips in this edition of the Radar touch on practical uses for AI for projects beyond suggesting code that requires tweaking: [AI-aided test-first development](#), using AI to help build analysis models, and many more. Similar to how spreadsheets allowed accountants to stop using adding machines to recalculate complex spreadsheets by hand, the next generation of AI will take on chores to relieve technology workers, including developers, by replacing tedious tasks that require knowledge (but not wisdom).


However, we caution against over- or inappropriate uses. Right now, the AI models are capable of generating a good first draft. But the generated content always needs to be monitored by a human who can validate, moderate and use it responsibly. If these precautions are ignored, the results can lead to reputational and security risks to organizations and users. Even some [product demos](#) caution users, "AI-generated content can contain mistakes. Make sure it's accurate and appropriate before using it."

Accessible accessibility

Accessibility has been an important consideration for organizations for many years. Recently, we've highlighted the experiences of our teams with the ever-growing set of tools and techniques that add improved accessibility to development, and several regions our teams highlighted awareness of these techniques via awareness campaigns. We've featured accessibility-related blips on continuous integration pipeline development, [design playbooks](#), [intelligent guided accessibility testing](#), [linting](#) and [unit testing](#). Growing awareness around this important topic is welcome; techniques that give more people access to functionality in improved ways can only be a good thing.

Lambda quicksand

Serverless functions — [AWS Lambdas](#) — increasingly appear in the toolboxes of architects and developers, and are used for a wide variety of useful tasks that realize the benefits of cloud-based infrastructure. However, like many useful things, solutions sometimes start suitably simple but then, from relentless gradual success, keep evolving until they reach beyond the limitations inherent in the paradigm and sink into the sand under their own weight. While we see many successful applications of serverless-style solutions, we also hear many cautionary tales from our projects, such as the [Lambda pinball antipattern](#). We also see more tools that appear to solve problems but are prone to



wide misuse. For example, tools that facilitate sharing code between Lambdas or orchestrate complex interactions might solve a common simple problem but are then at risk of recreating some terrible architecture antipatterns with new building blocks. If you need a tool to manage code sharing and independent deployment across a collection of serverless functions, then perhaps it's time to rethink the suitability of the approach. Like all technology solutions, serverless has suitable applications but many of its features include trade-offs that become more acute as the solution evolves.

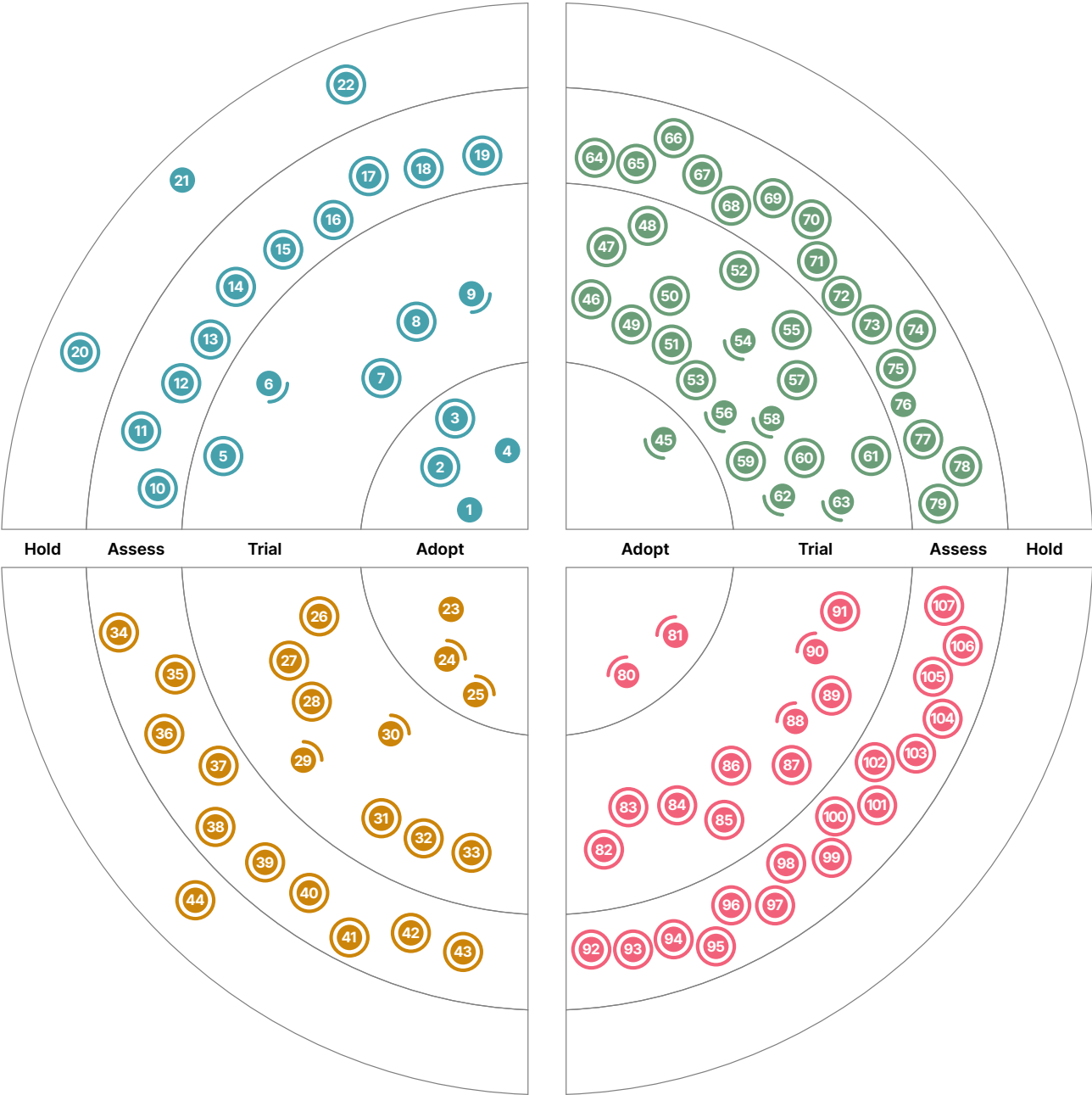
Engineering rigor meets analytics and AI

We've long viewed "building in quality" as a vital aspect of developing reliable analytics and machine learning models. Test-driven transformations, data sanity tests and data model testing strengthen the data pipelines that power analytical systems. Model validation and quality assurance are crucial in tackling biases and ensuring ethical ML systems with equitable outcomes. By integrating these practices, businesses become better positioned to leverage AI and machine learning and forge responsible, data-driven solutions that cater to a diverse user base. The corresponding tooling ecosystem has continued to grow and mature. For example, Soda Core, a data quality tool, allows the validation of data as it arrives in the system and automated monitoring checks for anomalies. Deepchecks allows for the intersection of continuous integration and model validation, an important step in incorporating good engineering practices in analytics settings. Giskard allows for quality assurance for AI models, allowing designers to detect bias and other negative facets of models, which aligns with our encouragement to tread ethical waters carefully when developing solutions with AI. We view these maturing tools as further evidence of the mainstreaming of analytics and machine learning and its integration with good engineering practices.

To declare or program?

A seemingly perpetual discussion that happens at every Radar gathering gained particular prominence this time — for a given task, should you write a declarative specification using JSON, YAML or something domain-specific like HCL, or should you write code in a general-purpose programming language? For example, we discussed the differences between Terraform Cloud Operator versus Crossplane, whether to use the AWS CDK or not and using Dagger for programming a deployment pipeline among other cases. Declarative specifications, while often easier to read and write, offer limited abstractions which leads to repetitive code. Proper programming languages can use abstractions to avoid duplication, but these abstractions can make the code considerably harder to follow, especially when the abstractions are layered after years of changes. In our experience, there's no universal answer to the question posed above. Teams should consider both approaches, and when a solution proves difficult to implement cleanly in one language type, they should reevaluate the other type. It can even make sense to split concerns and implement them with different languages.

The Radar



New
 Moved in/out
 No change

The Radar

Techniques

Adopt

1. Applying product management to internal platforms
2. CI/CD infrastructure as a service
3. Dependency pruning
4. Run cost as architecture fitness function

Trial

5. Accessibility annotations in designs
6. Bounded low-code platforms
7. Demo frontends for API-only products
8. Lakehouse architecture
9. Verifiable credentials

Assess

10. Accessibility-aware component test design
11. AI-aided test-first development
12. Domain-specific LLMs
13. Intelligent guided accessibility tests
14. Logseq as team knowledge base
15. Prompt engineering
16. Reachability analysis when testing infrastructure
17. Self-hosted LLMs
18. Tracking health over debt
19. Zero trust security for CI/CD

Hold

20. Casual management of webhooks
21. Lambda pinball
22. Planning for full utilization

Platforms

Adopt

23. Contentful
24. GitHub Actions
25. K3s

Trial

26. Apache Hudi
27. Arm in the cloud
28. Ax
29. DuckDB
30. Feature Store
31. RudderStack
32. Strapi
33. TypeDB

Assess

34. Autoware
35. Cozo
36. Dapr
37. Immuta
38. Matter
39. Modal
40. Neon
41. OpenLineage
42. Passkeys
43. Spin

Hold

44. Denodo as primary data transformation tool

The Radar

Tools

Adopt

45. DVC

Trial

- 46. Akeyless
- 47. Apicurio Registry
- 48. EventCatalog
- 49. FOSSA
- 50. Gitleaks
- 51. Helmfile
- 52. IBM Equal Access Accessibility Checker
- 53. Ktlint
- 54. Kubeflow
- 55. Mend SCA
- 56. Mozilla SOPS
- 57. Ruff
- 58. Soda Core
- 59. Steampipe
- 60. Terraform Cloud Operator
- 61. TruffleHog
- 62. Typesense
- 63. Vite

Assess

- 64. axe Linter
- 65. ChatGPT
- 66. DataFusion
- 67. Deepchecks
- 68. Design token translation tools
- 69. Devbox
- 70. Evidently
- 71. Giskard
- 72. GitHub Copilot
- 73. iamlive
- 74. Kepler
- 75. Kubernetes External Secrets Operator
- 76. Kubeshark
- 77. Obsidian
- 78. Ory Kratos
- 79. Philips's self-hosted GitHub runner

Hold

—

Languages and Frameworks

Adopt

- 80. Gradle Kotlin DSL
- 81. PyTorch

Trial

- 82. dbt-unit-testing
- 83. Jetpack CameraViewfinder
- 84. Jetpack DataStore
- 85. Mikro ORM
- 86. Per-app language preferences
- 87. Quarto
- 88. River
- 89. Stencil
- 90. Synthetic Data Vault
- 91. Vitest

Assess

- 92. .NET 7 Native AOT
- 93. .NET MAUI
- 94. dbt-expectations
- 95. Directus
- 96. Ferrocene
- 97. Flutter for embedded
- 98. Fugue
- 99. Galacean Engine
- 100. LangChain
- 101. mljar-supervised
- 102. nanoGPT
- 103. pandera
- 104. Qwik
- 105. SolidJS
- 106. Turborepo
- 107. WebXR Device API

Hold

—

Techniques

Adopt

1. Applying product management to internal platforms
2. CI/CD infrastructure as a service
3. Dependency pruning
4. Run cost as architecture fitness function

Trial

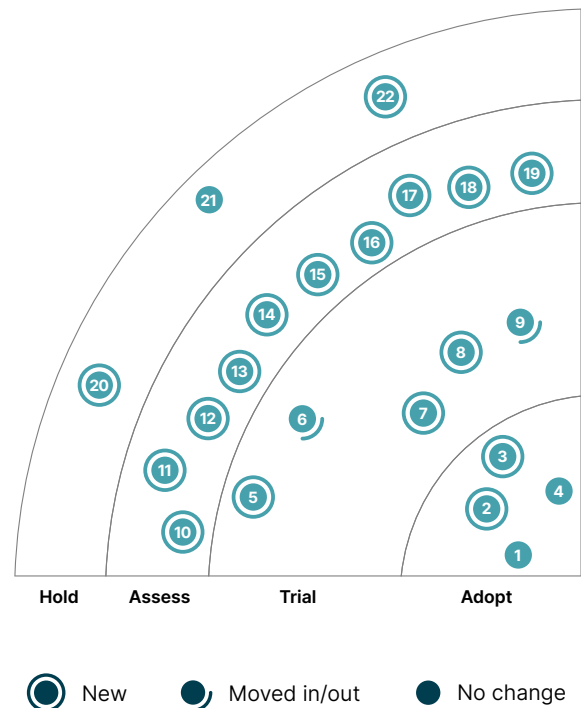
5. Accessibility annotations in designs
6. Bounded low-code platforms
7. Demo frontends for API-only products
8. Lakehouse architecture
9. Verifiable credentials

Assess

10. Accessibility-aware component test design
11. AI-aided test-first development
12. Domain-specific LLMs
13. Intelligent guided accessibility tests
14. Logseq as team knowledge base
15. Prompt engineering
16. Reachability analysis when testing infrastructure
17. Self-hosted LLMs
18. Tracking health over debt
19. Zero trust security for CI/CD

Hold

20. Casual management of webhooks
21. Lambda pinball
22. Planning for full utilization



1. Applying product management to internal platforms

Adopt

We keep getting good feedback from teams applying product management to internal platforms. One key feature to remember, though: It's not just about team structure or renaming existing platform teams; it's also about applying product-centric working practices within the team. Specifically, we've received feedback that teams face challenges with this technique unless they have a product-centric mindset. This likely means additional roles, such as a product manager, alongside changes to other areas, such as requirements gathering and the measurement of success. Working this way means establishing empathy with internal consumers (the development teams) and collaborating with them on the design. Platform product managers create roadmaps and ensure the platform delivers value to the business and enhances the developer experience. We continue to see this technique as key to building internal platforms to roll out new digital solutions quickly and efficiently.

2. CI/CD infrastructure as a service

Adopt

The options for CI/CD infrastructure as a service have become so manifold and mature that the cases in which it's worth managing your entire CI infrastructure yourself are becoming very rare. Using managed services like [GitHub Actions](#), [Azure DevOps](#) or [Gitlab CI/CD](#) comes with all the common advantages (and trade-offs) of managed cloud services. You don't have to spend time, effort and hardware costs on maintenance and operations of this often complex infrastructure. Teams can take advantage of elasticity and self-service, whereas provisioning more of the right agents or getting a new plugin or feature are often a bottleneck in companies that host CI themselves. Even the use cases that require to run build and verification on your own hardware can now mostly be covered with self-hosted runners (we've written about some for [GitHub Actions](#), [actions-runner-controller](#) and the [Philips's self-hosted GitHub runner](#)). Note, however, that you won't get out-of-the-box security just because you're using a managed services; while mature services provide all the security features you need, you'll still need to use them to implement [zero trust security for your CI/CD infrastructure](#).

3. Dependency pruning

Adopt

Starter kits and templates are widely used in software projects to speed up initial setup, but they can pull in many unnecessary dependencies for a particular project. It's important to practice dependency pruning — periodically taking a hard look at these dependencies and pruning any that are not used. This helps reduce build and deploy times and decrease the project's attack surface by removing potential vulnerabilities. Although this isn't a new technique, given the increasing frequency of attacks on software supply chains, we advocate for renewed attention to it.

4. Run cost as architecture fitness function

Adopt

Automatically estimating, tracking and predicting cloud infrastructure run cost is crucial for today's organizations. The cloud providers' savvy pricing models, combined with the proliferation of pricing parameters and the dynamic nature of today's architecture, can lead to surprisingly expensive run costs. Even though this technique has been in Adopt since 2019, we want to highlight the importance of considering run cost as an architecture fitness function, especially today, due to accelerated cloud

adoption and the growing attention to FinOps practices. Many commercial platforms provide tools that can consolidate and clarify cloud costs for business leaders. Some of them are designed to show cloud run costs to finance organizations or originating business units.

However, cloud consumption decisions are usually made at the engineering level, where systems are designed. It's important that the engineers making design decisions have some way of predicting the cost impact of their architectural decisions. Some teams automate this prediction early in the development lifecycle. Tools like [Infracost](#) help teams predict cost impact when thinking about possible changes to infrastructure as code. This computation can be automated and woven into the CD pipeline. Note that cost will be impacted by architectural decisions combined with actual usage levels; to do this properly, you need good projections of expected usage levels. Early and frequent feedback on run cost can prevent it from soaring. When the predicted cost deviates from what was expected or acceptable, the team can discuss whether it's time to evolve the architecture.

5. Accessibility annotations in designs

Trial

The earlier accessibility is considered in software delivery, the easier and cheaper it is to ensure what's built works for as many people as possible. Tools that help communicate accessibility annotations in designs help teams consider important elements like document structure, semantic HTML and alternative texts from the beginning of their work. This enables them to ensure user interfaces meet global accessibility standards and address common failures that are actually fairly easy to avoid. [Figma](#) offers a range of accessibility notation plugins: [The A11y Annotation Kit](#), Twitter's [Accessibility Annotation Library](#) and the Axe toolset's [Axe for Designers](#).

6. Bounded low-code platforms

Trial

We've always been advocates of writing less code. Simplicity is one of the core values underlying our sensible defaults for software development. For example, we try not to anticipate needs and only introduce code that satisfies immediate business requirements and nothing else. One way to achieve this is to create engineering platforms that make this possible on an organizational basis.

This is also the stated aim of many low-code platforms surging in popularity right now. Platforms like [Mendix](#) or [Microsoft Power Apps](#) can expose common business processes for reuse and simplify the problems of getting new functionality deployed and in the hands of users. These platforms have made great strides in recent years with testability and support for good engineering practices. They're particularly useful for simple tasks or event-triggered apps. However, asking them to adapt to a nearly infinite range of business requirements brings complexity. Although developers might be writing less (or zero) code, they must also become experts in an all-encompassing commercial platform. We would advise businesses to consider if they need all the functionality these products bring or if they're better off pursuing bounded low-code platforms, either by developing their own [platform as an internal product](#) or by carefully constraining the use of commercial low-code products to those simple tasks at which they excel.

7. Demo frontends for API-only products

Trial

One of the big challenges in developing APIs is capturing and communicating their business value. APIs are, by their nature, technical artifacts. Whereas developers can easily comprehend JSON payloads, OpenAPI (Swagger) specs and Postman demos, business stakeholders tend to respond better to demos they can interact with. The value of the product is more clearly articulated when you can see and touch it, which is why we sometimes find it worthwhile to invest in demo frontends for API-only products. When a custom graphical UI is built alongside an API product, stakeholders can see analogies to paper forms or reports that might be more familiar to them. As the interaction model and richness of the demo UI evolves, it allows them to make more informed decisions about the direction the API product should take. Working on the UI has the added benefit of increasing developers' empathy for business users. This isn't a new technique — we've been doing this successfully when necessary as long as API products have been around. However, because this technique isn't widely known, we thought it worthwhile calling attention to it.

8. Lakehouse architecture

Trial

Lakehouse architecture is an architectural style that combines the scalability of data lakes with the reliability and performance of data warehouses. It enables organizations to store and analyze large volumes of diverse data in a single platform as opposed to having them in separate lake and warehouse tiers, using the same familiar SQL-based tools and techniques. While the term is often associated with vendors like Databricks, open alternatives such as Delta Lake, Apache Iceberg and Apache Hudi are worth considering. Lakehouse architecture can complement data mesh implementations. Autonomous data product teams can choose to leverage a Lakehouse within their data products.

9. Verifiable credentials

Trial

When we first included it in the Radar three 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. Three 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. 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 helps users to better manage and selectively disclose their own information which greatly improves data privacy protection.

Several of our teams have engaged in projects involving verifiable credentials technology in the past six months. Not surprisingly, the scenarios vary across countries and government departments. Our team has explored different combinations of decentralized identifiers, verifiable credentials and verifiable presentation on multiple projects. This is a developing field, and now that we've had more experience, we want to keep track of it in the Radar.

10. Accessibility-aware component test design

Assess

One of the many places in the software delivery process to consider accessibility requirements early on is during web component testing. Testing framework plugins like [chai-a11y-axe](#) provide assertions in their API to check for the basics. But in addition to using what testing frameworks have to offer, accessibility-aware component test design further helps to provide all the semantic elements needed by screen readers and other assistive technologies.

Firstly, instead of using test ids or classes to find and select the elements you want to validate, use a principle of identifying elements by [ARIA](#) roles or other semantic attributes that are used by assistive technologies. Some testing libraries, like [Testing Library](#), even recommend this in their documentation. Secondly, do not just test for click interactions; also consider users who cannot use a mouse or see the screen, and consider adding additional tests for the keyboard and other interactions.

11. AI-aided test-first development

Assess

Like many in the software industry, we've been exploring the rapidly evolving AI tools that can support us in writing code. We see many people feed [ChatGPT](#) with an implementation, and then ask it to generate tests for that implementation. However, because we're big believers in TDD, and we don't always want to feed an external model with our potentially sensitive implementation code, one of our experiments in this space is a technique we call [AI-aided test-first development](#). In this approach, we get ChatGPT to generate tests for us, and then a developer implements the functionality. Specifically, we first describe the tech stack and the design patterns we're using in a prompt "fragment" that is reusable across multiple use cases. Then we describe the specific feature we want to implement, including the acceptance criteria. Based on all that, we ask ChatGPT to generate an implementation plan for that feature in our architectural style and tech stack. Once we sanity check that implementation plan, we ask it to generate tests for our acceptance criteria.

This approach has worked surprisingly well for us: It required the team to come up with a concise description of their architectural style and helped junior developers and new team members code features aligned with the team's existing style. The main drawback of this approach is that even though we don't give the model our source code, we still feed it potentially sensitive information such as our tech stack and feature descriptions. Teams should ensure they're working with their legal advisors to avoid any intellectual property issues, at least until a "for business" version of these AI tools becomes available.

12. Domain-specific LLMs

Assess

We've featured large language models (LLMs) like [BERT](#) and [ERNIE](#) in the Radar before; domain-specific LLMs, however, are an emerging trend. Fine-tuning general-purpose LLMs with domain-specific data can tailor them for various tasks, including information retrieval, customer support augmentation and content creation. This practice has shown promising results in industries like [legal](#) and [finance](#), as demonstrated by [OpenNyanAI](#) for legal document analysis. With more organizations experimenting with LLMs and new models like GPT4 being released, we can expect more domain-specific use cases in the near future.

However, there are challenges and pitfalls to consider. First, LLMs can be confidently wrong, so it's essential to build mechanisms into your process to ensure the accuracy of results. Second, third-party LLMs may retain and re-share your data, posing a risk to proprietary and confidential information. Organizations should carefully review the terms of use and trustworthiness of providers or consider training and running LLMs on an infrastructure they control. As with any new technology, businesses must tread carefully, understanding the implications and risks associated with LLM adoption.

13. Intelligent guided accessibility tests

Assess

It can be a bit daunting to make a web application compliant with assistive technologies when you yourself never use them, and you feel like you don't yet know anything about directives like the [Web Content Accessibility Guidelines \(WCAG\)](#). Intelligent guided accessibility tests are one category of tools that help test if you've done the right thing without needing to be an expert on accessibility. These tools are browser extensions that scan your website, summarize how assistive technology would interpret it and then ask you a set of questions to confirm whether the structure and elements you created are as intended. We've used [axe DevTools](#), [Accessibility Insights for Web](#) or the [ARC Toolkit](#) on some of our projects.

14. Logseq as team knowledge base

Assess

Team knowledge management is a familiar concept with teams using tools such as wikis to store information and onboard new team members. Some of our teams now prefer to use [Logseq](#) as a team knowledge base. An open-source knowledge-management system, Logseq is powered by a graph database, helps users organize thoughts, notes and ideas and can be adapted for team use with Git-based storage. Logseq allows teams to build a democratic and accessible knowledge base, providing each member with a personalized learning journey and facilitating efficient onboarding. However, as with any knowledge management tool, teams will need to apply good curation and management of their knowledge base to avoid information overload or disorganization.

While similar functionality is available in tools like [Obsidian](#), the key difference lies in Logseq's focus on consumption, with paragraph-based linking enabling team members to quickly find the relevant context without having to read an entire article.

15. Prompt engineering

Assess

Prompt engineering refers to the process of designing and refining prompts for generative AI models to obtain high-quality responses from the model. This involves carefully crafting prompts that are specific, clear and relevant to the desired task or application in order to elicit useful outputs from the model. Prompt engineering aims to enhance large language model (LLM) capabilities in tasks like question answering and arithmetic reasoning or in domain-specific contexts. For software creation, you might use prompt engineering to get an LLM to write a story, an API or a test suite based on a brief conversation with a stakeholder or some notes. Developing effective prompting techniques is becoming a valuable skill in working with AI systems. There is debate over whether prompt engineering is an art or science, and potential security risks, such as "prompt injection attacks," should be considered.

16. Reachability analysis when testing infrastructure

Assess

When deploying infrastructure as code, we've noticed that a lot of time can be spent diagnosing and repairing production issues that result from systems being unable to communicate with one another. Because the network topology between them can be complex, the entire route may not be traversable even if individual ports and endpoints have been configured correctly. Infrastructure testing practices usually include verifying the right ports are open or closed or that an endpoint can be accessed, but we've only recently begun doing reachability analysis when testing infrastructure. The analysis generally involves more than simple yes/no determinations. For example, a tool might traverse and report on multiple routes through transit gateways. This technique is supported by tools across all the major cloud providers. Azure has a service called [Network Watcher](#) that can be scripted in automated tests and GCP supports [Connectivity Tests](#). Now, in AWS, you can test reachability [across accounts](#) in the same organization.

17. Self-hosted LLMs

Assess

Large language models (LLMs) generally require significant GPU infrastructure to operate. We're now starting to see ports, like [llama.cpp](#), that make it possible to run LLMs on different hardware — including Raspberry Pis, laptops and commodity servers. As such, self-hosted LLMs are now a reality, with open-source examples including [GPT-J](#), [GPT-JT](#) and [LLaMA](#). This approach has several benefits, offering better control in fine-tuning for a specific use case, improved security and privacy as well as offline access. However, you should carefully assess the capability within the organization and the cost of running such LLMs before making the decision to self-host.

18. Tracking health over debt

Assess

Tracking technical debt is a perennial topic in software delivery organizations. What is [technical debt](#) and what is not? How do you prioritize it? And most importantly, how do you express the value of paying it off to your internal stakeholders? Following the Agile Manifesto's manner of reasoning — “while there is value in the item on the right, we value the item on the left more” — we like the idea of tracking health over debt. The folks at REA in Australia share a good [example](#) of what such health tracking can look like. They track system ratings in the categories of development, operations and architecture.

Focusing on health instead of debt is a more constructive framing. It connects a team to the ultimate value of reducing debt and helps them prioritize it. Every piece of tackled technical debt should ideally be connectable to one of the agreed expectations. Teams should treat the health rating the same as other service-level objectives (SLOs) and prioritize improvements whenever they drop out of the “green zone” for a given category.

19. Zero trust security for CI/CD

Assess

If not properly secured, the infrastructure and tools that run our build and delivery pipelines can become a big liability. Pipelines need access to critical data and systems like source code, credentials and secrets to build and deploy software. This makes these systems very inviting to malicious actors. We therefore highly recommend applying zero trust security for CI/CD pipelines and infrastructure — trusting them as little as necessary. This encompasses a number of techniques: If available, authenticate your pipelines with your cloud provider via federated identity mechanisms like OIDC, instead of giving them direct access to secrets. Implement the principle of least privilege by minimizing the access of individual user or runner accounts, rather than employing “god user accounts” with unlimited access. Use your runners in an ephemeral way instead of reusing them, to reduce the risk of exposing secrets from previous jobs or running jobs on compromised runners. Keep the software in your agents and runners up to date. Monitor the integrity, confidentiality and availability of your CI/CD systems the same way you would monitor your production software.

We’re seeing teams forget about these types of practices particularly when they’re used to working with a self-managed CI/CD infrastructure in internal network zones. While all of these practices are important in your internal networks, they become even more crucial when using a managed service, as that extends the attack surface and blast radius even more.

20. Casual management of webhooks

Hold

As remote work continues to increase, so does the adoption of chat collaboration platforms and ChatOps. These platforms often offer webhooks as a simple way to automate sending messages and notifications, but we’re noticing a concerning trend: the casual management of webhooks — where they’re treated as configuration rather than a secret or credential. This can lead to phishing attacks and compromised internal spaces.

Webhooks are credentials that offer privileged access to an internal space and may contain API keys that can be easily extracted and utilized directly. Not treating them as secrets opens up the possibility of successful phishing attacks. Webhooks in Git repos can easily be extracted and used to send fraudulent payloads, which the user may not have any way to authenticate. To mitigate this threat, teams handling webhooks need to shift their culture and treat webhooks as sensitive credentials. Software developers building integrations with ChatOps platforms must also be mindful of this risk and ensure that webhooks are handled with proper security measures.

21. Lambda pinball

Hold

While serverless architectures can be extremely useful for solving some problems, they do come with a certain level of complexity, especially when they involve nontrivial execution and data flows across multiple interdependent Lambdas — this can sometimes result in a Lambda pinball architecture. Our teams have reported that maintaining and testing Lambda pinball architectures can be very challenging: understanding the infrastructure, deployment, diagnosis and debugging can become difficult. At a code level, simple mapping between domain concepts and the multiple Lambdas involved is practically impossible, making any changes and additions challenging. Although we believe serverless is the right fit for some problems and domains, it's not a “silver bullet” for every problem, which is why you should try to avoid Lambda pinball. One pattern that can help is to draw a distinction between public and published interfaces and apply domain boundaries with published interfaces between them.

22. Planning for full utilization

Hold

While the practice of creating excess capacity in the delivery process is well-known in the product management community, we still see far too many teams planning for full utilization of team members. Reserving some capacity during sprint planning generally leads to better predictability *and* better quality; it promotes team resilience to unexpected events like illnesses, production issues, unexpected product requests and tech debt, while also allowing productive activities like team building and ideation that can lead to product innovation. Running at less than full utilization means teams can be more thoughtful about the robustness of the resulting software and pay closer attention to the right observability signals. Our experience is that a fully utilized team leads to a collapse in throughput as well, just as a fully utilized highway creates slow and demoralizing traffic. For example, when one of our teams had unpredictable support issues, they saw a 25% increase in throughput and a 50% decrease in cycle time volatility by planning feature velocity based on only two of the three developer pairs' capacities.

Platforms

Adopt

- 23. Contentful
- 24. GitHub Actions
- 25. K3s

Trial

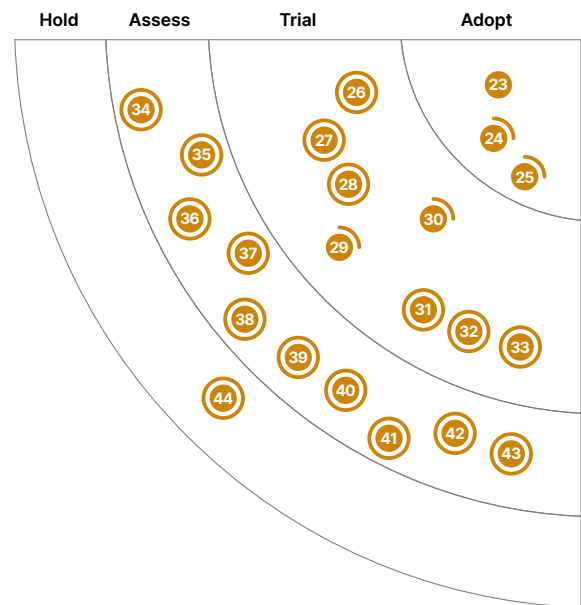
- 26. Apache Hudi
- 27. Arm in the cloud
- 28. Ax
- 29. DuckDB
- 30. Feature Store
- 31. RudderStack
- 32. Strapi
- 33. TypeDB

Assess

- 34. Autoware
- 35. Cozo
- 36. Dapr
- 37. Immuta
- 38. Matter
- 39. Modal
- 40. Neon
- 41. OpenLineage
- 42. Passkeys
- 43. Spin

Hold

- 44. Denodo as primary data transformation tool



● New ● Moved in/out ● No change

23. Contentful

Adopt

Headless content management systems have become a common component of digital platforms. Contentful is still our default choice in this space, but new entrants like Strapi have impressed us too. We particularly like Contentful's API-first approach and implementation of CMS as code. It supports powerful content modeling primitives as code and content model evolution scripts, which allow it to be treated like other data store schemas and enable evolutionary database design practices to be applied to CMS development. Recently, Contentful has released an app framework to write apps that make it easier to adapt Contentful to individual business processes and to integrate with other services. Apps can be built by and for a specific organization but a marketplace for apps is emerging, too.

24. GitHub Actions

Adopt

GitHub Actions has become a default starting point for many teams that need to get CI or CD up and running quickly in a greenfield environment. Among other things, it can take on more complex workflows and call other actions in composite actions. Although the ecosystem in GitHub Marketplace continues to grow, we still urge caution in giving third-party GitHub Actions access to your build pipeline. We recommend following GitHub's advice on security hardening to avoid sharing secrets in insecure ways. However, the convenience of creating your build workflow directly in GitHub next to your source code combined with the ability to run GitHub Actions locally, using open-source tools such as act, is a compelling option that has streamlined the setup and onboarding of our teams.

25. K3s

Adopt

K3s remains our default Kubernetes distribution for edge computing needs and resource-constrained environments. It's a lightweight, fully compliant Kubernetes but with reduced operational overhead. It uses sqlite3 as the default storage backend instead of etcd. It has a reduced memory footprint because it runs all relevant components in a single process. We've used K3s in environments like industrial control systems and point-of-sale machines, and we're quite happy with our decision. With the K3s runtime containerd now supporting wasm, K3s can run and manage WebAssembly workloads directly, further reducing the runtime overhead.

26. Apache Hudi

Trial

Apache Hudi is an open-source data lake platform that brings ACID transactional guarantees to the data lake. Our teams have had a great experience using Hudi in a high-volume, high-throughput scenario with real-time inserts and upserts. We particularly like the flexibility Hudi offers for customizing the compaction algorithm which helps in dealing with "small files" problems. Apache Hudi falls in the same category as Delta Lake and Apache Iceberg. They all support similar features, but each differs in the underlying implementations and detailed feature lists.

27. Arm in the cloud

Trial

Arm compute instances in the cloud have become increasingly popular over the past few years due to their cost and energy efficiency compared to traditional x86-based instances. Many cloud providers now offer Arm-based instances, including AWS, Azure and GCP. The cost benefits of running Arm in the cloud can be particularly beneficial for businesses that run large workloads or need to scale. Based on our experiences we recommend Arm compute instances for all workloads unless there are architecture-specific dependencies. The tooling to support multiple architectures, like multi-arch docker images, also simplify build and deploy workflows.

28. Ax

Trial

Faced with the challenge of exploring large configuration spaces, where it may take a significant amount of time to evaluate a given configuration, teams can turn to adaptive experimentation, a machine-guided, iterative process to find optimal solutions in a resource-efficient manner. Ax is a platform for managing and automating adaptive experiments, including machine learning experiments, A/B tests and simulations. Currently, it supports two optimization strategies: Bayesian optimization using BoTorch, which is built on top of PyTorch, and contextual bandits. Facebook, when releasing Ax and BoTorch, described use cases like increasing the efficiency of back-end infrastructure, tuning ranking models and optimizing hyperparameter search for a machine learning platform. We've had good experiences using Ax for a variety of use cases, and while tools for hyperparameter tuning exist, we're unaware of a platform that provides functionality in a scope similar to Ax.

29. DuckDB

Trial

DuckDB is an embedded, columnar database for data science and analytical workloads. Data analysts usually load the data locally in tools like pandas or data.table to quickly analyze patterns and form hypotheses before scaling the solution in the server. However, we're now using DuckDB for such use cases, because it unlocks the potential to do larger than memory analysis. DuckDB supports range joins, vectorized execution and multiversion concurrency control (MVCC) for large transactions, and our teams are quite happy with it.

30. Feature Store

Trial

Any software system needs to properly represent the given domain in which it is employed and should always be informed by key aims and goals. Machine learning (ML) projects are no different. Feature engineering is a crucial aspect of engineering and designing ML software systems. Feature Store is a related architectural concept to facilitate the identification, discovery and monitoring of the features pertinent to the given domain or business problem. Implementing this concept involves a combination of architectural design, data engineering and infrastructure management to create a scalable, efficient and reliable ML system. From a tooling perspective, you can find open-source and fully managed platforms, but they're only one part of this concept. In the end-to-end design of ML systems, implementing a feature store enables the following capabilities: the ability to (1) define the right

features; (2) enhance reusability and make features consistently available regardless of the type of model, which also includes setting up the feature engineering pipelines that curate data as described in the feature store; (3) enable feature discovery and (4) enable feature serving. Our teams leverage feature stores in production to reap these benefits for end-to-end ML systems.

31. RudderStack

Trial

RudderStack is a customer data platform (CDP) that makes it easy to store data in a data warehouse or data lake. This approach, increasingly known as Headless CDP, separates the CDP's features from its user interface and emphasizes configurability through APIs and the data warehouse/lake as primary storage. As expected from a product in this category, RudderStack has a rich repository of integrations with third-party products (both as source and sink) and the ability to ingest custom events. RudderStack has both a commercial offering and a self-hosted OSS version with functionality limitations.

32. Strapi

Trial

Strapi is an open-source, NodeJS-based, headless content management system (CMS) similar to Contentful. It has been around for a while, and we've used it successfully in a few projects. Strapi provides both REST and GraphQL APIs, has comprehensive documentation, features an easy-to-use data model API and supports customizing both UI and logic.

33. TypeDB

Trial

TypeDB is a knowledge graph database, designed to work with intricate data relationships which makes it easier to query and analyze large data sets. TypeDB's TypeQL query language has a SQL-like syntax which eases the learning curve for schema definition, querying and exploration. TypeDB comes with a variety of tools that make it easier to work with the database, including a command-line interface and a graphical user interface, TypeDB Studio, which provides some features for working with TypeDB, such as managing schemas, querying data, visualizing relationships or even collaborating with others. There is a good deal of documentation available and an active community for support. Our teams used it to build knowledge graphs of taxonomic concepts across different databases and took advantage of its strong inference capabilities by adding new inference rules increasing efficiency and reducing workload. With its intuitive developer experience and supportive community, TypeDB is a good candidate to consider for any team looking to build data solutions that depend on complex data relationships, including natural language data, recommendation engines and knowledge graphs.

34. Autoware

Assess

Autoware is an open-source autonomous driving software stack built on ROS (Robot Operating System) which can be used to develop and deploy advanced driver assist systems (ADAS) for a wide range of vehicles like cars and trucks. It provides a set of tools and algorithms for various aspects

of autonomous driving, such as perception, decision-making and control. It also has a planning and control module that generates a trajectory for the vehicle based on its environment and objectives. It encourages open innovations in autonomous driving technology. We're building prototypes using Autoware to validate new product ideas and find it helpful.

35. Cozo

Assess

Cozo is an embeddable relational database that uses Datalog for querying. We're intrigued with its support for time-travel queries and modeling graph data in relational schema. We quite like that it delegates data storage to existing popular engines — including SQLite, RocksDB, Sled and TiKV. Although Cozo is still in its early stages of development, we find it's worth assessing.

36. Dapr

Assess

Dapr, short for Distributed Application Runtime, helps developers to build resilient, stateless and stateful microservices that run in the cloud. Some people may confuse it with a service mesh, because it uses a sidecar architecture that runs as a separate process alongside the application. Dapr is more application oriented and focuses on encapsulating the fault tolerance and connectivity required for building distributed applications. For example, Dapr provides multiple building blocks, from service invocation and message pub/sub to distributed lock, all of which are common patterns in distributed communication. One of our teams evaluated Dapr on a recent project; given their positive experience, they're looking forward to bringing it to other projects in the future.

37. Immuta

Assess

Immuta is a data security platform that allows you to secure access to your data, automatically discover sensitive data and audit how data is being used in an organization. In the past, we've talked about the importance of automation, engineering practices and treating security policy as code when we think about security concerns. Data security is no different. Our teams have been exploring Immuta to manage data policies as code to allow for fine-grained access control which is beyond what role-based access control (RBAC) can offer. Version-controlled policies can be tested and then provisioned as part of a CI/CD pipeline. In a decentralized data ecosystem, like one facilitated by data mesh, having domain-specific roles can lead to role or group proliferation in the identity system. Immuta's attribute-based access control (ABAC) capability reduces the access grant to a mathematical equation of matching an "attribute" on the user to a "tag" on the data source. This platform is still new but certainly worth highlighting for data security needs.

38. Matter

Assess

Matter is an open standard for smart home technology, launched by Amazon, Apple, Google, Comcast and the Zigbee Alliance (now Connectivity Standards Alliance, or CSA). It enables devices to work with any Matter-certified ecosystem, thus reducing fragmentation and promoting interoperability among devices and IoT platforms from different providers. Its focus on standardization at the

application level, support for Wi-Fi and [Thread](#) as communication mediums and backing from major tech companies set it apart from other protocols like Zigbee. Although the number of Matter-enabled devices is still relatively low, its growing importance in the IoT space makes it worth assessing for those looking to build smart home and IoT solutions.

39. Modal

Assess

[Modal](#) is a platform as a service (PaaS) that offers on-demand compute without the need for your own infrastructure. Modal lets you deploy machine learning models, massively parallel compute jobs, task queues and web apps. It provides a container abstraction that makes the switch from local to cloud deployment seamless, with hot reload both locally and in the cloud. It even removes deployments automatically, avoiding the need for manual clean-up, but can also make them persistent.

Modal is written by the same team that developed the first recommendation engine for Spotify. It takes care of the AI/ML stack end-to-end and can provide on-demand GPU resources, which is useful if you have particularly intensive computational needs. Whether you're working on your laptop or in the cloud, Modal just works, providing an easy and efficient way to run and deploy your projects.

40. Neon

Assess

[Neon](#) is an open-source alternative to AWS Aurora PostgreSQL. Cloud-native analytical databases have embraced the technique of separating storage from compute nodes to elastically scale on demand. However, it's difficult to do the same in a transactional database. Neon achieves this with its new [multi-tenant storage engine](#) for PostgreSQL. With minimal changes to the mainstream PostgreSQL code, Neon leverages AWS S3 for long-term data storage and elastically scales the processing up or down (including scale-to-zero) for compute. This architecture has several benefits — including cheap and fast clones, copy-on-write and [branching](#). We're quite excited to see new innovations on top of PostgreSQL. Our teams are evaluating Neon, and we recommend you assess it as well.

41. OpenLineage

Assess

[OpenLineage](#) is an open standard for lineage metadata collection for data pipelines, designed to instrument jobs as they're running. It defines a generic model of run, job and data set entities using consistent naming conventions. The core lineage model is extensible by defining specific facets to enrich those entities. OpenLineage solves the interoperability problem between producers and consumers of lineage data who otherwise would need to know how to speak to each other in various ways. Although there is a risk of it being another "standard in the middle," being a [Linux Foundation AI & Data Foundation](#) project increases its chances of gaining widespread adoption. OpenLineage currently supports data collection for multiple platforms, such as [Spark](#), [Airflow](#) and [dbt](#), although users need to configure its listeners. Support for OpenLineage data consumers is [more limited at this time](#).

42. Passkeys

Assess

The “end of passwords” might be near, finally. Shepherded by the FIDO alliance and backed by Apple, Google and Microsoft, [passkeys](#) are nearing mainstream usability. When setting up a new login with passkeys, a key pair is generated: the website receives the public key and the user keeps the private key. Handling login uses asymmetric cryptography. The user proves that they’re in possession of the private key, but, unlike passwords, it’s never sent to the website. On users’ devices, access to passkeys is protected using biometrics or a PIN.

Passkeys can be stored and synced within the Big Tech ecosystems, using Apple’s iCloud Keychain, Google Password Manager or Windows Hello. In most cases this works only with recent OS and browser versions. Notably, storing passkeys in Windows Hello is not supported on Windows 10. Fortunately, though, the [Client to Authenticator Protocol \(CTAP\)](#) makes it possible for passkeys to be kept on a different device other than the one that creates the key or needs it for login. For example, a user creates a passkey for a website on Windows 10 and stores it on an iPhone by scanning a QR code. Because the key is synced via iCloud the user can log in to the website from, say, their MacBook. Passkeys can be stored on hardware security keys, too, and support for native apps has arrived on iOS and Android.

Despite some usability issues — for example, Bluetooth needs to work because device proximity is checked when a QR code is scanned — passkeys are worth considering. We suggest you experiment with them on [passkeys.io](#) to get a feeling for their usability.

43. Spin

Assess

[Spin](#) is an open-source platform for building and running microservices in [WebAssembly \(WASM\)](#). In previous editions of the Radar, we talked about WebAssembly in the context of browsers, but we’re now witnessing the penetration on the server side due to its capabilities for [fine-grained sandboxing](#), cross-language interoperability and hot reloading. With Spin CLI, you can quickly create and distribute WebAssembly microservices in [Rust](#), [TypeScript](#), Python and [TinyGo](#). We’re excited about Spin, and we recommend you carefully assess it as it moves out of early preview.

44. Denodo as primary data transformation tool

Hold

[Denodo](#) is a data virtualization tool that aims to make it easier to expose and secure transformed, consumer-friendly data (from multiple underlying data sources and through a variety of interfaces) from one platform. Data transformations within Denodo can be defined by creating virtual databases and views using a SQL-like language called VQL which are executed when a user queries the virtual database. Underneath, Denodo can delegate queries on the virtual databases against one or multiple underlying databases.

Although Denodo makes it easy to start exposing consumer-friendly data, performance degrades as layers of views and virtual databases are built on top of each other and queries with multiple joins start hitting multiple underlying databases. These problems are solvable, but they require fairly deep knowledge of the product’s behavior and performance tuning options. Because of these drawbacks and given its limited support for unit testing, we recommend that you do not use Denodo as a primary data transformation tool and use tools like [Spark](#) or SQL (with [dbt](#)) for your data transformations instead.

Tools

Adopt

45. DVC

Trial

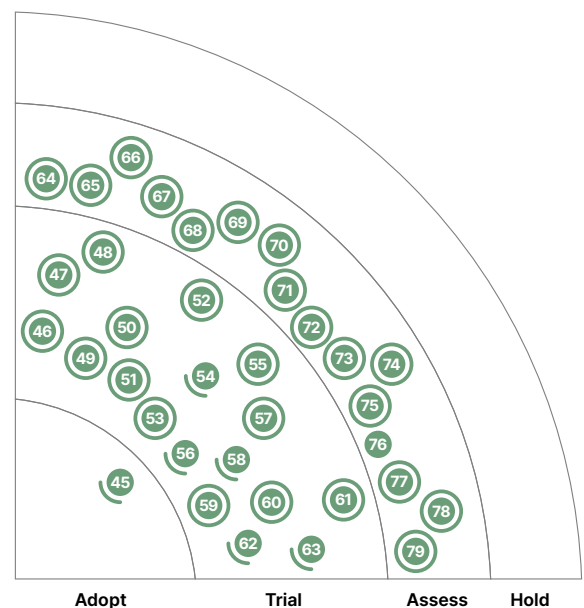
- 46. Akeyless
- 47. Apicurio Registry
- 48. EventCatalog
- 49. FOSSA
- 50. Gitleaks
- 51. Helmfile
- 52. IBM Equal Access Accessibility Checker
- 53. Ktlint
- 54. Kubeflow
- 55. Mend SCA
- 56. Mozilla SOPS
- 57. Ruff
- 58. Soda Core
- 59. Steampipe
- 60. Terraform Cloud Operator
- 61. TruffleHog
- 62. Typesense
- 63. Vite

Assess

- 64. axe Linter
- 65. ChatGPT
- 66. DataFusion
- 67. Deepchecks
- 68. Design token translation tools
- 69. Devbox
- 70. Evidently
- 71. Giskard
- 72. GitHub Copilot
- 73. iamlive
- 74. Kepler
- 75. Kubernetes External Secrets Operator
- 76. Kubeshark
- 77. Obsidian
- 78. Ory Kratos
- 79. Philips's self-hosted GitHub runner

Hold

—



● New ● Moved in/out ● No change

45. DVC

Adopt

DVC continues to be our tool of choice for managing experiments in data science projects. The fact that it's Git-based makes it a known turf for developers to bring engineering practices to the data science ecosystem. DVC's opinionated view of a model checkpoint carefully encapsulates a training data set, a test data set, model hyperparameters and the code. By making reproducibility a first-class concern, it allows the team to time travel across various versions of the model. Our teams have successfully used DVC in production to enable continuous delivery for ML (CD4ML); it can be plugged in with any type of storage (including AWS S3, Google Cloud Storage, MinIO and Google Drive). However, with data sets getting bigger, file system-based snapshotting could become particularly expensive. When the underlying data is changing rapidly, DVC on top of a good versioned storage allows tracking model drifts over a period of time. Our teams have effectively used DVC on top of data storage formats like Delta Lake which optimizes versioning (COW). A majority of our data science teams set up DVC as a day zero task while they bootstrap a project; for this reason we're happy to move it to Adopt.

46. Akeyless

Trial

As more organizations adopt cloud computing, many are starting to integrate multiple cloud providers simultaneously to maximize flexibility and minimize vendor lock-in. However, managing keys and access controls across multiple cloud providers can be a significant challenge, leading to increased complexity and security risks. Akeyless is a centralized, cloud-based platform that provides unified secrets management with a range of advantages for managing secrets and sensitive data. It integrates seamlessly with different providers, simplifying the management of secrets and access controls to monitor and control who has access to sensitive data; with encryption, access controls, multi-factor authentication and other security mechanisms it ensures only authorized users are able to access sensitive data. Additionally, it provides an intuitive interface for administration and monitoring, providing a less complex and more scalable developer and administration experience.

47. Apicurio Registry

Trial

Within any organization, API producers and consumers need to stay in sync about the schemas that will be used for communication among them. Especially as the number of APIs and related producers and consumers grow in the organization, what may start with simply passing around schemas among teams will start to hit scaling challenges. Faced with this issue, some of our teams have turned to Apicurio Registry, an open-source, centralized registry for various types of schemas and API artifacts, including OpenAPI specifications and Protobuf and Avro schemas. Apicurio Registry allows users to interact with it through a UI as well as a REST API and a Maven plugin. It also has the option to enforce schema evolution restrictions, such as backward compatibility. Moreover, when it comes to working with Kafka clients, Apicurio Registry is compatible with Confluent Schema Registry. While our teams have found Confluent Schema Registry's documentation more helpful, Apicurio Registry meets their needs for a source of truth for various schemas.

48. EventCatalog

Trial

Enterprises now often use event streaming as the source of truth and as an information-sharing mechanism in microservices architectures. This creates the need to standardize event types and share those standards across the enterprise. Event schema registries are commonly deployed but the existing offerings tend to be specialized to a single broker such as Apache Kafka or Azure Event Hub. They also fall short of conveying rich documentation about event types that goes beyond simple schema definitions. EventCatalog is an open-source project that provides something we often see businesses building for themselves: a widely accessible repository of documentation for events and schemas. These describe the role the events play in the business, where they belong in a business domain model and which services subscribe and publish them. If you're looking for a way to publish event documentation to your organization, this tool might save you the trouble of building it yourself.

49. FOSSA

Trial

FOSSA is an open-source compliance tool that helps developers and teams determine which open-source components their code relies on and which licenses these components are released under. This information is essential for ensuring compliance with various open-source licenses and maintaining the Software Bill of Materials. FOSSA integrates with dependency management tools of various tech stacks to identify which open-source components are used in a project. It also highlights any license issues based on the organization's policies and generates reports of the same. Some key features of FOSSA include its ability to integrate with development workflows, such as the CI, and to perform real-time compliance monitoring. Many of our clients and teams have found FOSSA to be a valuable and effective tool.

50. Gitleaks

Trial

Gitleaks is an open-source SAST (static application security testing) command line tool for detecting and preventing hardcoded secrets like passwords, API keys and tokens in Git repositories. It can be used as a Git pre-commit hook or in the CI/CD pipeline. Our teams found Gitleaks to be more sensitive than some of the other secret-scanning tools. Gitleaks utilizes regular expressions and entropy string coding to detect secrets. In our experience, the flexibility to supply custom regex along with entropy coding allowed the teams to better categorize secrets based on their needs. For example, instead of categorizing all API keys as "generic-api-key," it allowed categorization as specific "cloud provider key."

51. Helmfile

Trial

Helmfile is an open-source command line tool and a declarative specification for managing and installing a collection of Helm charts. You can use it to help with version control of the Helm values files, the charts used and other overrides. It enables CI/CD workflows with Helm charts and helps create reproducible environments. We've used Helmfile to manage complex deployments with multiple dozens of Helm charts and found it simplifies the deployment workflow.

52. IBM Equal Access Accessibility Checker

Trial

Defects are cheaper to fix when they're caught early. That's why we always try to get the fastest possible feedback to developers in the form of static analysis, unit tests or end-to-end tests run in the local environment. Accessibility is no exception to this and that's why we've featured tools such as [Lighthouse](#), [axe-core](#) and [axe Linter](#) in the past. When it comes to automatically testing web pages that are already deployed in production, one of our teams chose instead to go with [IBM Equal Access Accessibility Checker](#) in a head-to-head comparison. Although we're still in the process of assessing the results, we can say that it offers an efficient way to test pages once they've been deployed. We emphasize that this should be used to augment, not replace, early automated testing by the developer. The tool is distributed under a Creative Commons license and is free to use under those restrictions.

53. Ktlint

Trial

The [Kotlin](#) ecosystem continues to evolve, and our teams report positive experiences with [Ktlint](#), a simple and easy-to-configure linter and formatter for Kotlin code. We like [opinionated and automated code formatting](#) as it lets developers focus more on what the code does rather than how it looks; this tool enables development teams to maintain consistency and readability in their codebases efficiently, reducing the likelihood of messy merges due to formatting issues. Ktlint can be easily configured to run in pre-commit hooks, targeting only the files with changes and resulting in faster integration processes.

54. Kubeflow

Trial

[Kubeflow](#) is a [Kubernetes](#)-native machine learning (ML) platform that simplifies build, train and deploy lifecycles of models to diverse infrastructure. We've extensively used [Pipelines](#) to encode ML workflows for several models across experimentation, training and serving use cases. Besides [Pipelines](#), Kubeflow ships with multiple [components](#), among which we find hyperparameter tuning with [Katib](#) and [multi-tenancy](#) to be quite useful.

55. Mend SCA

Trial

[Mend SCA](#) (software composition analysis), previously [Whitesource](#), helps detect open-source software dependencies by identifying if they are up to date, contain security flaws or have licensing requirements. Our teams have had good experience with integrating Mend SCA in their paths to production. Right from IDE integration, raising an automatic PR based on an identified issue to integrating into the CI/CD pipeline, this tool offers a great developer experience. Other popular SCA tools, such as [Snyk](#), are comparable and also worth exploring for your security needs.

56. Mozilla SOPS

Trial

Our advice when it comes to secrets management has always been to decouple it from source code. However, teams are often presented with a tradeoff between full automation (in the spirit of infrastructure as code) versus a few manual steps (using tools like vaults) for managing, seeding and rotating seed secrets. For instance, our teams use SOPS to manage seed credentials for bootstrapping infrastructure. In some situations, however, it's impossible to remove secrets from legacy code repositories. For such needs, we found Mozilla SOPS to be a good choice for encrypting secrets in text files. SOPS integrates with cloud-managed keystores such as AWS and GCP Key Management Service (KMS) or Azure Key Vault as sources of encryption keys. It also works cross-platform and supports PGP keys.

57. Ruff

Trial

Ruff is a new linter for Python. For us, the question is not whether to use a linter or not but which linter to use, and there are several choices for Python. Ruff stands out for two reasons: its out-of-box experience and its speed. It has over 500 rules built in and readily replaces Flake8, including many of that linter's plug-ins. The claims by the team behind Ruff about its performance are borne out by our experience. It really is at least an order of magnitude faster than other linters, which is a huge benefit because it helps reduce build times on large codebases.

58. Soda Core

Trial

Soda Core is an open-source data quality and observability tool. Our teams have used it to validate data as it arrives in a system, before and after transformations, and set up automated monitoring checks for anomalies. We're happy with SodaCL, the DSL for writing data checks in Soda Core, as it helps team members other than data engineers write quality checks. Overall, our experience using Soda Core to find and resolve data issues at scale has been positive.

59. Steampipe

Trial

Steampipe is an open-source tool that lets you instantly query cloud services like AWS, Azure and GCP with SQL. With 100+ plugins and built-in support for creating dashboards, Steampipe makes it trivial to connect live cloud configuration data with internal or external data sets and create security or compliance dashboards. We've enjoyed working with Steampipe and created several such dashboards with AWS cloud configurations.

60. Terraform Cloud Operator

Trial

More and more teams are using the [Kubernetes Operators](#) pattern to manage their Kubernetes clusters. We used to recommend [Crossplane](#) for this, and now we have an alternative tool, [Terraform Cloud Operator](#) for Kubernetes. This tool integrates Terraform Cloud and Kubernetes by extending the Kubernetes control plane to enable lifecycle management of cloud and on-premise infrastructures through Kubernetes manifests. Our team uses it to provision resources from Kubernetes namespaces and RoleBindings to cloud database instances and other SaaS resources. We quite like it because it leverages the Terraform module, which is a more familiar abstraction layer for us to operate cloud resources.

61. TruffleHog

Trial

[TruffleHog](#) is an open-source SAST (static application security testing) tool for detecting secrets in various sources. While GitHub and GitLab repositories are the most popular use cases, it can also be used to scan cloud storage buckets like S3 and GCS, local files and directories and [CircleCI](#) logs. Developers can set up TruffleHog as a pre-commit hook or scan the history of existing repositories in an entire GitHub organization to detect secrets. The tool supports detecting custom regex patterns, which have been found to be quite useful even in its current alpha stage. TruffleHog also has an enterprise version, but our devs have found the open-source version easy to set up and sufficient for the most common use cases. The tool has a very active community who regularly adds features.

62. Typesense

Trial

[Typesense](#) is an open-source, typo-tolerant search engine optimized for low-latency and high-performance search experiences. If you're building a latency-sensitive search application with a search index size that can fit in memory, Typesense is a powerful alternative. Our teams use Typesense in high availability multi-node clusters to distribute workload and ensure critical search infrastructure is resilient. They had a good experience with Typesense in production, which is why we've moved it to Trial.

63. Vite

Trial

[Vite](#), a front-end build tool, has continued to mature and grow in popularity since we featured it in the [Assess](#) ring in the previous Radar. It is rapidly becoming the default choice among our teams when starting a new front-end project. Vite provides a set of defaults for building, bundling and managing dependencies in applications that depend on ES modules in the browser. Because it takes advantage of the native speed of [esbuild](#) and the [Rollup](#) bundler, [Vite significantly improves](#) the front-end developer experience. Moreover, when used with React, Vite offers an attractive alternative to the [stalwart but nearly defunct](#) Create React App. Vite relies on ES modules, and unlike most older tools, it doesn't provide [shimming](#) or polyfills, which means you need a different strategy for older browsers that don't support ES modules. In cases where older browsers had to be supported, some of our teams import polyfills at the module level so that Vite can be used consistently across environments.

64. axe Linter

Assess

It's becoming increasingly easy for developers to catch accessibility issues early in the development process. While tools like [axe-core](#) scan code for accessibility issues in your pipelines, the [axe Linter](#) VSCode extension helps find them even before that, while writing code. The vast majority of accessibility issues fall into categories that could be prevented by automated testing and using live feedback linters like this.

65. ChatGPT

Assess

[ChatGPT](#) is an interesting tool that has the potential to be useful for various aspects of the software creation process. As a large language model (LLM) that has “read” billions of web pages, ChatGPT can provide additional perspectives and assist with different tasks, from generating ideas and requirements to creating code and tests. Its ability to work across multiple parts of the software lifecycle makes it a versatile tool that might improve efficiency and reduce errors in the development process. GPT4, the LLM that powers ChatGPT, now also has the ability to integrate with external tools such as a knowledge management repository, sandboxed coding environment or web search. For now, we think that ChatGPT is best used as an input to a process, such as helping with a first draft of a story or the boilerplate for a coding task, rather than a tool that produces “fully baked” results.

There are concerns around intellectual property and data privacy with these AI tools, including some unresolved legal questions, so we recommend organizations seek advice from their legal teams before use. Some of our clients have already begun experimenting with ChatGPT across various stages of the software lifecycle, and we encourage others to explore the tool and assess its potential benefits. We expect that, like [GitHub Copilot](#), a “for business” offering will soon be available which may ease intellectual property concerns.

66. DataFusion

Assess

[DataFusion](#) is a part of the data community's exploration of [Rust's](#) performance, memory safety and concurrency features applied to data processing. It shares similarities with [Polars](#), namely a familiar DataFrame API in Rust (with Python bindings), the use of [Apache Arrow](#) under the hood and SQL support. Even though it's primarily designed for single-process execution, distributed processing support is in the works within [Ballista](#). We think the Rust libraries for data processing are an evolving space worth following and exploring, and DataFusion is a part of it.

67. Deepchecks

Assess

As machine learning finds its way into the mainstream, practices are maturing around automatically testing models, validating training data and observing model performance in production. Increasingly, these automated checks are being incorporated into continuous delivery pipelines or run against production models to detect drift and model performance. A number of tools with similar or overlapping capabilities have emerged to handle various steps in this process ([Giskard](#) and [Evidently](#) are also covered in this volume). [Deepchecks](#) is another of these tools that's available as an open-source Python library and can be invoked from pipeline code through an extensive set of APIs. One unique feature is its ability to handle either tabular or image data with a module for language data currently in alpha release. At the moment, no single tool can handle the variety of tests and guardrails across the entire ML pipeline. We recommend assessing Deepchecks for your particular application niche.

68. Design token translation tools

Assess

[Design tokens](#) are a useful mechanism for defining standard elements in [design systems](#). But, keeping those design elements consistent across media such as mobile apps or web frameworks is an increasingly formidable task. [Design token translation tools](#) simplify this problem by organizing and automating transformation from the token description (in YAML or JSON) into the code that actually controls rendering in a given medium such as CSS, React components or HTML. [Style Dictionary](#) is an open-source example that is widely used and integrates well into automated build pipelines, but there are also commercial alternatives such as [Specify](#).

69. Devbox

Assess

[Devbox](#) provides an approachable interface for creating reproducible, per-project development environments leveraging the Nix package manager. Our teams use it to eliminate version and configuration mismatches in their development environments, and they like it for its ease of use. Devbox supports shell hooks, custom scripts and [devcontainer.json](#) generation for integration with VSCode.

70. Evidently

Assess

[Evidently](#) is an open-source Python tool designed to help build monitoring for machine learning models to guarantee their quality and stable production operations. It can be used at various stages of a model lifecycle: as a dashboard to review the model in a notebook, as part of a pipeline or as a monitoring service after deployment. With a particular focus on model drift detection, Evidently also offers features such as model quality, data quality inspection and target drift detection. In addition, it has many built-in metrics, associated visualizations and tests which are easily combined into a report, dashboard or a test-driven pipeline.

71. Giskard

Assess

Giskard is an open-source tool designed to help organizations build more robust and ethical AI models by providing quality assurance capabilities with a focus on explainability and fairness. It facilitates cooperation between technical and nontechnical stakeholders, allowing them to evaluate models collaboratively and establish acceptance criteria based on bias avoidance and other essential quality metrics. Giskard ensures model outcomes are better aligned with business objectives and helps to solve quality issues before production deployment.

72. GitHub Copilot

Assess

GitHub Copilot is an AI coding assistant, created by a collaboration between Microsoft and OpenAI. It uses machine learning models to generate suggestions based on the context of what a developer is working on. It features strong IDE integration and uses an existing codebase and editor context to create suggestions. Despite being billed as “your AI pair programmer” we would not call what it does “pairing” — we’d probably describe it as a kind of supercharged, context-sensitive Stack Overflow. When it correctly predicts what a developer is trying to do, it can be a powerful tool for getting stuff done. Like all LLM-based AIs, though, it has a tendency to hallucinate the use of plausible but nonexistent APIs and may introduce bugs through slightly faulty algorithms. We’ve had success generating code at the line, block and method level, as well as creating tests or infrastructure configurations. Interestingly, it works best when you use good naming practices, so it encourages more readable code.

AI tool capabilities are advancing rapidly, and we think it’s sensible for organizations to try them. Some sales pitches for Copilot have claimed very high efficiency gains, but we remain skeptical: after all, writing code isn’t the only thing that developers spend time on, and it’s notoriously difficult to measure developer productivity in the first place. That said, Copilot is a fairly inexpensive tool; if it offers any productivity gain at all, it’s probably worth it. Copilot X — in preview as of this writing — offers additional functionality and integration within a software creation workflow. Copilot has a “for business” offering, which provides more clarity around intellectual property issues as well as the ability to manage tool features centrally across an organization. We think these features are critical for enterprise adoption.

73. iamlive

Assess

Creating exactly the minimum viable AWS IAM policies we want, according to the principle of least privilege, can be a long journey of trial and error. iamlive can shorten that journey considerably. It monitors the AWS CLI calls made from a machine and determines the policies needed to execute those calls. The tool generates a policy document with statements, actions, principals and resources that can be used as a good starting point. We’ve found it particularly useful to create policies needed in CI/CD pipelines that provision infrastructure, reducing the usual back and forth after a Terraform run fails because the IAM role’s policy is insufficient.

74. Kepler

Assess

Measuring energy consumption is an important step for teams to reduce the carbon footprint of their software. [Cloud Carbon Footprint \(CCF\)](#) estimates energy based on billing and usage data retrieved from cloud APIs. [Kepler](#) — short for Kubernetes-based Efficient Power Level Exporter — goes one step further: it uses software counters via [RAPL](#), [ACPI](#) and [nvml](#) to measure power consumption by hardware resources and employs an [eBPF](#)-based approach to attribute power consumption to processes, containers and Kubernetes pods. Power usage is then converted to energy estimates using a custom ML model and data from the [SPEC Power](#) benchmark. Finally, pod-level energy consumption reporting is made available as [Prometheus](#) metrics. In cases where Kubernetes is running on virtual machines, for example when not using bare metal instances, Kepler uses cgroups to [estimate](#) energy consumption. We've had significant experience with CCF and can attest to its usefulness, but we're intrigued by the Kepler project's approach.

75. Kubernetes External Secrets Operator

Assess

[Kubernetes External Secrets Operator](#) allows external secret providers to be integrated with [Kubernetes](#). It reads from the external provider API and injects the result into a Kubernetes Secret. The operator works with a large variety of secret management tools, including some we've featured in previous editions of the Radar. Our teams have found it simplified the use of secrets when working with Kubernetes by allowing the use of a single store across a whole project.

76. Kubeshark

Assess

[Kubeshark](#) is an API traffic viewer for [Kubernetes](#). Until November 2022, it was known as Mizu. Unlike other tools, Kubeshark 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.

77. Obsidian

Assess

Knowledge management is critical for tech workers, as we need to be constantly learning and staying up to date with the latest technology developments. Recently, tools such as [Obsidian](#) and [Logseq](#) have emerged in the category of note-taking tools that support linking notes to form a knowledge graph, while storing them in plain markdown files in a local directory, thus letting users own their data. These tools help users organize and link their notes in a flexible, nonlinear way.

Obsidian has a rich repository of community plugins. Some that have caught our attention, in particular, are [Canvas](#), akin to a local version of Miro or Mural, and [Dataview](#), which effectively treats your notes as a database and provides a query language for filtering, sorting and extracting data from your markdown notes.

78. Ory Kratos

Assess

We've already assessed [Ory Hydra](#) as a self-hosted OAuth2 solution, and the feedback from the team has been good. This time, we turn to [Ory Kratos](#), an API-first identity and user management system that's developer friendly and easy to customize. It already provides common functions we want to achieve in an identity management system, including self-service login and registration, multi-factor authentication (MFA/2FA), account verification and account recovery. Like Hydra, Kratos is headless and requires developers to build the UI themselves, which gives the team more flexibility. Developers can also customize identity schema to fit different business contexts. Kratos has no external dependencies other than the database, and it's easy to deploy and scale in different cloud environments. If you need to build a user management system, we recommend you give Kratos a try.

79. Philips's self-hosted GitHub runner

Assess

While [GitHub Actions](#) runners cover a wide range of the most common runtimes, sometimes you need something that is more specific to your particular use case, such as a less common language runtime or a particular hardware configuration. That's when you need a self-hosted runner. Philips's [self-hosted GitHub runner](#) is a Terraform module that lets you spin up custom runners on AWS EC2 spot instances. The module also creates a set of Lambdas to make up for the fact that you lose some of GitHub Actions' lifecycle management when you self-host runners. They do the heavy lifting for scaling runners up and down as needed. That helps manage costs and allows you to make runners ephemeral, which helps improve repeatability and security. When you do need to self-host runners, you might miss a lot of things when building custom runners from scratch. Look for tools like this one instead.

Languages and Frameworks

Adopt

- 80. Gradle Kotlin DSL
- 81. PyTorch

Trial

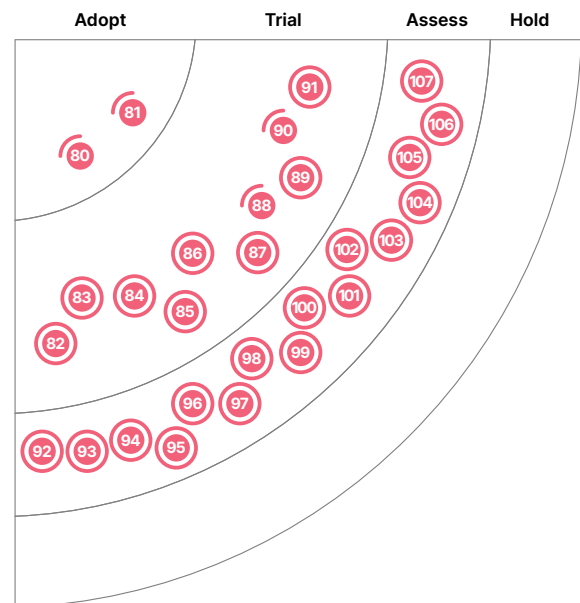
- 82. dbt-unit-testing
- 83. Jetpack CameraViewfinder
- 84. Jetpack DataStore
- 85. Mikro ORM
- 86. Per-app language preferences
- 87. Quarto
- 88. River
- 89. Stencil
- 90. Synthetic Data Vault
- 91. Vitest

Assess

- 92. .NET 7 Native AOT
- 93. .NET MAUI
- 94. dbt-expectations
- 95. Directus
- 96. Ferrocene
- 97. Flutter for embedded
- 98. Fugue
- 99. Galacean Engine
- 100. LangChain
- 101. mljar-supervised
- 102. nanoGPT
- 103. pandera
- 104. Qwik
- 105. SolidJS
- 106. Turborepo
- 107. WebXR Device API

Hold

—



● New ● Moved in/out ● No change

80. Gradle Kotlin DSL

Adopt

Our teams now view Gradle Kotlin DSL as the default for starting new projects using [Gradle](#), preferring it over [Groovy](#). Teams already using Groovy should consider migration. [Kotlin](#) provides better support for refactoring and simpler editing in IDEs, and our teams report that it produces code that is easier to read and maintain. Given some IDEs now support migration, it should be relatively quick to experiment with replacing existing Groovy. In some situations Kotlin might be slower than Groovy; however, for many projects, this is unlikely to impact the team.

81. PyTorch

Adopt

[PyTorch](#) continues to be our choice of machine learning (ML) framework. Most of our teams prefer [PyTorch](#) over [TensorFlow](#). [PyTorch](#) exposes the inner workings of ML that [TensorFlow](#) hides, making it easier to debug. With dynamic computational graphs, model optimization is much easier compared to any other ML framework. The extensive availability of [State-of-the-Art \(SOTA\) models](#) and the ease of implementing research papers make [PyTorch](#) stand out. When it comes to graph ML, [PyTorch Geometric](#) is a more mature ecosystem and our teams have had great experiences with it. [PyTorch](#) has also gradually bridged gaps when it comes to model deployment and scaling; our teams have used [TorchServe](#) to serve pretrained models successfully in production, for example. With many teams defaulting to [PyTorch](#) for their end-to-end deep-learning needs, we happily recommend adopting [PyTorch](#).

82. dbt-unit-testing

Trial

[dbt-unit-testing](#) is a [dbt](#) package that allows writing unit tests for a model and its logic by mocking its dependencies. This brings the engineering rigor of fast development feedback to the data ecosystem. Our teams use this package with [Snowflake](#) to practice test-driven development (TDD), although it was only feasible for simple transformations. The library certainly has rough edges when it comes to debugging failed test runs, but the ability to write unit tests on transformers as we develop the model provided a neat developer experience.

83. Jetpack CameraViewfinder

Trial

When adding camera capabilities to Android apps, developers had to look out for pitfalls. The recently introduced [Jetpack CameraViewfinder](#) API significantly improves the developer experience in this area. Internally it uses either a [TextureView](#) or [SurfaceView](#) to display the camera feed and applies transformations to correctly display the viewfinder, fixing aspect ratio, scale and rotation where necessary. Optimized layouts for foldable devices are provided, too. While not a major feature, we highlight it here to ensure teams are aware of its existence.

84. Jetpack DataStore

Trial

Jetpack DataStore is a new data storage solution to store data asynchronously, consistently and transactionally. It has two implementations: Preferences DataStore for untyped key-value pairs and Proto DataStore for complex data types using Protobufs. By default it is used with Kotlin coroutines and Flow but additional support for RXJava 2 and 3 is available. The documentation recommends you consider migrating to DataStore if you're currently using SharedPreferences, and we agree with that recommendation.

85. Mikro ORM

Trial

Mikro ORM is an object-relational mapping (ORM) framework that has an interesting TypeScript-centric approach. By leveraging TypeScript throughout the entire stack, it offers a consistent development experience from browser to backend, making it easier for developers to write and maintain code. Notably, Mikro ORM's performance is excellent, enabling rapid query execution and minimizing latency. While Mikro ORM offers appealing features, it's essential to keep in mind the general caveats associated with object-relational mappers. ORM frameworks are often complex and offer only a leaky abstraction over a relational data store, and so using one is always a balance of trade-offs.

86. Per-app language preferences

Trial

Many people speak more than one language and use different languages in different contexts. Devices and platforms that can run apps generally ask the user to select one language for the system and then make the apps follow suit. For mobile phones, in particular, users may prefer certain apps to run in a language other than the system's language; Apple introduced a per-app language setting in iOS a while ago. Android app developers, however, had to implement a custom solution within their apps if they wanted to provide this option — until now. Android 13 introduced a new system setting, per-app language preferences, and a public API, making it easier for developers to offer this feature. For backward compatibility, equivalent APIs are available in AndroidX via AppCompatDelegate. We encourage developers to replace their custom solutions and instead use this feature in their apps.

87. Quarto

Trial

Quarto is an open-source scientific and technical publishing system. With it, we can build computational notebooks that allow you to write documents in markdown, embed code and emit that code's output into the final document. It can be used to create reproducible and customizable data analysis reports, which can be easily shared in a variety of formats. Our data science teams used Quarto to share data analysis reports containing visualizations (plots) and tables. They liked being able to use R and Python to generate these dynamic reports and then export them as HTML to share with stakeholders. If you're looking to share your research and analysis within or outside of your organization, we recommend evaluating Quarto.

88. River

Trial

At the heart of many approaches to machine learning lies the creation of a model from a set of training data. Once a model is created, it can be used over and over again. However, the world isn't stationary, and often the model needs to change as new data becomes available. Simply re-running the model creation step can be slow and costly. Incremental learning addresses this issue, making it possible to learn from streams of data incrementally to react to change faster. As a bonus, the compute and memory requirements are lower and predictable. Our practical experience with River continues to be positive. Vowpal Wabbit, which can be an alternative, has a much steeper learning curve, and the Scikit-like API offered by River makes River more accessible to data scientists.

89. Stencil

Trial

Stencil is a library that enables developers to build reusable Web Components using well-established tools such as TypeScript, JSX and JSDoc. According to our teams' experiences, Stencil is a very good choice for building platform-agnostic design systems. For the few browsers that don't support modern browser features, Stencil ensures compatibility by polyfilling unsupported features and APIs on demand.

90. Synthetic Data Vault

Trial

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've had good experiences using SDV to generate large data for performance testing. SDV fares well with modeling a single table. However, data generation time increases considerably as the number of tables with foreign key constraints increases. Nonetheless, SDV offers great promise for local performance testing. It's a good tool for synthetic data generation and worth considering for your testing needs.

91. Vitest

Trial

Vitest is a unit testing framework for JavaScript. Up to now, many teams have relied on Jest, but Jest doesn't play well with Vite, a modern front-end build tool. Using Jest and Vite together forced teams to create two pipelines — one for build and development and one for unit testing — which required tedious configuration of the pipelines with duplicate settings. These problems are solved with Vitest. It is designed specifically for Vite and uses Vite as a bundler. As an additional feature, Vitest has Jest-compatible APIs which makes it possible to use Vitest as a drop-in replacement for Jest in various build setups. However, using Vite and Vitest together provides a better developer experience, and although Vitest is fast, in our experience, it isn't necessarily faster than using Jest.

92. .NET 7 Native AOT

Assess

.NET 7 Native AOT is a big step forward in a long line of approaches to deploying .NET applications natively. It does away with IL and JIT at runtime entirely. Introduced in .NET 7, this improvement is particularly significant for running .NET applications in serverless functions. This new deployment option eliminates the cold start issue, which has been a persistent problem for .NET on serverless platforms like AWS Lambda and Azure Functions. With Native AOT, you can generate a smaller deployable binary than previous methods, resulting in faster cold start times. AWS has officially embraced Native AOT, supporting it with their Amazon Lambda Tools. This new deployment option brings .NET 7 on par with TypeScript/JavaScript in terms of cold start times, making it a viable option for organizations with a largely .NET-oriented infrastructure.

93. .NET MAUI

Assess

.NET MAUI is a new cross-platform framework for creating native mobile and desktop apps with C# and XAML. It allows the creation of apps that can run on Android, iOS, macOS and Windows from a single shared codebase. However, as a new technology, the ecosystem around MAUI is not as developed as React Native or other cross-system platforms, and it only supports C#. Additionally, MAUI may face challenges encountered by organizations using Xamarin in the past, including poor cross-platform tooling, mobile integration problems, developer availability and an immature ecosystem.

While Microsoft announced their commitment to MAUI as an open-source, mobile-first framework for mobile development, its success has yet to be proven. If you're already using Xamarin, you may want to assess MAUI as a potential upgrade; however, if C# or Xamarin isn't part of your tool set yet, you may want to approach MAUI with some caution until the technology is more widely adopted and proven in the market.

94. dbt-expectations

Assess

dbt-expectations is an extension package for dbt inspired by Great Expectations. Data quality is an important tenet of data governance, so when it comes to automated data governance, it's important to craft built-in controls that flag anomalies or quality issues in data pipelines. Just as unit tests run in a build pipeline, dbt-expectations makes assertions during the execution of a data pipeline. In the dbt world, you can run Great Expectations-style data quality tests on your warehouse directly within dbt. Our teams have been exploring this, and it made sense to highlight it.

95. Directus

Assess

We've used Directus as a headless content management system (CMS). Although we have options when it comes to headless CMS products, we needed a self-hosted solution with rich digital asset management and content authoring workflows. In this evaluation we find Directus to be a good fit for our needs; we quite like its event-driven data processing and automation via flows.

96. Ferrocene

Assess

The Rust language has been gaining popularity in recent years for its safety, performance and concurrency features. However, certified Rust toolchains have been missing for applications in safety-critical markets like automotive. This gap is being addressed by Ferrocene, a Rust compiler toolchain. Ferrocene promises to be ISO26262 functional safety standard compliant for the electronic systems in road vehicles; an effort to qualify the language and toolchain for use in such domains is already underway. We're excited by its progress and the availability of such safety-compliant tools will certainly speed up the adoption of Rust in the automotive industry.

97. Flutter for embedded

Assess

Flutter for embedded makes it relatively easy to create and maintain a modern UI similar to mobile apps but for embedded systems like human-machine interface (HMI) in cars, refrigerators and other consumer appliances. This is made possible with Flutter now supporting custom embedders, which allows portability to different platforms. The apps are written in the Dart programming language using the Flutter SDK and ecosystem. We've been building prototypes with it — our developers love the dev experience and our customers like the agility, speed and modern user experience that it brings.

98. Fugue

Assess

In data engineering we're seeing a bewildering choice of tools and technologies. For less experienced engineers especially, it can make sense to work with an abstraction layer to get into the tools, to focus on the task at hand without having to learn several technology-specific APIs and to have the option of switching underlying technologies without too much effort. Fugue is such an abstraction layer. It provides a unified interface for distributed computing, which makes it possible to run Python, pandas and SQL code on Spark, Dask, Ray and DuckDB with minimal rewrites. However, if your team has already decided on a set of technologies, and if they're familiar with their APIs and deep into tweaking and tuning their backend systems, such an abstraction layer provides less value in our experience.

99. Galacean Engine

Assess

Galacean Engine is a web- and mobile-first interactive engine, designed to provide a seamless way to render component-based architecture and animation in a mobile-friendly manner. With its focus on lightweight and high-performance rendering, it has become an increasingly popular choice for developers creating engaging mobile games. It's a TypeScript-based engine that developers report outperforms alternatives.

100. LangChain

Assess

LangChain is a framework for building applications with large language models (LLMs). These models have triggered a race to incorporate generative AI in several use cases. However, using these LLMs in isolation may not be enough — you have to combine them with your differentiated assets to build an impactful product. LangChain fills this niche with some neat features, including prompt management, chaining, data augmented generation and a rich set of agents to determine which actions to take and in what order. We expect more tooling and frameworks to evolve with LLMs, and we recommend assessing LangChain.

101. mljar-supervised

Assess

mljar-supervised is an AutoML Python package that assists with understanding and explaining tabular data. Our data science teams are excited about it and use it to automate exploratory data analysis. It abstracts the common way to preprocess the data, construct the machine learning (ML) models and perform hyper-parameters tuning to find the best model. Explainability and transparency are important tenets, and that's where mljar-supervised shines. It allows you to see exactly how the ML pipeline is constructed with a detailed markdown report for each ML model. It's definitely an interesting AutoML package that's worth assessing for your ML needs.

102. nanoGPT

Assess

nanoGPT is a framework for training and fine-tuning medium-sized generative pretrained transformers (GPT). The author, Andrej Karpathy, references Attention is All You Need and OpenAI's GPT-3 papers to build a GPT from scratch using PyTorch. With all the hype around generative AI, we want to highlight nanoGPT for its simplicity and focus on clearly articulating the building blocks of the GPT architecture.

103. pandera

Assess

In previous Radars, we've featured data validation and testing platforms like Great Expectations that can be used to validate assumptions and test the quality of incoming data used for training or classification. Sometimes, though, all you need is a simple code library to implement tests and quality checks directly in pipelines. pandera is a Python library for testing and validating data across a wide range of frame types such as pandas, Dask or PySpark. pandera can implement simple assertions about fields or hypothesis tests based on statistical models. The wide range of supported frame libraries means tests can be written once and then applied to a variety of underlying data formats. pandera can also be used to generate synthetic data to test ML models.

104. Qwik

Assess

One of the challenges of creating a rich, interactive browser-based experience is in minimizing the time from first request to full user interactivity. When starting up, the application may need to download large amounts of JavaScript to the browser or execute a lengthy process to restore application state on the server. [Qwik](#) is a new front-end framework that serializes application state so it can be rendered on the server without rehydrating and replaying application logic. This is achieved through *resumability*, which involves pausing execution on the server to resume it on the client. Like other newer front-end frameworks, such as [Astro](#) or [Svelte](#), Qwik also speeds up initial page load times by minimizing the amount of JavaScript to load. In Qwik's case, the initial application download is primarily HTML, with most JavaScript loaded dynamically on demand from a local cache, if possible.

105. SolidJS

Assess

[SolidJS](#) is a declarative JavaScript library for creating user interfaces. In the last year, we've seen an increase in SolidJS's visibility and popularity among developers, particularly those interested in creating richer user interactions. SolidJS compiles its templates to real DOM nodes (instead of using vDOM) and updates them with fine-grained reactions which reduces unnecessary DOM updates and results in faster performance and a better user experience. It has a simple API and great support for [TypeScript](#), which can help catch errors during development. Another benefit of SolidJS is its small bundle size, which is ideal for building fast and lightweight web applications and benefits a mobile-first approach. SolidJS is a relatively new framework, so it doesn't have as large of a community or ecosystem as other frameworks. However, judging by the growing number of useful libraries and tools, it seems to be growing in popularity. Its reactive update system, functional component model and templating system make SolidJS an attractive choice to assess, and we're seeing interest from several teams and communities.

106. Turborepo

Assess

One of the topics that seems to perennially draw interest in our discussions is the issue of monorepos. Some places have embraced them for the whole organization, while others have applied the concept in certain narrow applications such as mobile applications or combined UI/BFF development. Regardless of whether or where monorepos are appropriate, the industry seems to be revisiting tools that can effectively manage large codebases and build them efficiently into deployable units. [Turborepo](#) is a relatively new tool in this category that offers an alternative to [Nx](#) or [Lerna](#) for large JavaScript or TypeScript codebases. One of the challenges with large repos is executing builds quickly enough that they don't interrupt developer flow or reduce efficiency. Turborepo is written in [Rust](#) which makes it highly performant; it also builds incrementally and caches intermediate steps to speed things up further. However, it does require changes to the developer workflow that take time to learn and is probably best suited to large codebases with multiple independent builds where a different approach is warranted. We've found that the documentation is sparse, leading some teams to stick with more established tools for now. However, it's worth assessing and seeing if Turborepo and its newer companion, [Turbopack](#) (currently in beta), continue to evolve.

107. WebXR Device API

Assess

When working on the [WebVR](#) experimental API, it became clear that it would make more sense to have a combined API for VR and AR. Rather than changing the WebVR API significantly, a new specification was created: WebXR. At its core is the [WebXR Device API](#) which provides key capabilities for writing VR and AR applications in a web browser. The API is [extensive](#), and at the time of writing it isn't [fully supported](#) by all browsers. Our teams have used WebXR on several occasions, and we see the [benefits described by the Immersive Web Working Group](#). For prototypes, we especially like that the experience is available immediately in a web browser. The development team doesn't have to go through an app-store process, and users can play with the experience without having to install an app. Given the status of the API and the fact it's hidden behind a feature toggle in some browsers, we haven't seen it used beyond proofs of concept and prototypes.

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 12,500+ 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.

 **thoughtworks**