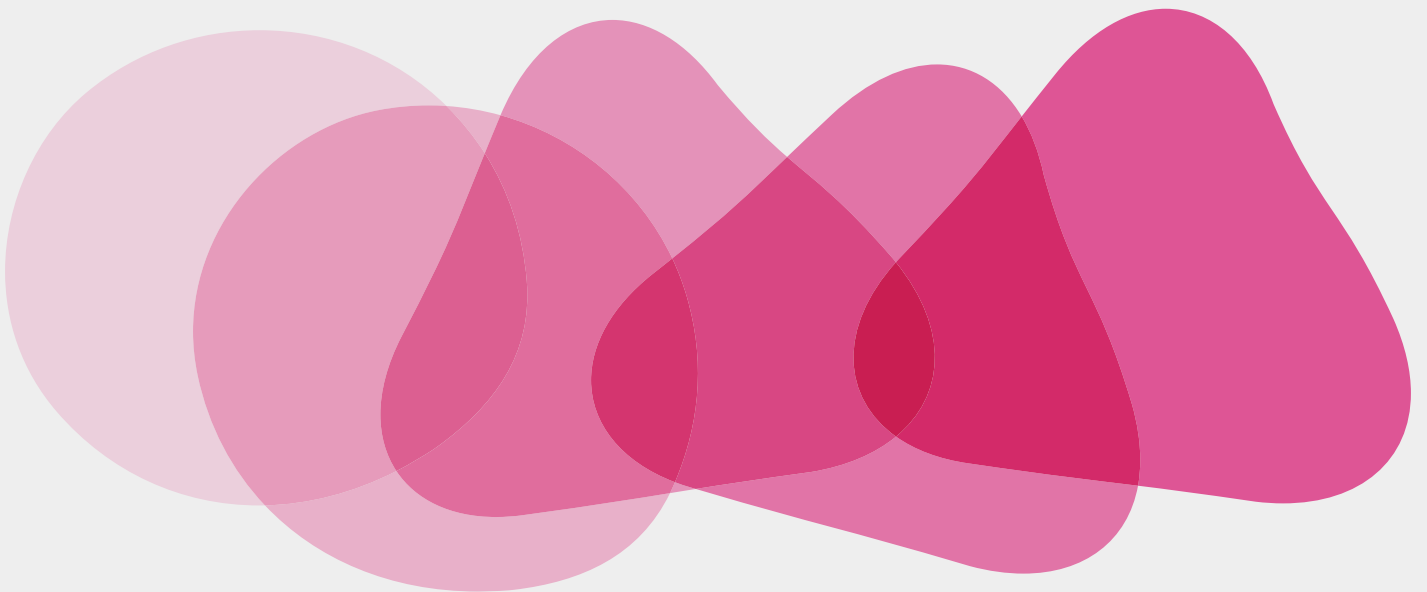


ThoughtWorks®

# TECHNOLOGY RADAR *VOL.19*

Our thoughts on the  
technology and trends that  
are shaping the future



[thoughtworks.com/radar](https://thoughtworks.com/radar)

#TWTechRadar

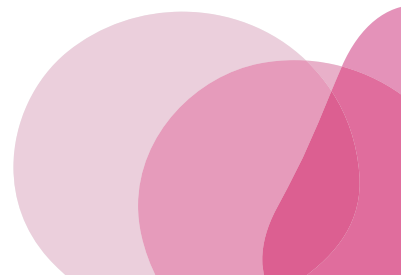
# CONTRIBUTORS

*The Technology Radar is prepared by the  
ThoughtWorks Technology Advisory Board, comprised of:*



[Rebecca Parsons \(CTO\)](#) | [Martin Fowler \(Chief Scientist\)](#) | [Bharani Subramaniam](#) | [Camilla Crispim](#) | [Erik Doernenburg](#)  
[Evan Bottcher](#) | [Fausto de la Torre](#) | [Hao Xu](#) | [Ian Cartwright](#) | [James Lewis](#) | [Jonny LeRoy](#)  
[Ketan Padegaonkar](#) | [Lakshminarasimhan Sudarshan](#) | [Marco Valtas](#) | [Mike Mason](#) | [Neal Ford](#)  
[Ni Wang](#) | [Rachel Laycock](#) | [Scott Shaw](#) | [Shangqi Liu](#) | [Zhamak Dehghani](#)

This edition of the ThoughtWorks Technology Radar is based on a meeting of the Technology Advisory Board in Atlanta in October 2018





# WHAT'S NEW?

*Highlighted themes in this edition*

## STICKY CLOUDS

Cloud providers know they're competing in a tight market, and to succeed, they need to sign up and retain long-term customers. Thus, to stay competitive, they race to add new features, and we see them reaching feature parity, which is reflected in our placing of [AWS](#), [GCP](#), and [Azure](#) in the Trial ring in this edition. However, once customers sign up, these providers tend to create as sticky a connection as possible with their customers to discourage roaming to another provider. Often this manifests in a strong dependency on their specific suite of services and tools, offering a better developer experience as long as customers stay with them. Some companies are taken by surprise when the stickiness becomes apparent, often at the time of making a choice to move parts or all of their workloads to another cloud or finding their cloud usage and payments spiraling out of control. We encourage our clients to use either the [run cost as architecture fitness function](#) technique to monitor the cost of operation as an indicator of stickiness or [Kubernetes and containers](#) to increase workload portability and reduce the cost of change to another cloud through [infrastructure as code](#). In this Radar, we also introduce two new cloud infrastructure automation tools, [Terragrunt](#) and [Pulumi](#). While we support assessing the new offerings of your cloud provider through the lens of stickiness, we caution against generic cloud usage. In our experience, the overhead of creating and maintaining cloud-agnostic abstraction layers outweigh the exit cost for a particular provider.

## LINGERING ENTERPRISE ANTIPATTERNS

No matter how fast technology changes, enterprises still find ways to reimplement antipatterns from the past. Many of our Hold entries decry an old wolf hiding in new sheep's clothing: enterprise service bus (ESB) behavior implemented on event-streaming platforms—[Recreating ESB antipatterns with Kafka](#), [Layered microservices architecture](#), [Data-hungry packages](#), [Overambitious API gateways](#), [Low-code platforms](#) and other noxious old practices. The fundamental problem, as always, is the balance between isolation and coupling: we isolate things to make them manageable from a technical perspective, but then we need to add coordination to make them useful for solving business problems, resulting in some form of coupling. Thus, these old patterns keep re-emerging. New architectures and tools provide appropriate ways to solve these problems, but that requires deliberate effort to understand how to use them appropriately and how to avoid falling back to reimplementing old patterns with shiny new technology.

## ENDURING ENGINEERING PRACTICES

One side effect of the increased pace of technological innovation is a repeating expansion and contraction pattern. When an innovation appears that fundamentally changes how we think about some aspect of software development, the industry races to adopt it: containerization, reactive frontends, machine learning and so on. That's the expansion phase. However, to make this "new thing" truly effective requires figuring out how to apply enduring engineering practices to it: continuous delivery, testing, collaboration and so on. The contraction phase occurs as we ascertain how to use this new capability effectively, creating a firm foundation to allow the next explosive expansion. During this phase we learn how to apply practices such as comprehensive automated testing and the scripting of sequences of recurring steps within the context of the new technology. Often, this goes hand in hand with the creation of new development tools. While it may seem that the introduction of a new technological innovation alone advances our industry, it's the combination of innovation with enduring engineering practices that underpins our continued progress.

## PACE = DISTANCE / TIME

Our themes usually highlight a pattern we've seen over a handful of entries in the current Radar, but this one concerns all the entries over the lifetime of the Radar. We've noticed (and we've done some research to back it up) that the length of time our blips remain in the Radar is falling. When we started the Radar a decade ago, the default for entries was to remain for two Radar editions (approximately one year) with no movement before they fade away automatically. However, as indicated by the formula in this theme's title, *pace = distance over time*: change in the software development ecosystem continues to accelerate. Time has remained constant (we still create the Radar twice a year), but the distance traveled in terms of technology innovation has noticeably increased, providing yet more evidence of what's obvious to any astute observer: the pace of technology change continues to increase. We see increased pace in all our Radar quadrants and also in our client's appetite to adopt new and diverse technology choices. Consequently, we modified our traditional default for this Radar: now, each entry must appear in the Radar based on its current merit—we no longer allow them to stay by default. We made this change after careful consideration, feeling that it allows us to better capture the frenetic pace of change ever present in the technology ecosystem.

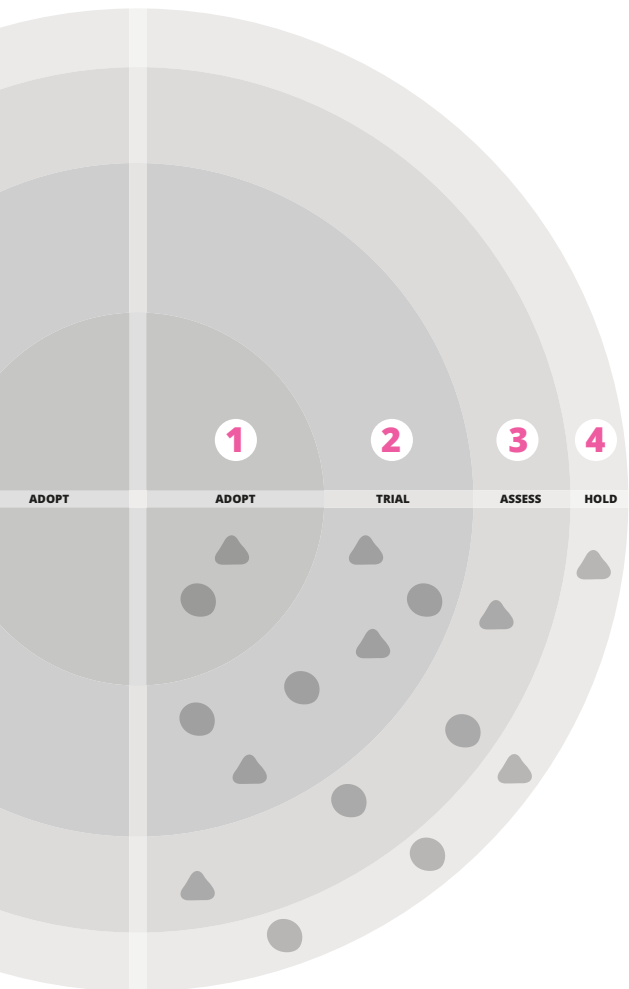
# ABOUT THE RADAR

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 for more detail. 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](https://thoughtworks.com/radar/faq)*



## RADAR AT A GLANCE

### 1 ADOPT

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

### 2 TRIAL

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

### 3 ASSESS

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

### 4 HOLD

Proceed with caution.

### ▲ NEW OR CHANGED

Items that are new or have had significant changes since the last Radar are represented as triangles, while items that have not changed are represented as circles.

### ● 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 our limited Radar real estate.

# THE RADAR

## TECHNIQUES

### ADOPT

1. Event Storming

### TRIAL

2. 1% canary *NEW*
3. Bounded Buy *NEW*
4. Crypto shredding *NEW*
5. Four key metrics *NEW*
6. Multi-account cloud setup *NEW*
7. Observability as code *NEW*
8. Risk-commensurate vendor strategy *NEW*
9. Run cost as architecture fitness function *NEW*
10. Secrets as a service *NEW*
11. Security Chaos Engineering
12. Versioning data for reproducible analytics *NEW*

### ASSESS

13. Chaos Katas *NEW*
14. Distrosless Docker images *NEW*
15. Incremental delivery with COTS *NEW*
16. Infrastructure configuration scanner
17. Pre-commit downstream build checks *NEW*
18. Service mesh

### HOLD

19. "Handcranking" of Hadoop clusters using config management tools *NEW*
20. Generic cloud usage
21. Layered microservices architecture *NEW*
22. Master data management *NEW*
23. Microservice envy
24. Request-response events in user-facing workflows *NEW*
25. RPA *NEW*

## PLATFORMS

### ADOPT

### TRIAL

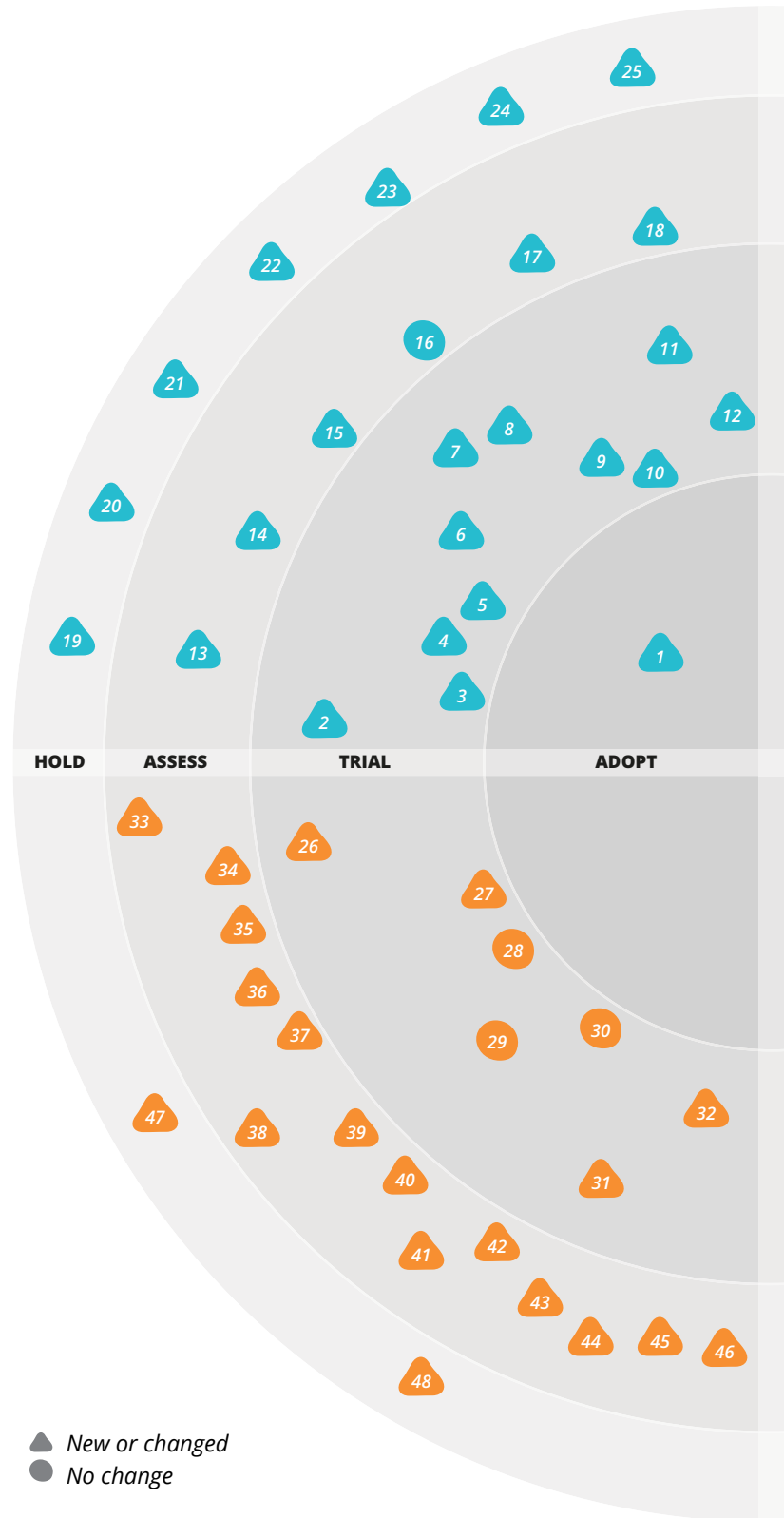
26. Apache Atlas *NEW*
27. AWS
28. Azure
29. Contentful
30. Google Cloud Platform
31. Shared VPC *NEW*
32. TICK Stack

### ASSESS

33. Azure DevOps *NEW*
34. CockroachDB *NEW*
35. Debezium *NEW*
36. Glitch *NEW*
37. Google Cloud Dataflow *NEW*
38. gVisor *NEW*
39. IPFS *NEW*
40. Istio *NEW*
41. Knative *NEW*
42. Pulumi *NEW*
43. Quorum *NEW*
44. Resin.io *NEW*
45. Rook *NEW*
46. SPIFFE *NEW*

### HOLD

47. Data-hungry packages *NEW*
48. Low-code platforms *NEW*



# THE RADAR

## TOOLS

### ADOPT

#### TRIAL

- 49. acs-engine *NEW*
- 50. Archery *NEW*
- 51. ArchUnit
- 52. Cypress
- 53. git-secrets *NEW*
- 54. Headless Firefox
- 55. LocalStack *NEW*
- 56. Mermaid *NEW*
- 57. Prettier *NEW*
- 58. Rider *NEW*
- 59. Snyk *NEW*
- 60. UI dev environments *NEW*
- 61. Visual Studio Code
- 62. VS Live Share *NEW*

#### ASSESS

- 63. Bitrise *NEW*
- 64. Codefresh *NEW*
- 65. Grafeas *NEW*
- 66. Heptio Ark *NEW*
- 67. Jaeger *NEW*
- 68. kube-bench *NEW*
- 69. Ocelot *NEW*
- 70. Optimal Workshop *NEW*
- 71. Stanford CoreNLP *NEW*
- 72. Terragrunt *NEW*
- 73. TestCafe *NEW*
- 74. Traefik *NEW*
- 75. Wallaby.js *NEW*

### HOLD

## LANGUAGES & FRAMEWORKS

### ADOPT

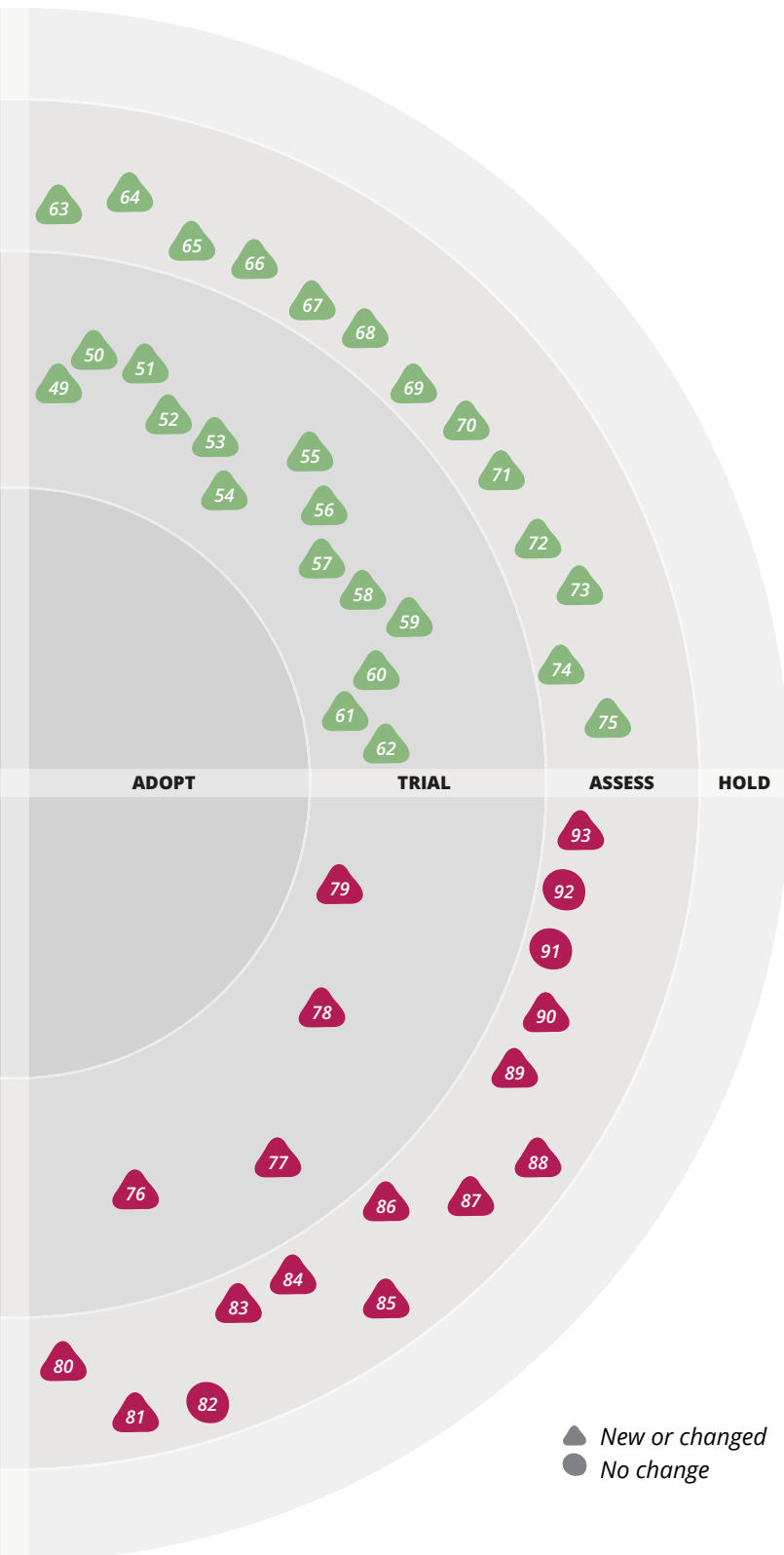
#### TRIAL

- 76. Jepsen
- 77. MMKV *NEW*
- 78. MockK *NEW*
- 79. TypeScript

#### ASSESS

- 80. Apache Beam *NEW*
- 81. Camunda *NEW*
- 82. Flutter
- 83. Ktor *NEW*
- 84. Nameko *NEW*
- 85. Polly.js *NEW*
- 86. PredictionIO *NEW*
- 87. Puppeteer *NEW*
- 88. Q# *NEW*
- 89. SAFE stack *NEW*
- 90. Spek *NEW*
- 91. troposphere
- 92. WebAssembly
- 93. WebFlux *NEW*

### HOLD



# TECHNIQUES

## ADOPT

1. Event Storming

## TRIAL

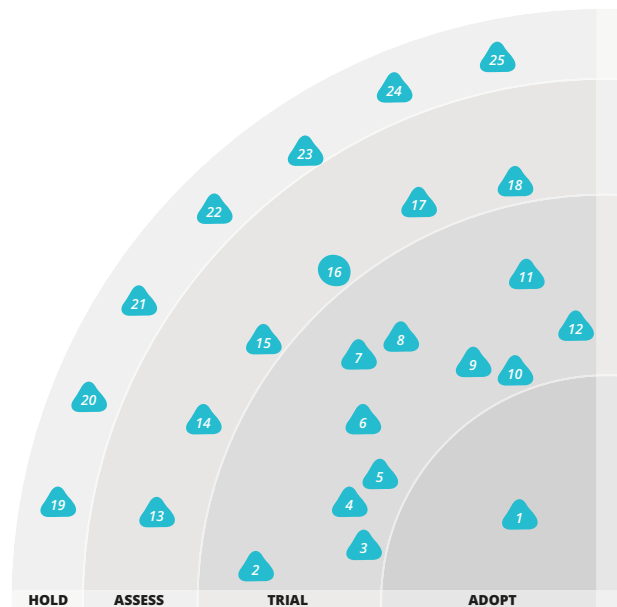
2. 1% canary **NEW**
3. Bounded Buy **NEW**
4. Crypto shredding **NEW**
5. Four key metrics **NEW**
6. Multi-account cloud setup **NEW**
7. Observability as code **NEW**
8. Risk-commensurate vendor strategy **NEW**
9. Run cost as architecture fitness function **NEW**
10. Secrets as a service **NEW**
11. Security Chaos Engineering
12. Versioning data for reproducible analytics **NEW**

## ASSESS

13. Chaos Katas **NEW**
14. Distroless Docker images **NEW**
15. Incremental delivery with COTS **NEW**
16. Infrastructure configuration scanner
17. Pre-commit downstream build checks **NEW**
18. Service mesh

## HOLD

19. "Handcranking" of Hadoop clusters using config management tools **NEW**
20. Generic cloud usage
21. Layered microservices architecture **NEW**
22. Master data management **NEW**
23. Microservice envy
24. Request-response events in user-facing workflows **NEW**
25. RPA **NEW**



When organizations move toward microservices, one of the main drivers is the hope for faster time to market. However, this aspiration only tends to be realized when services (and their supporting teams) are cleanly sliced along long-lived business domain boundaries. Otherwise meaningful features will naturally require tight coordination between multiple teams and services, introducing natural friction in competing roadmap prioritization. The solution to this problem is good domain modeling, and **EVENT STORMING** has rapidly become one of our favorite methods for rapidly identifying the key concepts in a problem space and aligning a variety of stakeholders in the best way to slice potential solutions.

Fast feedback is one of our core values for building software. For many years, we've used the canary release approach to encourage early feedback on new software versions, while reducing the risk through incremental rollout to selected users. One of the questions regarding this technique is how to segment

users. Canary releases to a very small segment (say 1%) of users can be a catalyst for change. While starting with a very small segment of users enables teams to get comfortable with the technique, capturing fast user feedback enables diverse teams to observe the impact of new releases and learn and adjust course as necessary—a priceless change in engineering culture. We call this, the mighty **1% CANARY**.

*Out-of-the-box or SaaS solutions tend to aggressively expand their scope to entangle themselves into every part of your business. We recommend a strategy to only select vendor products that are modular and decoupled and can be contained within the Bounded Context of a single business capability.*

(Bounded Buy)



Most organizations that don't have the resources to custom-build their software will select out-of-the-box or SaaS solutions to meet their requirements. All too often, however, these solutions tend to aggressively expand their scope to entangle themselves into every part of your business. This blurs integration boundaries and makes change less predictable and slow. To mitigate this risk, we recommend that organizations develop a clear target capability model and then employ a strategy we call **BOUNDED BUY**—that is, only select vendor products that are modular and decoupled and can be contained within the Bounded Context of a single business capability. This modularity and independent deliverability should be included in the acceptance criteria for a vendor selection process.

Maintaining proper control over sensitive data is difficult, especially when—for backup and recovery purposes—data is copied outside of a master system of record.

**CRYPTO SHREDDING** is the practice of rendering sensitive data unreadable by deliberately overwriting or deleting encryption keys used to secure that data. For example, an entire table of customer personal details could be encrypted using random keys for each record, with a different table storing the keys. If a customer exercised their “right to be forgotten,” we can simply delete the appropriate key, effectively “shredding” the encrypted data. This technique can be useful where we're confident of maintaining appropriate control of a smaller set of encryption keys but less confident about control over a larger data set.

*The State of DevOps report and subsequent Accelerate book highlight a surprising fact: in order to predict and improve the performance of a team, we only need to measure lead time, deployment frequency, mean time to restore (MTTR), and change fail percentage.*

(Four key metrics)

The State of DevOps report, first published in 2014, states that high-performing teams create high-

performing organizations. Recently, the team behind the report released Accelerate, which describes the scientific method they've used in the report. A key takeaway of both are the **FOUR KEY METRICS** to support software delivery performance: lead time, deployment frequency, mean time to restore (MTTR), and change fail percentage. As a consultancy that has helped many organizations transform, these metrics have come up time and time again as a way to help organizations determine whether they're improving the overall performance. Each metric creates a virtuous cycle and focuses the teams on continuous improvement: to reduce lead time, you reduce wasteful activities which, in turn, lets you deploy more frequently; deployment frequency forces your teams to improve their practices and automation; your speed to recover from failure is improved by better practices, automation and monitoring which reduces the frequency of failures.

On-demand self-service is a key characteristic (and benefit) of cloud computing. When large-scale service landscapes are deployed using a single account, rules and processes around usage of that account become necessary, often involving approval steps that increase turnaround time. A better approach is a **MULTI-ACCOUNT CLOUD SETUP** where several accounts are used, in the extreme one account per team. This does add overhead in other places, for example, ensuring shared billing, enabling communication between VPCs and managing the relationship with the cloud provider. However, it often accelerates development and it usually improves security, because single-service accounts are easier to audit and, in the case of a breach, the impact is greatly reduced. Having multiple accounts also reduces stickiness, because an account provides a good boundary for services that can be moved en bloc to another cloud provider.

*Observability is an integral part of operating a distributed and microservices-based architecture. We recommend treating your observability ecosystem configurations as code.*

(Observability as code)



The observability is an integral part of operating a distributed and [microservices architecture](#). We rely on different system outputs such as distributed tracing, aggregate logs and metrics to infer the internal state of the distributed components, diagnose where the problems are and get to the root cause. An important aspect of an observability ecosystem is monitoring—visualizing and analyzing the system's output—and alerting when unexpected conditions are detected. Traditionally, configuration of monitoring dashboards and setting up alerts is done through GUI-based point-and-click systems. This approach leads to nonrepeatable dashboard configurations, no ability to continuously test and adjust alerts to avoid alert fatigue or missing out on important alerts, and drift from organizational best practices. We highly recommend treating your observability ecosystem configurations as code, called **OBSERVABILITY AS CODE**, and adopt [infrastructure as code](#) for your monitoring and alerting infrastructure. Choose observability products that support configuration through version-controlled code and execution of APIs or commands via infrastructure CD pipelines. Observability as code is an often-forgotten aspect of infrastructure as code and, we believe, crucial enough to be called out.

Often, in an effort to outsource risk to their suppliers, businesses look for “one throat to choke” on their most critical and risky system implementations. Unfortunately, this gives them fewer solution choices and less flexibility. Instead, businesses should look to maintain the greatest vendor independence where the business risk exposure is highest. We see a new **RISK-COMMENSURATE VENDOR STRATEGY** emerging that encourages investment to maintain vendor independence for highly critical business systems. Less critical business functions can take advantage of the streamlined delivery of a vendor-native solution because it allows them to absorb more easily the impact of losing that vendor. This trade-off has become apparent as the major cloud providers have expanded their range of service offerings. For example, using AWS Secret Management Service can speed up initial development and has the benefit of ecosystem integration, but it will also add more inertia if you ever need to migrate to a different cloud provider than it would if you had implemented, for example, [Vault](#).

We still see teams who aren't tracking the cost of running their applications as closely as they should as their software architecture or usage evolves. This is particularly true when they're using [serverless](#), which developers assume will provide lower costs since you're not paying for unused server cycles. However, the major cloud providers are pretty savvy at setting their pricing models, and heavily used serverless functions, although very useful for rapid iteration, can get expensive quickly when compared with dedicated cloud (or on-premise) servers. We advise teams to frame a system's **RUN COST AS ARCHITECTURE FITNESS FUNCTION**, which means: track the cost of running your services against the value delivered; when you see deviations from what was expected or acceptable, have a discussion about whether it's time to evolve your architecture.

We've long cautioned people about the temptation to check secrets into their source code repositories. Previously, we've recommended [decoupling secret management from source code](#). However, now we're seeing a set of good tools emerge that offer **SECRETS AS A SERVICE**. With this approach, rather than hardwiring secrets or configuring them as part of the environment, applications retrieve them from a separate process. Tools such as [Vault](#) by HashiCorp let you manage secrets separately from the application and enforce policies such as frequent rotation externally.

Although we've had mostly new blips in this edition of the Radar, we think it's worth continuing to call out the usefulness of **SECURITY CHAOS ENGINEERING**. We've moved it to Trial because the teams using this technique are confident that the security policies they have in place are robust enough to handle common security failure modes. Still, proceed with caution when using this technique—we don't want our teams to become desensitized to these issues.


*Versioning data for large-scale data analysis or machine intelligence problems allows us to reproduce different versions of analysis done on different data sets and parameters, and is immensely valuable.*

(Versioning data for reproducible analytics)

When it comes to large-scale data analysis or machine intelligence problems, being able to reproduce different versions of analysis done on different data sets and parameters is immensely valuable. To achieve reproducible analysis, both the data and the model (including algorithm choice, parameters and hyperparameters) need to be version controlled.

#### **VERSIONING DATA FOR REPRODUCIBLE ANALYTICS**


is a relatively trickier problem than versioning models because of the data size. Tools such as DVC help in versioning data by allowing users to commit and push data files to a remote cloud storage bucket using a git-like workflow. This makes it easy for collaborators to pull a specific version of data to reproduce an analysis.



*Repeated execution of Katas helps engineers to internalize their new skills. We combine the discipline of Katas with Chaos Engineering techniques to help engineers discover problems, recover from failure and find root causes.*

(Chaos Katas)

**CHAOS KATAS** is a technique that our teams have developed to train and upskill infrastructure and platform engineers. It combines Chaos Engineering techniques—that is, creating failures and outages in a controlled environment—with the systematic teaching and training approach of Kata. Here, Kata refers to code patterns that trigger controlled failures, allowing engineers to discover the problem, recover from the failure, run postmortem and find the root cause. Repeated execution of Katas helps engineers to internalize their new skills.



*When building Docker images for our applications, we're often concerned with two things: the security and the size of the image. With this technique, the footprint of the image is reduced to the application, its resources and language runtime dependencies, without operating system distribution.*

(Distroless Docker images)

When building Docker images for our applications, we're often concerned with two things: the security and the size of the image. Traditionally, we've used container

security scanning tools to detect and patch common vulnerabilities and exposures and small distributions such as Alpine Linux to address the image size and distribution performance. In this Radar, we're excited about addressing the security and size of containers with a new technique called **DISTROLESS DOCKER IMAGES**, pioneered by Google. With this technique, the footprint of the image is reduced to the application, its resources and language runtime dependencies, without operating system distribution. The advantages of this technique include reduced noise of security scanners, smaller security attack surface, reduced overhead of patching vulnerabilities and even smaller image size for higher performance. Google has published a set of distroless container images for different languages. You can create distroless application images using the Google build tool Bazel, which has rules for creating distroless containers or simply use multistage Dockerfiles. Note that distroless containers by default don't have a shell for debugging. However, you can easily find debug versions of distroless containers online, including a busybox shell.

At ThoughtWorks, as early adopters and leaders in the agile space, we've been proponents of the practice of incremental delivery. We've also advised many clients to look at off-the-shelf software through a "Can this be released incrementally?" lens. This has often been difficult because of the big-bang approach of most vendors which usually involves migrating large amounts of data. Recently, however, we've also had success using **INCREMENTAL DELIVERY WITH COTS** (commercial off-the-shelf), launching specific business processes incrementally to smaller subsets of users. We recommend you assess whether you can apply this practice to the vendor software of your choice, to help reduce the risks involved in big-bang deliveries.

*Most organizations deploy tightly locked-down and centrally managed cloud configurations to reduce risk, but this creates substantial productivity bottlenecks. An alternative approach is to allow teams to manage their own configuration and use a tool to ensure the configuration is set in a safe and secure way.*

(Infrastructure configuration scanner)

For some time now we've recommended increased

delivery team ownership of their entire stack, including infrastructure. This means increased responsibility in the delivery team itself for configuring infrastructure in a safe, secure, and compliant way. When adopting cloud strategies, most organizations default to a tightly locked-down and centrally managed configuration to reduce risk, but this also creates substantial productivity bottlenecks. An alternative approach is to allow teams to manage their own configuration, and use an **INFRASTRUCTURE CONFIGURATION SCANNER** to ensure the configuration is set in a safe and secure way. Watchmen is an interesting tool, built to provide rule-driven assurance of AWS account configurations that are owned and operated independently by delivery teams. Scout2 is another example of configuration scanning to support secure compliance.

In more complex architectures and deployments, it may not be immediately obvious that a build that depends on the code currently being checked in is broken. Developers trying to fix a broken build could find themselves working against a moving target, as the build is continually triggered by upstream dependencies. **PRE-COMMIT DOWNSTREAM BUILD CHECKS** is a very simple technique: have a pre-commit or pre-push script check the status of these downstream builds and alert the developer beforehand that a build is broken.

As large organizations transition to more autonomous teams owning and operating their own microservices, how can they ensure the necessary consistency and compatibility between those services without relying on a centralized hosting infrastructure? To work together efficiently, even autonomous microservices need to align with some organizational standards. A **SERVICE MESH** offers consistent discovery, security, tracing, monitoring and failure handling without the need for a shared asset such as an API gateway or ESB. A typical implementation involves lightweight reverse-proxy processes deployed alongside each service process, perhaps in a separate container. These proxies communicate with service registries, identity providers, log aggregators and other services. Service interoperability and observability are gained through a shared implementation of this proxy but not a shared runtime instance. We've advocated for a decentralized approach to microservices management for some time and are happy to see this consistent pattern emerge. Open source projects such as Linkerd and Istio will continue to mature and make service meshes even easier to implement.

When organizations choose a vanilla Hadoop or Spark distribution instead of one of the vendor distributions, they have to decide how they want to provision and manage the cluster. Occasionally, we see **"HANDCRANKING" OF HADOOP CLUSTERS USING CONFIG MANAGEMENT TOOLS** such as Ansible, Chef and others. Although these tools are great at provisioning immutable infrastructure components, they're not very useful when you have to manage stateful systems and can often lead to significant effort trying to manage and evolve clusters using these tools. We instead recommend using tools such as Ambari to provision and manage your stateful Hadoop or Spark clusters.

*Increasingly, we're seeing organizations prepare to use "any cloud" and to avoid vendor lock-in at all costs. Their strategy limits the use of the cloud to only those features common across all cloud providers—thereby missing out on the providers' unique benefits.*

(Generic cloud usage)

The major cloud providers have become increasingly competitive in their pricing and the rapid pace of releasing new features. This leaves consumers in a difficult place when choosing and committing to a provider. Increasingly, we're seeing organizations prepare to use "any cloud" and to avoid vendor lock-in at all costs. This, of course, leads to **GENERIC CLOUD USAGE**. We see organizations limiting their use of the cloud to only those features common across all cloud providers—thereby missing out on the providers' unique benefits. We see organizations making large investments in home-grown abstraction layers that are too complex to build and too costly to maintain to stay cloud agnostic. The problem of lock-in is real. We recommend approaching this problem with a multicloud strategy that evaluates the migration cost and effort of capabilities from one cloud to another against the benefits of using cloud-specific features. We recommend increasing the portability of the workloads by shipping the applications as widely adopted Docker containers: use open source security and identity protocols to easily migrate the identity of the workloads, a risk-commensurate vendor strategy to maintain cloud independence only where necessary and Polycloud to mix and match services from different providers where it makes sense. In short, shift your approach from a generic cloud usage to a sensible multicloud strategy.

A defining characteristic of a microservices architecture is that system components and services are organized around business capabilities. Regardless of size, microservices encapsulate a meaningful grouping of functionality and information to allow for the independent delivery of business value. This is in contrast to earlier approaches in service architecture which organized services according to technical characteristics. We've observed a number of organizations who've adopted a **LAYERED MICROSERVICES ARCHITECTURE**, which in some ways is a contradiction in terms. These organizations have fallen back to arranging services primarily according to a technical role, for example, experience APIs, process APIs or system APIs. It's too easy for technology teams to be assigned by layer, so delivering any valuable business change requires slow and expensive coordination between multiple teams. We caution against the effects of this layering and recommend arranging services and teams primarily according to business capability.

**MASTER DATA MANAGEMENT (MDM)** is a classic example of the enterprise "silver bullet" solution: it promises to solve an apparently related class of problems in one go. Through creating a centralized single point of change, coordination, test and deployment, MDM solutions negatively impact an organization's ability to respond to business change. Implementations tend to be long and complex, as organizations try to capture and map all "master" data into the MDM while integrating the MDM solution into all consuming or producing systems.

Microservices has emerged as a leading architectural technique in modern cloud-based systems, but we still think teams should proceed carefully when making this choice. **MICROSERVICE ENVY** tempts teams to complicate their architecture by having lots of services simply because it's a fashionable architecture choice. Platforms such as Kubernetes make it much easier to deploy complex sets of microservices, and vendors are pushing their solutions to managing microservices, potentially leading teams further down this path. It's important to remember that microservices trade development complexity for operational complexity and require a solid foundation of automated testing, continuous delivery and DevOps culture.

On a number of occasions we have seen system designs that use **REQUEST-RESPONSE EVENTS IN USER-FACING WORKFLOWS**. In these cases, the UI is blocked or the user has to wait for a new page to load until a corresponding response message to a request message is received. The main reasons cited for designs like this are performance or a unified approach to communication between backends for synchronous and asynchronous use cases. We feel that the increased complexity—in development, testing and operations—far outweighs the benefit of having a unified approach, and we strongly suggest to use synchronous HTTP requests when synchronous communication between backend services is needed. When implemented well, communication using HTTP rarely is a bottleneck in a distributed system.

*The problem with focusing only on automating business processes—without addressing the underlying software systems or capabilities—is that this can make it even harder to change those underlying systems. RPA tends to introduce additional coupling, making any future attempts to address the legacy IT landscape even more difficult.*

(RPA)

Robotic process automation (**RPA**) is a key part of many digital transformation initiatives, as it promises to deliver cost savings without having to modernize the underlying architecture and systems. The problem with this approach of focusing only on automating business processes, without addressing the underlying software systems or capabilities, is that this can make it even harder to change the underlying systems by introducing additional coupling. This makes any future attempts to address the legacy IT landscape even more difficult. Very few systems can afford to ignore change and hence RPA efforts need to be coupled with appropriate legacy modernization strategies.

# PLATFORMS

## ADOPT

### TRIAL

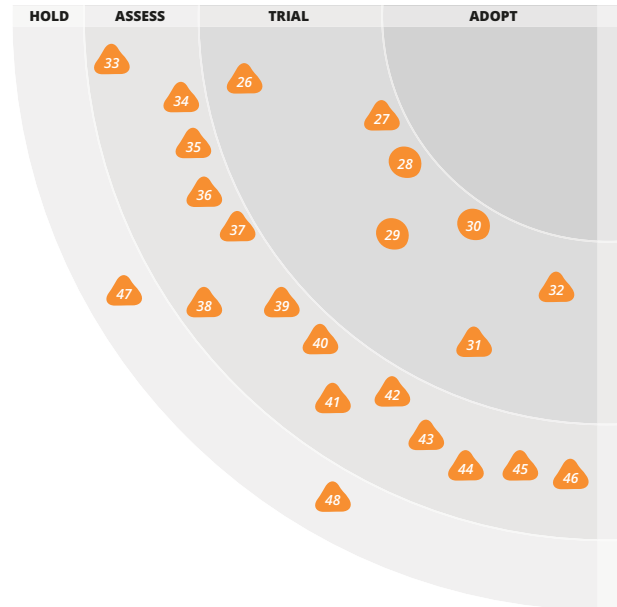
- 26. Apache Atlas **NEW**
- 27. AWS
- 28. Azure
- 29. Contentful
- 30. Google Cloud Platform
- 31. Shared VPC **NEW**
- 32. TICK Stack

### ASSESS

- 33. Azure DevOps **NEW**
- 34. CockroachDB **NEW**
- 35. Debezium **NEW**
- 36. Glitch **NEW**
- 37. Google Cloud Dataflow **NEW**
- 38. gVisor **NEW**
- 39. IPFS **NEW**
- 40. Istio **NEW**
- 41. Knative **NEW**
- 42. Pulumi **NEW**
- 43. Quorum **NEW**
- 44. Resin.io **NEW**
- 45. Rook **NEW**
- 46. SPIFFE **NEW**

### HOLD

- 47. Data-hungry packages **NEW**
- 48. Low-code platforms **NEW**



*Metadata management needs are growing. Atlas provides the capability to model types for metadata, classify data assets, track data lineage and enable data discovery. This fits enterprise data governance needs.*

(Apache Atlas)

With the growing and diverse data needs of enterprises comes a growing need for metadata management.

**APACHE ATLAS** is a metadata management framework that fits the data governance needs of enterprises. Atlas provides capabilities to model types for metadata, classify data assets, track the data lineage and enable data discovery. However, when building a metadata management platform, we need to be careful not to repeat the mistakes of master data management.

We first placed **AWS** in Adopt seven years ago, and the breadth, depth and reliability of its services have improved in leaps and bounds since then. However, we're now moving AWS back into Trial, not because of any deficiencies in its offering, but because its competitors, GCP and Azure, have


matured considerably and selecting a cloud provider has become increasingly complex. We reserve Adopt for when we see a clear winner in a field. For many years, AWS was the default choice, but we now feel that organizations should make a balanced selection across cloud providers that takes into account their geographic and regulatory footprint, their strategic alignment (or lack thereof) with the providers, and, of course, the fit between their most important needs and the cloud providers' differentiating products.

Microsoft has steadily improved **AZURE** and today not much separates the core cloud experience provided by the major cloud providers—Amazon, Google and Microsoft. The cloud providers seem to agree and seek to differentiate themselves in other areas such as features, services and cost structure. Microsoft is the provider who shows real interest in the legal requirements of European companies. They've a nuanced and plausible strategy, including unique offerings such as Azure Germany and Azure Stack which gives some certainty to European companies in anticipation of the GDPR and possible legislative changes in the United States.



Headless content management systems (CMSes) are becoming a common component of digital platforms. **CONTENTFUL** is a modern headless CMS that our teams have successfully integrated into their development workflows. We particularly like its API-first approach and implementing CMS as code. It supports powerful content modeling primitives as code and content model evolution scripts, which allow treating it as other data store schemas and applying evolutionary database design practices to CMS development. Other notable features that we've liked include inclusion of two CDNs to deliver media assets and JSON documents, good support for localization and the ability—albeit with some effort—to integrate with Auth0.

As **GOOGLE CLOUD PLATFORM** (GCP) has expanded in terms of available geographic regions and maturity of services, customers globally can now seriously consider it for their cloud strategy. In some areas, GCP has reached feature parity with its main competitor, Amazon Web Services, while in other areas it has differentiated itself—notably with accessible machine learning platforms, data engineering tools and a workable Kubernetes as a service solution (GKE). In practice, our teams have nothing but praise for the developer experience working with the GCP tools and APIs.



*One recommended pattern is to use a virtual private cloud network managed at the organizational level and divided into smaller subnets under the control of each delivery team. Shared VPC makes organizations, projects, VPCs and subnets first-class entities in network configurations—this simplifies configuration and makes security and access control more transparent.*


(Shared VPC)

As we've gained more experience with the public cloud across organizations large and small, certain patterns have emerged. One of those patterns is a virtual private cloud network managed at the organizational level and divided into smaller subnets under the control of each delivery team. This is closely related to the idea

of multi-account cloud setup and helps to partition an infrastructure along team bounds. After configuring this setup many times using VPCs, subnets, security groups and NACLs, we really like Google's notion of the **SHARED VPC**. Shared VPC makes organizations, projects, VPCs and subnets first-class entities in network configurations. VPCs can be managed by an organization's administrators who can delegate subnet administration to projects. Projects can then be explicitly associated with subnets in the VPC. This simplifies configuration and makes security and access control more transparent.

**TICK STACK** is a collection of open source components that combine to deliver a platform for easily storing, visualizing and monitoring time series data such as metrics and events. The components are: Telegraf, a server agent for collecting and reporting metrics; InfluxDB, a high-performance time series database; Chronograf, a user interface for the platform; and Kapacitor, a data-processing engine that can process, stream and batch data from InfluxDB. Unlike Prometheus, which is based on the pull model, TICK Stack is based on the push model of collecting data. The heart of the system is the InfluxDB component, which is one of the best time series databases. The stack is backed by InfluxData and although you need the enterprise version for features such as database clustering, it's still a fairly good choice for monitoring. We're using it in a few places in production and have had good experiences with it.


**AZURE DEVOPS** services include a set of managed services such as hosted Git repos, CI and CD pipelines and artifact repository. Azure DevOps services have replaced Visual Studio Team Services. We've had a good experience in starting projects quickly with Azure DevOps services—managing, building and releasing applications to Azure. We've also run into a few challenges—such as lack of full support for CI and CD pipeline as code, slow build agent startup time, separation of build and release into different pipelines—and experienced a few downtimes. We're hoping that Azure DevOps services improve over time to provide a good developer experience when hosting applications on Azure, with a frictionless experience integrating with other Azure services.



*Inspired by Google's white paper on Spanner—its distributed database based on atomic clocks—CockroachDB is an open-source alternative that provides distributed transactions and geo-partitioning while still supporting SQL.*

(CockroachDB)

**COCKROACHDB** is an open source distributed database inspired by the white paper [Spanner: Google's distributed database](#). In CockroachDB, data is automatically divided into ranges, usually 64MB, and distributed across nodes in the cluster. Each range has a consensus group and, because it uses the [Raft consensus algorithm](#), the data is always kept in sync. With its unique design, CockroachDB provides distributed transactions and geo-partitioning while still supporting SQL. Unlike Spanner, which relies on TrueTime with atomic clock for linearizability, CockroachDB uses NTP for clock synchronization and provides serializability as the default isolation level. If you're working with structured data that fits in a single node, then choose a traditional relational database. However, if your data needs to scale across nodes, be consistent and survive failures, then we recommend you take a closer look at CockroachDB.



*We're always on the lookout for tools or platforms to support Change Data Capture and streaming data updates. Debezium is an excellent choice. It works by reacting to changes in database log files and is highly scalable and resilient to failures.*

(Debezium)

**DEBEZIUM** is a change data capture (CDC) platform that can stream database changes onto [Kafka](#) topics. CDC is a popular technique with multiple use cases, including replicating data to other databases, feeding analytics systems, extracting microservices from monoliths and invalidating caches. We're always on the lookout for tools or platforms in this space (we talked about [Bottled Water](#) in a previous Radar) and Debezium is an excellent choice. It uses a log-based CDC approach which means

it works by reacting to changes in the database's log files. Debezium uses Kafka Connect which makes it highly scalable and resilient to failures and has CDC connectors for multiple databases including Postgres, Mysql and MongoDB. We're using it in a few projects and it has worked very well for us.

We've been intrigued by **GLITCH**, which is a collaborative online development environment that lets you easily copy and adapt (or "remix") existing web apps or create your own. Rooted in the "tinkerer" ethos, it's ideal for people learning to code but it has the capability to support more complex applications. The main focus is on JavaScript and [Node.js](#), but it also has limited support for other languages. With integrated live editing, hosting, sharing and automatic source versioning, Glitch offers a refreshing and distinctive take on collaborative programming.

**GOOGLE CLOUD DATAFLOW** is useful in traditional ETL scenarios for reading data from a source, transforming it and then storing it to a sink, with configurations and scaling being managed by dataflow. Dataflow supports Java, Python and Scala and provides wrappers for connections to various types of data sources. However, the current version won't let you add additional libraries, which may make it unsuitable for certain data manipulations. You also can't change the dataflow DAG dynamically. Hence, if your ETL has conditional execution flows based on parameters, you may not be able to use dataflow without workarounds.

**GVISOR** is a user-space kernel for containers. It limits the host kernel surface accessible to the application without taking away access to all the features it expects. Unlike existing sandbox technologies, such as virtualized hardware ([KVM](#) and [Xen](#)) or rule-based execution ([seccomp](#), [SELinux](#) and [AppArmor](#)), gVisor takes a distinct approach to container sandboxing by intercepting application system calls and acting as the guest kernel without the need for translation through virtualized hardware. gVisor includes an [Open Container Initiative \(OCI\)](#) runtime called [runsc](#) that integrates with [Docker](#) and provides experimental support for [Kubernetes](#). gVisor is a relatively new project and we recommend assessing it for your container security landscape.



In most cases, blockchain is not the right place to store a blob file (e.g., image or audio). When developing DApp, one option is to put blob files in some off-chain centralized data storage, which usually signals lack of trust. Another option is to store them on InterPlanetary File System (**IPFS**), which is a content-addressed, versioned, peer-to-peer file system. It's designed to distribute high volumes of data with high efficiency and removed from any centralized authority. Files are stored on peers that don't need to trust each other. IPFS keeps every version of a file so you never lose important files. We see IPFS as a good complement to blockchain technology. Beyond its blockchain application, IPFS has an ambitious goal to decentralize the Internet infrastructure.

When building and operating a microservices ecosystem, one of the early questions to answer is how to implement cross-cutting concerns such as service discovery, service-to-service and origin-to-service security, observability (including telemetry and distributed tracing), rolling releases and resiliency. Over the last couple of years, our default answer to this question has been using a service mesh technique. A service mesh offers the implementation of these cross-cutting capabilities as an infrastructure layer that is configured as code. The policy configurations can be consistently applied to the whole ecosystem of microservices; enforced on both in and out of mesh traffic (via the mesh proxy as a gateway) as well as on the traffic at each service (via the same mesh proxy as a sidecar container). While we're keeping a close eye on the progress of different open source service mesh projects such as Linkerd, we've successfully used **ISTIO** in production with a surprisingly easy-to-configure operating model.

As application developers, we love to focus on solving core business problems and let the underlying platform handle the boring but difficult tasks of deploying, scaling and managing applications. Although serverless architecture is a step in that direction, most of the popular offerings are tied to a proprietary implementation, which means vendor lock-in. **KNATIVE** tries to address this by being an open source serverless platform that integrates well with the popular Kubernetes ecosystem. With Knative you can model computations on request in a supported framework of your choice (including Ruby on Rails, Django and Spring among others); subscribe, deliver and manage

events; integrate with familiar CI and CD tools; and build containers from source. By providing a set of middleware components for building source-centric and container-based applications that can be elastically scaled, Knative is an attractive platform that deserves to be assessed for your serverless needs.

*While tightly focusing on cloud-native architectures, Pulumi is an infrastructure automation tool that distinguishes itself by allowing configurations to be written in TypeScript/JavaScript, Python and Go.*

(Pulumi)

We're quite interested in **PULUMI**, a promising entrant in cloud infrastructure automation. Pulumi distinguishes itself by allowing configurations to be written in TypeScript/JavaScript, Python and Go—no YAML required. Pulumi is tightly focused on cloud-native architectures—including containers, serverless functions and data services—and provides good support for Kubernetes.

Ethereum is the leading developer ecosystem in blockchain tech. We've seen emerging solutions that aim to spread this technology into enterprise environments that usually require network permissioning and transaction privacy as well as higher throughput and lower latency. **QUORUM** is one of these solutions. Originally developed by J.P. Morgan, Quorum positions itself as "an enterprise-focused version of Ethereum." Unlike the Hyperledger Burrow node, which creates a new Ethereum virtual machine (EVM), Quorum forks code from Ethereum's official client so that it can evolve alongside Ethereum. Although it keeps most features of the Ethereum ledger, Quorum changes the consensus protocol from PoW to more efficient ones and adds private transaction support. With Quorum, developers can use their Ethereum knowledge of using, for example, Solidity and Truffle contracts to build enterprise blockchain applications. However, based on our experience, Quorum is not yet enterprise ready; for example, it lacks access control for private contracts, doesn't work well with load balancers and only has partial database support, all of which will lead to significant deployment and design burden. We recommend that you're cautious in implementing Quorum while keeping an eye on its development.

**RESIN.IO** is an Internet of Things (IoT) platform that does one thing and does it well: it deploys containers onto devices. Developers use a software as a service (SaaS) portal to manage devices and assign applications, defined by Dockerfiles, to them. The platform can build containers for various hardware types and deploys the images over the air. For the containers, Resin.io uses [balena](#), an engine based on the Moby framework created by [Docker](#). The platform is still under development, has some rough edges and lacks some features (e.g., working with private registries), but the current feature set, including the option to ssh into a container on a device from the web portal, points toward a promising future.

**ROOK** is an open source cloud native storage orchestrator for [Kubernetes](#). Rook integrates with [Ceph](#) and brings File, Block and Object storage systems into the Kubernetes cluster, running them seamlessly alongside other applications and services that are consuming the storage. By using Kubernetes operators, Rook orchestrates Ceph at the control plane and stays clear of the data path between applications and Ceph. Storage is one of the important components of cloud-native computing and we believe that Rook, though still an incubating-level project at [CNCF](#), takes us a step closer to self-sufficiency and portability across public cloud and on-premise deployments.

Making key elements of Google's groundbreaking, high-scale platform available as open source offerings appears to have become a trend. In the same way that HBASE drew on BigTable and [Kubernetes](#) drew on Borg, **SPIFFE** is now drawing upon Google's LOAS to bring to life a critical cloud-native concept called workload identity. The SPIFFE standards are backed by

the OSS SPIFFE Runtime Environment (SPIRE), which automatically delivers cryptographically provable identities to software workloads. Although SPIRE isn't quite ready for production use, we see tremendous value in a platform-agnostic way to make strong identity assertions between workloads in modern, distributed IT infrastructures. SPIRE supports many use cases, including identity translation, OAuth client authentication, mTLS "encryption everywhere," and workload observability. [Istio](#) uses SPIFFE by default.

**DATA-HUNGRY PACKAGES** are solutions that require absorption of data into themselves in order to function. In some cases they may even require that they become the "master" for that data. Once the data is owned by the package, that software becomes the only way to update, change or access the data. The data-hungry package might solve a particular business problem such as ERP. However, inventory or finance "data demands" placed upon an organization will often require complex integration and changes to systems that lie well outside of the original scope.

**LOW-CODE PLATFORMS** use graphical user interfaces and configuration in order to create applications. Unfortunately, low-code environments are promoted with the idea that this means you no longer need skilled development teams. Such suggestions ignore the fact that writing code is just a small part of what needs to happen to create high-quality software—practices such as source control, testing and careful design of solutions are just as important. Although these platforms have their uses, we suggest approaching them with caution, especially when they come with extravagant claims for lower cost and higher productivity.

# TOOLS

## ADOPT

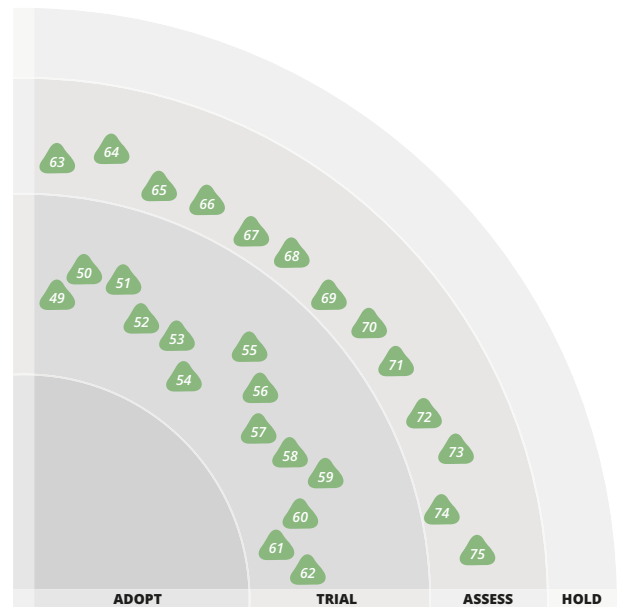
### TRIAL

- 49. [acs-engine](#) *NEW*
- 50. [Archery](#) *NEW*
- 51. [ArchUnit](#)
- 52. [Cypress](#)
- 53. [git-secrets](#) *NEW*
- 54. [Headless Firefox](#)
- 55. [LocalStack](#) *NEW*
- 56. [Mermaid](#) *NEW*
- 57. [Prettier](#) *NEW*
- 58. [Rider](#) *NEW*
- 59. [Snyk](#) *NEW*
- 60. [UI dev environments](#) *NEW*
- 61. [Visual Studio Code](#)
- 62. [VS Live Share](#) *NEW*

### ASSESS

- 63. [Bitrise](#) *NEW*
- 64. [Codefresh](#) *NEW*
- 65. [Grafeas](#) *NEW*
- 66. [Heptio Ark](#) *NEW*
- 67. [Jaeger](#) *NEW*
- 68. [kube-bench](#) *NEW*
- 69. [Ocelot](#) *NEW*
- 70. [Optimal Workshop](#) *NEW*
- 71. [Stanford CoreNLP](#) *NEW*
- 72. [Terragrunt](#) *NEW*
- 73. [TestCafe](#) *NEW*
- 74. [Traefik](#) *NEW*
- 75. [Wallaby.js](#) *NEW*

## HOLD




Azure Container Service Engine (**ACS-ENGINE**) is an Azure Resource Manager (ARM) template generator. The required configurations of the cluster are defined in a JSON file; [acs-engine](#) reads these [cluster definitions](#) and generates a number of files that can be consumed by ARM. The tool also provides flexibility to choose different orchestrators—including [Kubernetes](#), [DC/OS](#), [OpenShift](#), [Swarm mode](#) and [Swarm](#)—and to configure features and agents of the cluster. We've been using [acs-engine](#) in a number of projects and would recommend it for managing clusters in Azure Container Service.

We're seeing significant advances in security tooling integration with modern software delivery processes. **ARCHERY** is an open source tool with an active community that's doing a good job of pulling together a collection of other tools, including [Zap](#). Designed primarily for web applications, Archery makes it easy to integrate security tooling into your build and deployment systems. Its dashboards also let you track vulnerabilities as well as application and network scans.

**ARCHUNIT** is a Java testing library for checking architecture characteristics such as package and class dependencies, annotation verification and even layer consistency. We like that it runs as unit tests within your existing test setup, even though it supports only Java-based architectures. The ArchUnit test suite can be incorporated into a CI environment or a deployment pipeline, making it easier to implement [fitness functions](#) in an [evolutionary architecture](#) way.

Running end-to-end tests can present challenges, such as the long duration of the running process, the flakiness of some tests and the challenges of fixing failures in CI when running tests in headless mode. Our teams have had very good experiences with **CYPRESS** by solving common issues such as lack of performance and long wait time for responses and resources to load. Cypress is a useful tool that helps developers build end-to-end tests and records all test steps as a video in an MP4 file to make it easier to identify errors.



*A simple tool that prevents you from committing passwords and other sensitive information to a git repository, it also scans all historical revisions before making a repository public.*

(git-secrets)

Security continues to be paramount, and inadvertently checking credentials and other secrets into source control is a major attack vector. **GIT-SECRETS** is a simple tool that prevents you from committing passwords and other sensitive information to a git repository. It can also scan all historical revisions before making a repository public, if you want to ensure you've never accidentally checked in a credential. git-secrets comes with built-in support for common AWS keys and credentials and can be set up quickly for other providers too.

**HEADLESS FIREFOX** has the same maturity as that of Headless Chrome for front-end test. Similar to Headless Chrome, with Firefox in headless mode we now get to enjoy browser tests without the visible UI components, executing the UI tests suite much faster.



*Testing cloud services locally is a challenge. LocalStack solves this problem for AWS by providing local test double implementations of a wide range of AWS services, including S3, Kinesis, DynamoDB and Lambda.*

(LocalStack)

One of the challenges of using cloud services is being able to develop and test locally using those services. **LOCALSTACK** solves this problem for AWS by providing local test double implementations of a wide range of AWS services, including S3, Kinesis, DynamoDB and Lambda. It builds on top of existing best-of-breed tools such as Kinesalite, Dynalite and Moto and adds isolated processes and error injection functionality. LocalStack is very easy to use and ships with a simple JUnit runner and a JUnit 5 extension. We're using it in a few of our projects and have been impressed with it.

**MERMAID** lets you generate diagrams from a markdown-like markup language. Born out of need

to simplify documentation, Mermaid has grown into a larger ecosystem with plugins for Confluence, Visual Studio Code and Jekyll to name a few. To see how it works, you can use the Live Editor on GitHub. Mermaid also has a convenient command line interface that lets you generate SVG, PNG and PDF files as output from definition files. We've been using Mermaid in many projects and we like the simplicity of describing graphs and flowcharts with markdown and checking in the definition files with the code repository.

*An opinionated, automated code formatter for JavaScript, Prettier increases consistency and readability and reduces developer effort both on formatting and engaging in wasteful team debates about code style.*

(Prettier)

**PRETTIER** is an opinionated, automated code formatter for JavaScript (with growing support for other languages). By enforcing its own opinionated formatting style it increases consistency and readability and reduces developer effort both on formatting and engaging in wasteful team debates about code style. Even though you may disagree with the stylistic choices enforced by Prettier, we find that the benefits to the team generally outweigh small style issues. Prettier can be used with a precommit hook or an IDE plugin. As with any formatter, a one-time reformatting of your codebase can confuse your version control history, but we feel that's a minor drawback. We particularly like the way Prettier flips the linter-based approach and, borrowing from gofmt, instead of validating your code, it ensures that your code will always be valid.

We've covered Visual Studio Code in the Radar since 2015, but it isn't the only cross-platform .NET Core IDE kid on the block anymore. Recently, **RIDER**, which is part of the IDEA platform developed by JetBrains, has gained adoption, especially by those used to the speed and dexterity provided by ReSharper, which drives the refactoring in Rider. Rider, however, does more than ReSharper to bring the full IDEA platform to .NET and increase developer productivity. Regardless of your preferred platform, it's worth exploring Rider as it currently has the productivity edge on Visual Studio Code. It's also great to see the ecosystem alive and well, as competition ensures these tools continue to improve.

**SNYK** helps you find, fix and monitor known vulnerabilities in npm, Ruby, Python, Scala, Golang, .NET, PHP, Java and Docker dependency trees. When added to your build pipeline, Snyk continuously monitors and tests the library dependency tree against a hosted vulnerability database and suggests the minimal direct dependency version upgrade needed for remediation.

As more and more teams embrace DesignOps, practices and tooling in this space mature, too. Many of our teams now work with what could be called **UI DEV ENVIRONMENTS**, which provide a comprehensive environment for quickly iterating on UI components, focusing on collaboration between user experience designers and developers. We now have a few options in this space: Storybook, react-styleguidist, Compositor and MDX. You can use these tools standalone in component library or design system development as well as embedded in a web application project. Rather than spinning up the app, plus a BFF, plus services simply to add a feature to a component, you can start up the Storybook dev server instead.

**VISUAL STUDIO CODE** is Microsoft's free IDE editor, available across platforms. We've had good experience using this for front-end development using React and TypeScript, and back-end languages such as GoLang, without having to switch between different editors. The tooling, language support and extensions for Visual Studio Code continue to soar and get better. We'd particularly like to call out VS Live Share for real-time collaboration and remote pairing. While complex projects in statically typed languages, such as Java, .NET or C++, will likely find better support from the more mature IDEs from Microsoft or JetBrains, we find that Visual Studio Code is increasingly becoming a tool of choice among infrastructure and front-end development teams.

*The real-time collaboration with VS Live Share makes remote pairing easier. Particularly, we like that it allows developers to collaborate with their own preconfigured editor.*

(VS Live Share)

**VS LIVE SHARE** is a suite of extensions for Visual Studio Code and Visual Studio. The real-time collaboration for editing and debugging of code, voice calls, sharing a terminal and exposing local ports have reduced some of the obstacles we'd otherwise encounter when pairing remotely. In particular, we like that Live Share allows developers to collaborate with each other, while continuing to use their preconfigured editor, which includes themes, key maps and extensions.

*Building, testing and deploying mobile applications entails a number of complex steps, especially for a pipeline from source code repositories to app stores. This easy-to-set-up, domain-specific tool can reduce the complexity and maintenance overhead.*

(Bitrise)

Building, testing and deploying mobile applications entails a number of complex steps, especially when we consider a pipeline from source code repositories to app stores. All these steps can be automated with scripts and build pipelines in generic CI/CD tools. However, for teams that focus on mobile development and have little or no requirement to integrate with build pipelines for back-end systems, a domain-specific tool can reduce the complexity and maintenance overhead. **BITRISE** is easy to set up and provides a comprehensive set of prebuilt steps for most mobile development needs.

**CODEFRESH** is a hosted CI server similar to CircleCI or Buildkite. It's container-centric, making Dockerfiles and container-hosting clusters first-class entities. We like that the tool encourages a pipelined delivery approach and supports branching and merging. Early reports from our teams are positive, but we've yet to see how it works for larger projects and complex pipelines.

*We're continually on the lookout for tools and techniques that allow delivery teams to work independently from the rest of a larger organization while staying within its security and risk guardrails. Grafeas is such a tool.*

(Grafeas)

We're continually on the lookout for tools and techniques that allow delivery teams to work independently from the rest of a larger organization while staying within its security and risk guardrails. **GRAFEAS** is such a tool. It lets organizations publish authoritative metadata about software artifacts— Docker images, libraries, packages—that is then accessible from build scripts or other automated compliance controls. The access control mechanisms allow for a separation of responsibility between the teams that publish approvals or vulnerabilities and the teams that build and deploy software. Although several organizations, including Google and JFrog, use Grafeas in their workflows, note that the tool is still in alpha.

*As a tool for managing disaster recovery for Kubernetes clusters and persistent volumes, Ark is easy to use and configure, and lets you backup and restore your clusters through a series of checkpoints.*

(Heptio Ark)

**HEPTIO ARK** is a tool for managing disaster recovery for Kubernetes clusters and persistent volumes. Ark is easy to use and configure and lets you back up and restore your clusters through a series of checkpoints. With Ark you can significantly reduce recovery time in case of an infrastructure failure, easily migrate Kubernetes resources from one cluster to another and replicate the production environment for testing and troubleshooting. Ark supports key backup storage providers (including AWS, Azure and Google Cloud) and, as of version 0.6.0, a plugin system that adds compatibility for additional backup and volume storage platforms. Managed Kubernetes environments, such as GKE, provide these services out of the box. However, if you're operating Kubernetes either on premise or in the cloud, take a closer look at Heptio Ark for disaster recovery.

**JAEGER** is an open source distributed tracing system. Similar to Zipkin, it's been inspired by the Google Dapper paper and complies with OpenTracing. Jaeger is a younger open source project than Zipkin, but

it's gained popularity quickly due to a larger number of supported languages for the client libraries and easy installation on Kubernetes. We've used Jaeger successfully with Istio, integrating application traces with Envoy on Kubernetes and like its UI. With Jaeger joining CNCF, we anticipate a larger community engagement effort and deeper integration with other CNCF projects.

**KUBE-BENCH** is an example of an infrastructure configuration scanner that automates checking your Kubernetes configuration against the CIS benchmark for K8s. It covers user authentication, permissions and secure data among other areas. Our teams have found kube-bench valuable in the identification of vulnerable configurations.

**OCELOT** is a .NET API gateway. After three years of development, Ocelot has built a relatively complete feature set and an active community. Although there is no dearth of excellent API gateways (e.g., Kong), the .NET community appears to prefer Ocelot when building microservices. Part of the reason is that Ocelot integrates well with the .NET ecosystem (e.g., with IdentityServer). Another reason may be that the .NET community has extended Ocelot to support communication protocols such as gRPC, Orleans and WebSocket.

UX research demands data collection and analysis to make better decisions about the products we need to build. **OPTIMAL WORKSHOP** is a suite of tools that helps to do this digitally. Features such as first-click or card sorting help to both validate prototypes and improve website navigation and information display. For distributed teams, in particular, benefit from Optimal Workshop as it lets them conduct remote research.

*Extracting meaningful business information from text data is a key technique for unstructured data processing. Stanford CoreNLP, a Java-based set of natural language processing tools, helps us to use the latest research in the field of NLP to solve various business problems.*

(Stanford CoreNLP)



We have more and more projects that require unstructured data processing. To extract meaningful business information from text data is a key technique. [STANFORD CORENLP](#) is a Java-based set of natural language processing tools. It supports named-entity recognition, relationship extraction, sentiment analysis and text classification as well as multiple languages, including English, Chinese and Arabic. We also find tools usable to label corpus and training models for our scenario. With Stanford CoreNLP, we were able to use the latest research in the field of NLP to solve various business problems.

We widely use [Terraform](#) as code to configure a cloud infrastructure. [TERRAGRUNT](#) is a thin wrapper for Terraform that implements the practices advocated by the [Terraform: Up and Running](#) book. We've found Terragrunt helpful as it encourages versioned modules and reusability for different environments with some handy features, including recursive code execution in subdirectories. We'd like to see the tool evolve to support CD practices natively, where all code can be packaged, versioned and reused across different environments on CD pipelines. Our team achieves this today with workarounds.

Our teams are reporting good success with [TESTCAFE](#), a JavaScript-based browser test automation tool. TestCafe allows you to write tests in JavaScript or

[TypeScript](#) and runs tests in any browser that supports JavaScript. TestCafe has several useful features including out-of-the-box parallel execution and HTTP request mocking. TestCafe uses an asynchronous execution model with no explicit wait times, which results in much more stable test suites.

[TRAEFIK](#) is an open-source reverse proxy and load balancer. If you're looking for an edge proxy that provides simple routing without all the features of [NGINX](#) and [HAProxy](#), Traefik is a good choice. The router provides a reload-less reconfiguration, metrics, monitoring and circuit breakers that are essential when running microservices. It also integrates nicely with [Let's Encrypt](#) to provide SSL termination. When compared to Traefik, tools such as NGINX and HAProxy may require additional tooling to templatize configuration in response to scaling, adding or removing microservices and may, at times, require a restart which can be annoying in production environments.

We all obsess about fast feedback during test-driven development and we're always looking for new ways to make this even faster. [WALLABY.JS](#) is a commercial extension for popular editors that provides continuous execution of JavaScript unit tests, highlighting the results in line next to your code. The tool identifies and runs the minimum set of tests affected by each code change and lets you run tests continuously as you type.



# LANGUAGES & FRAMEWORKS

## ADOPT

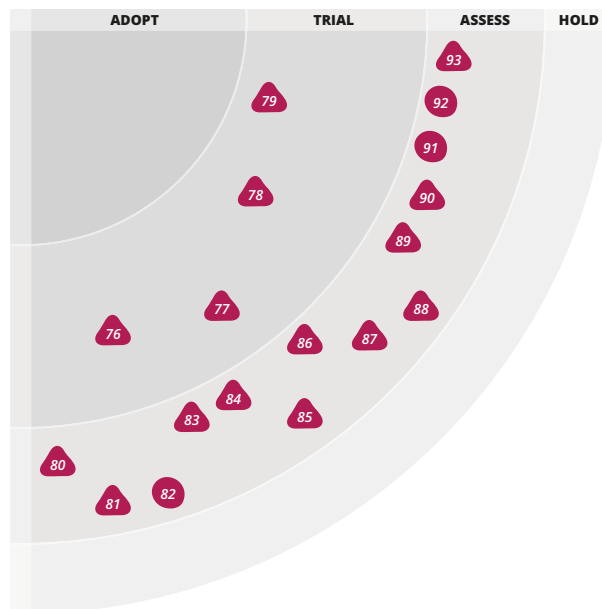
### TRIAL

- 76. Jepsen
- 77. MMKV **NEW**
- 78. MockK **NEW**
- 79. TypeScript

### ASSESS

- 80. Apache Beam **NEW**
- 81. Camunda **NEW**
- 82. Flutter
- 83. Ktor **NEW**
- 84. Nameko **NEW**
- 85. Polly.js **NEW**
- 86. PredictionIO **NEW**
- 87. Puppeteer **NEW**
- 88. Q# **NEW**
- 89. SAFE stack **NEW**
- 90. Spek **NEW**
- 91. troposphere
- 92. WebAssembly
- 93. WebFlux **NEW**

## HOLD



With the increased adoption of a [microservices](#) architecture, we're building more distributed applications than before. Although there are many benefits of a decoupled architecture, the complexity and the effort involved in proving the correctness of the overall system has dramatically increased.

**JEPSEN** provides much needed tooling to verify correctness in coordination of task schedulers, test eventual consistency, linearizability and serializability characteristics of distributed databases. We've used Jepsen in a few projects and we like the fact that we can test drive configurations, inject and correct faults, and verify the state of the system after recovery.

An open source framework developed by WeChat, **MMKV** provides fast key-value storage for mobile apps. It uses iOS memory-mapping features to avoid the need to explicitly save changes and is extremely fast and

performant. In the event of an unexpected crash, MMKV allows the app to restore the data quickly.


*As a native library, MockK helps our teams to write clean and concise tests for Kotlin applications instead of using inconvoluted wrappers of Mockito or PowerMock.*

(MockK)

**MOCKK** is a library for mocking written in Kotlin. Its main philosophy is to provide first-class support for Kotlin language features such as [Coroutines](#) or lambda blocks. As a native library, it helps our teams to write clean and concise code on testing Kotlin applications instead of using inconvoluted wrappers of Mockito or PowerMock.

**TYPESCRIPT** is a carefully considered language and its consistently improving tools and IDE support continues to impress us. With a good repository of TypeScript-type definitions, we benefit from all the rich JavaScript libraries while gaining type safety. This is particularly important as our browser-based code base continues to grow. The type safety in TypeScript lets you use IDEs and other tools to provide deeper context into your code and make changes and refactor code with safety. TypeScript, being a superset of JavaScript, and documentation and the community has helped ease the learning curve.

**APACHE BEAM** is an open source unified programming model for defining and executing both batch and streaming data-parallel processing pipelines. Beam provides a portable API layer for describing these pipelines independent of execution engines (or runners) such as [Apache Spark](#), [Apache Flink](#) or [Google Cloud Dataflow](#). Different runners have different capabilities and providing a portable API is a difficult task. Beam tries to strike a delicate balance by actively pulling innovations from these runners into the Beam model and also working with the community to influence the roadmap of these runners. Beam has a rich set of [built-in I/O transformations](#) that cover most of the data pipeline needs and it also provides a mechanism to implement [custom transformations](#) for specific use cases. The portable API and extensible IO transformations make a compelling case for assessing Apache Beam for data pipeline needs.



*Although we tend to be skeptical of business process model and notation (BPMN) tools, Camunda is easy to test, version and refactor. Integrating with Spring and Spring Boot makes it a solid choice.*

(Camunda)

We tend to be quite skeptical of business process model and notation (BPMN) tools in general as they're often associated with low-code environments and their downsides. Although the OSS BPMN framework **CAMUNDA** provides some of this whizziness, it also offers workflow and decision engines that can be directly integrated as a library in your Java code. This makes it easy to test, version and refactor workflows.

Camunda also integrates with Spring and Spring Boot, among other frameworks, making it a solid choice.

**FLUTTER** is a cross-platform framework that enables you to write native mobile apps in [Dart](#). It benefits from Dart and can be compiled into native code and communicates with the target platform without bridge and context switching—something that can cause performance bottlenecks in frameworks such as [React Native](#) or [Weex](#). Flutter's hot-reload feature is impressive and provides superfast visual feedback when editing code. Currently, Flutter is still in beta, but we'll continue keeping an eye on it to see how its ecosystem matures.

[Kotlin](#) is no longer just a great fit for mobile app development. New tools and frameworks have emerged that demonstrate the value of the language for web application development as well. **KTOR** is one such framework. In contrast to other web frameworks that support Kotlin, Ktor is written in Kotlin, using language features such as [coroutines](#) which allows for an asynchronous non-blocking implementation. The flexibility to incorporate different tools for logging, DI or a templates engine—in addition to its lightweight architecture—makes Ktor an interesting option for our teams for creating RESTful services.

One insight we gained after talking with our teams is that [Python](#) is making a comeback across many technology domains. In fact, it's well on its way to become the [most-used programming language](#). In part, this is driven by its adoption by data scientists and in machine learning, but we also see teams adopting it to build microservices. **NAMEKO** is a super-lightweight microservices framework and an alternative to [Flask](#) for writing services. Unlike Flask, Nameko only has a limited set of features that includes WebSocket, HTTP and AMQP support. We also like its focus on testability. If you don't need features such as templating that Flask provides, then Nameko is worth a look.

**POLLY.JS** is a simple tool that helps teams test JavaScript websites and applications. Our teams particularly like that it enables them to intercept and stub HTTP interactions which allows for easier and faster testing of JavaScript code without having to spin up dependent services or components.

**PREDICTIONIO** is an open source machine-learning server. Developers and data scientists can use it to build intelligent applications for prediction. Like all intelligent applications, PredictionIO has three parts: data collection and storage, model training, and model deployment and expose service. Developers could focus on implementing data-processing logic, model algorithm and prediction logic based on the corresponding interfaces and liberate themselves from data storage and model training deployment. In our experience, PredictionIO can support both small and large volumes of data with low concurrency. We mostly use PredictionIO to build predictive services for small and medium-sized enterprises or as a proof of concept when building more complex, customized prediction engines.

In the previous Radar we mentioned [Headless Chrome for front-end test](#). With the adoption of [Chrome DevTools Protocol \(CDP\)](#) by other browsers a new set of libraries is emerging for browser automation and testing. CDP allows for fine-grained control over the browser even in headless mode. New high-level libraries are being created using CDP for testing and automation. **PUPPETEER** is one of these new libraries. It can drive headless Chrome through a single-page application, obtain time-trace for performance diagnostics and more. Our teams found it faster and also more flexible than alternatives based on WebDriver.

*While still waiting for the hardware to arrive, we can experiment with quantum computing using languages and simulators. Samples from Q# can give you an idea of programming's potential future.*

(Q#)

Quantum computing currently exists in a twilight zone of being available for testing without having arrived yet. While we're still waiting for the hardware to arrive, we can experiment with and learn from languages and simulators. Although IBM and others have been making good progress, we've paid particular attention to Microsoft's efforts based around the **Q#** language and its simulator (32 qubits locally and 40 on Azure). If you want to start wrapping your head around the potential future of programming, check out their set of [samples on GitHub](#).

The **SAFE STACK**—short for [Suave](#), [Azure](#), [Fable](#) and [Elmish](#)—brings a number of technologies into

a coherent stack for web development. It's built around the F# programming language, both on the server side and in the browser, and therefore has a focus on functional, type-safe programming with an asynchronous approach. It offers productivity features such as hot reloading and lets you substitute parts of the stack, for example, the server-side web framework or the cloud provider.

The adoption of a new language typically spawns the emergence of new tools that support mature engineering practices such as test automation. [Kotlin](#) is no exception. **SPEK** is a testing framework—inspired by well-known tools such as [Cucumber](#), [RSpec](#) and [Jasmine](#)—that writes tests in Gherkin and Specification, allowing teams to bring mature practices such as [behaviour-driven development](#) into the Kotlin space.

We're trying out **TROPOSPHERE** as a way of defining the [infrastructure as code](#) on AWS for our projects that use [AWS CloudFormation](#) instead of Terraform. troposphere is a Python library that allows us to write Python code to generate CloudFormation JSON descriptions. What we like about troposphere is that it facilitates catching JSON errors early, applying type checking, and unit testing and DRY composition of AWS resources.

**WEBASSEMBLY** is a big step forward in the capabilities of the browser as a code execution environment. Supported by all major browsers and backward compatible, it's a binary compilation format designed to run in the browser at near native speeds. It opens up the range of languages you can use to write front-end functionality, with early focus on C, C++ and Rust, and it's also an LLVM compilation target. When run in the sandbox, it can interact with JavaScript and shares the same permissions and security model. When used with [Firefox's new streaming compiler](#), it also results in faster page initialization. Although it's still early days, this W3C standard is definitely one to start exploring.

*After working with a functional-reactive style on a number of applications, our teams have come away impressed and report that the approach improves code readability and system throughput.*

(WebFlux)

Spring Framework 5, released over a year ago, embraces reactive streams, a standard for asynchronous stream processing with non-blocking backpressure. The **WEBFLUX** module introduces a reactive alternative to the traditional Spring MVC module for writing web applications in the Spring ecosystem. After working with it on a

number of applications, our teams have come away impressed and report that the reactive (functional) approach improves code readability and system throughput. They do note, though, that adopting WebFlux requires a significant shift in thinking and recommend to factor this into the decision to choose WebFlux over Spring MVC.

Be the first to know when the  
Technology Radar launches, and  
keep up to date with exclusive  
webinars and content.

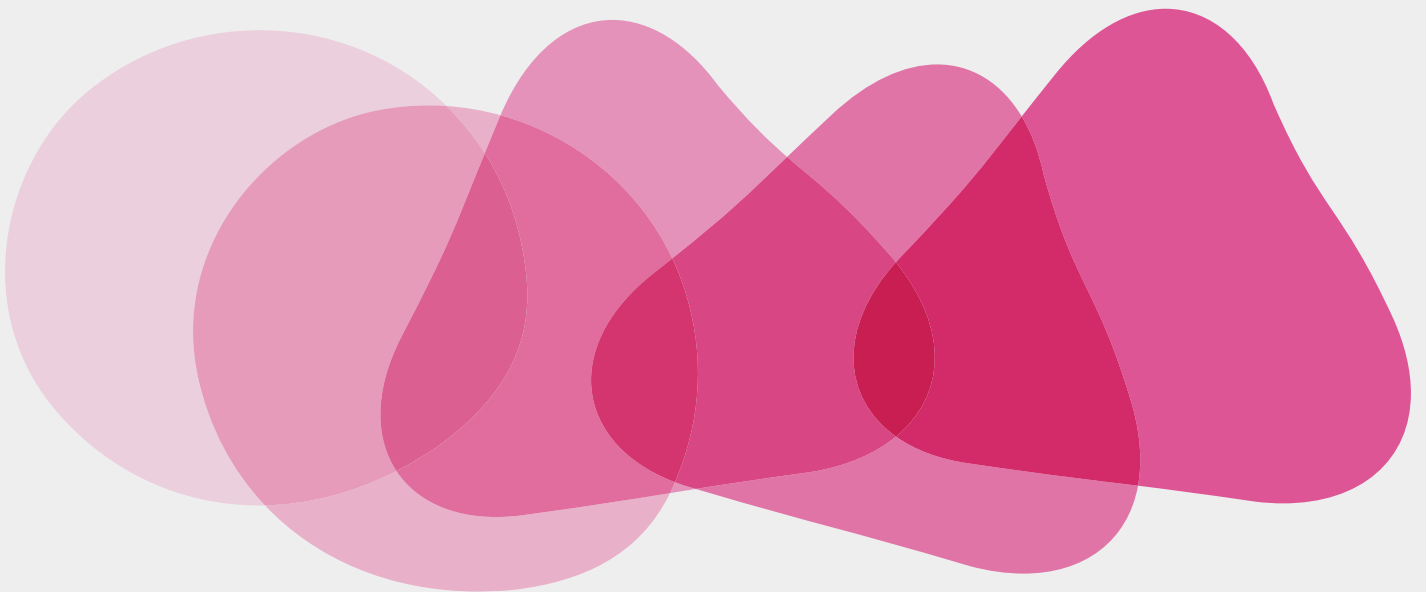
***SUBSCRIBE NOW***

*[thght.works/Sub-EN](https://thght.works/Sub-EN)*

# ThoughtWorks®

ThoughtWorks is a technology consultancy and community of passionate, purpose-led individuals. We help our clients put technology at the core of their business and together create the software that matters most to them. Dedicated to positive social change; our mission is to better humanity through software, and we partner with many organizations striving in the same direction.

Founded 25 years ago, ThoughtWorks has grown to a company of over 5,000 people, including a products division that makes pioneering tools for software teams. ThoughtWorks has 41 offices across 14 countries: Australia, Brazil, Canada, Chile, China, Ecuador, Germany, India, Italy, Singapore, Spain, Thailand, the United Kingdom and the United States.



[thoughtworks.com/radar](https://thoughtworks.com/radar)

**#TWTechRadar**