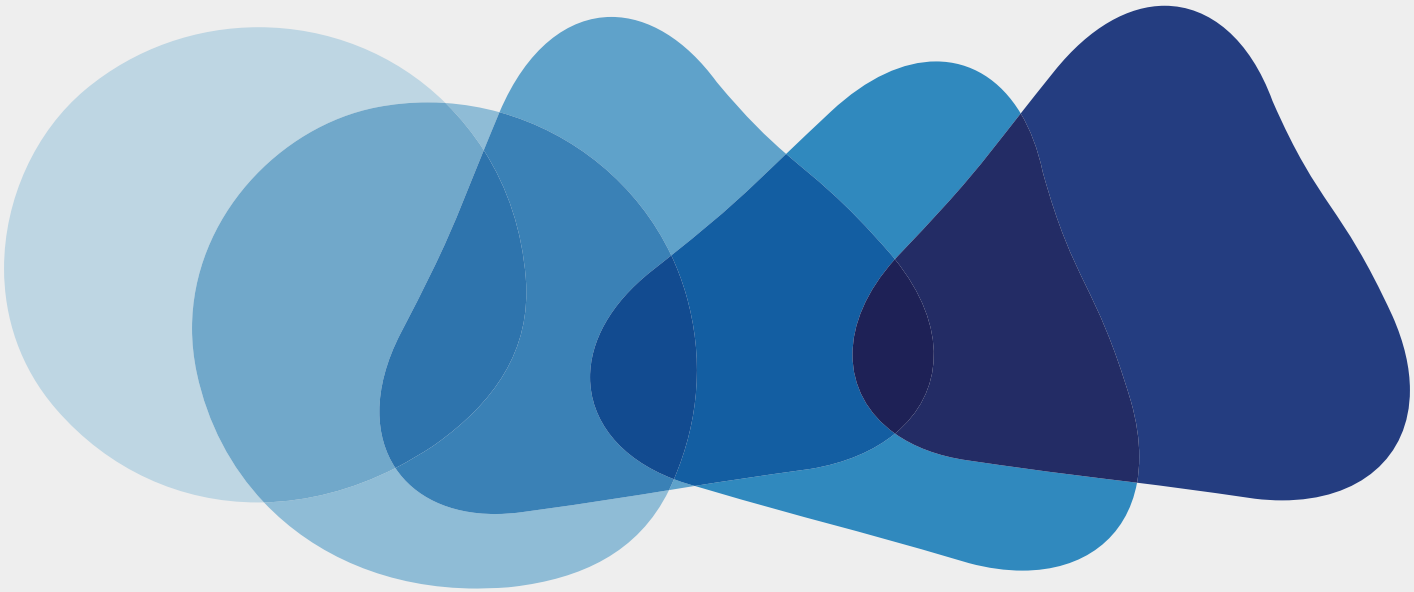
ThoughtWorks®

TECHNOLOGY RADAR *VOL.18*

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



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 | Rachel Laycock | Scott Shaw | Shangqi Liu | Zhamak Dehghani

This edition of the ThoughtWorks Technology Radar based on a meeting of the Technology Advisory Board, which met in Sydney in March 2018

WHAT'S NEW?

Highlighted themes in this edition:

BROWSER BULKS UP, SERVER SLIMS DOWN

The browser continues to expand its capabilities as a deployment target for application logic. As platforms take care of more cross-cutting concerns and nonfunctional requirements, we see a trend toward reduced complexity in back-end logic. The introduction of WebAssembly opens up new language options to create logic for web applications and pushes processing closer to the metal (and the GPU). Web Bluetooth enables browsers to handle functionality formerly reserved for native applications, and we increasingly see open standards such as CSS Grid Layout and CSS Modules supplanting custom libraries. The search for better user experiences encourages the trend toward pushing functionality into the browser, and many back-end services become thinner and less complex as a result. Display and interaction logic resides in the client, platforms or sidecars handle cross-functional concerns, and the back-end services can focus on core domain logic.

CREEPING CLOUD COMPLEXITY

While AWS continues to race ahead with a dizzying array of new services, we increasingly see Google Cloud Platform (GCP) and Microsoft Azure mature as viable alternatives. Abstraction layers such as Kubernetes and practices such as continuous delivery and infrastructure as code facilitate the transition between clouds by supporting easier evolutionary change. But cloud strategies necessarily become more complex with the advent of Polycloud (which allows organizations to pick and choose multiple providers based on differentiated capabilities) and growing regulatory and privacy concerns. For example, many EU countries now legally mandate data locality, making the jurisdiction of data storage and the underlying host policies a new dimension of differentiation for cloud evaluators. The range of options for compute environments is also increasing, with AWS Fargate offering containers as a service (CaaS) as an intriguing middle ground between functions as a service and managing longer-lived clusters. While cloud resources continue to mature within organizations, an inevitable creeping complexity always accompanies building real solutions with these new pieces.

TRUST TEAMS BUT VERIFY

Security remains of paramount concern for virtually all software development. But we see a shift in the traditional "lock everything down globally" approach to a more nuanced, localized approach. Many systems now manage trust within smaller domains and use modern mechanisms to create transitive trust between disparate systems. The philosophy has shifted from "never trust anything" outside of the domain and "never verify anything" inside the domain to "trust teams but verify" everywhere — that is to say, assume well-intentioned interactions with other parts of the system but verify trust at the local level. This enables teams to enjoy high degrees of control over their own infrastructure, equipment, and application stacks, leading to high visibility and, when necessary, high guardrails for access. Tools such as Scout2 and techniques such as BeyondCorp reflect this maturing perspective on trust. We welcome this shift toward localized autonomy, especially when tools and automation can ensure equal or better compliance.

`things` EVOLVE

The Internet of Things (IoT) ecosystem continues to evolve at a steady and strong pace and includes critical success factors such as security and maturing engineering practices. We see growth across the entire IoT ecosystem, from on-device operating systems to connectivity standards and most strongly in cloud-based device management and data processing. We see maturity in tools and frameworks that support good engineering practices such as continuous delivery, deployment, and a host of other necessities for eventual widespread use. In addition to the main cloud providers — including Google IoT Core, AWS IoT, and Microsoft Azure IoT Hub — companies such as Alibaba and Aliyun are also investing heavily in IoT PaaS solutions. Our EMQ and Mongoose OS blips provide a glimpse of the mainstream capabilities of today's IoT ecosystem and illustrate that **`things`** are evolving nicely indeed.

© ThoughtWorks, Inc. All Rights Reserved.

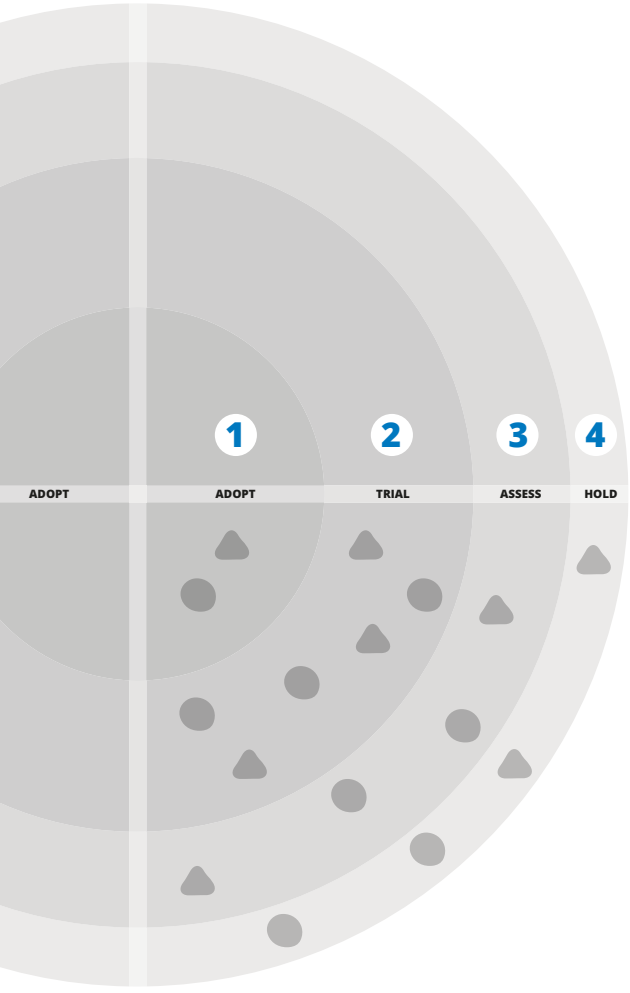
ABOUT THE RADAR

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 in 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 & 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

1 ADOPT

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

2 TRIAL

Worth pursuing. It is 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

● NO CHANGE

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

! 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.

© ThoughtWorks, Inc. All Rights Reserved.

THE RADAR

TECHNIQUES

ADOPT
1. Lightweight Architecture Decision Records

TRIAL
2. Applying product management to internal platforms
3. Architectural fitness function
4. Autonomous bubble pattern
5. Chaos Engineering
6. Domain-scoped events *NEW*
7. Hosted identity management as a service *NEW*
8. Micro frontends
9. Pipelines for infrastructure as code
10. Polycloud

ASSESS
11. BeyondCorp *NEW*
12. Embedded mobile mocks *NEW*
13. Ethereum for decentralized applications
14. Event streaming as the source of truth
15. GraphQL for server side resource aggregation *NEW*
16. Infrastructure configuration scanner *NEW*
17. Jupyter for automated testing *NEW*
18. Log level per request *NEW*
19. Security Chaos Engineering *NEW*
20. Service mesh
21. Sidecars for endpoint security
22. The three Rs of security

HOLD
23. Generic cloud usage *NEW*
24. Recreating ESB antipatterns with Kafka

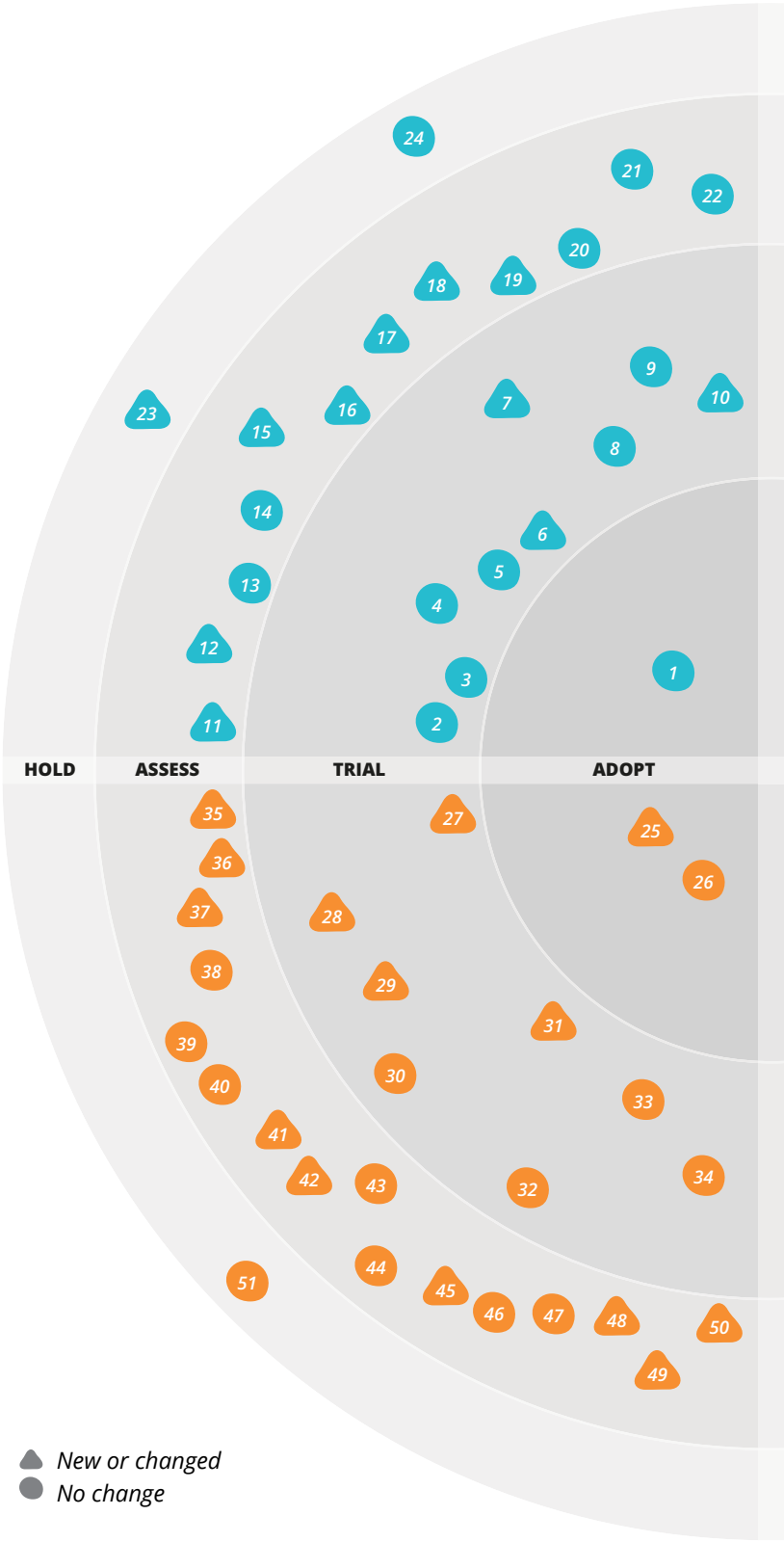
PLATFORMS

ADOPT
25. .NET Core
26. Kubernetes

TRIAL
27. Azure
28. Contentful *NEW*
29. EMQ *NEW*
30. Flood IO
31. GKE
32. Google Cloud Platform
33. Keycloak
34. WeChat

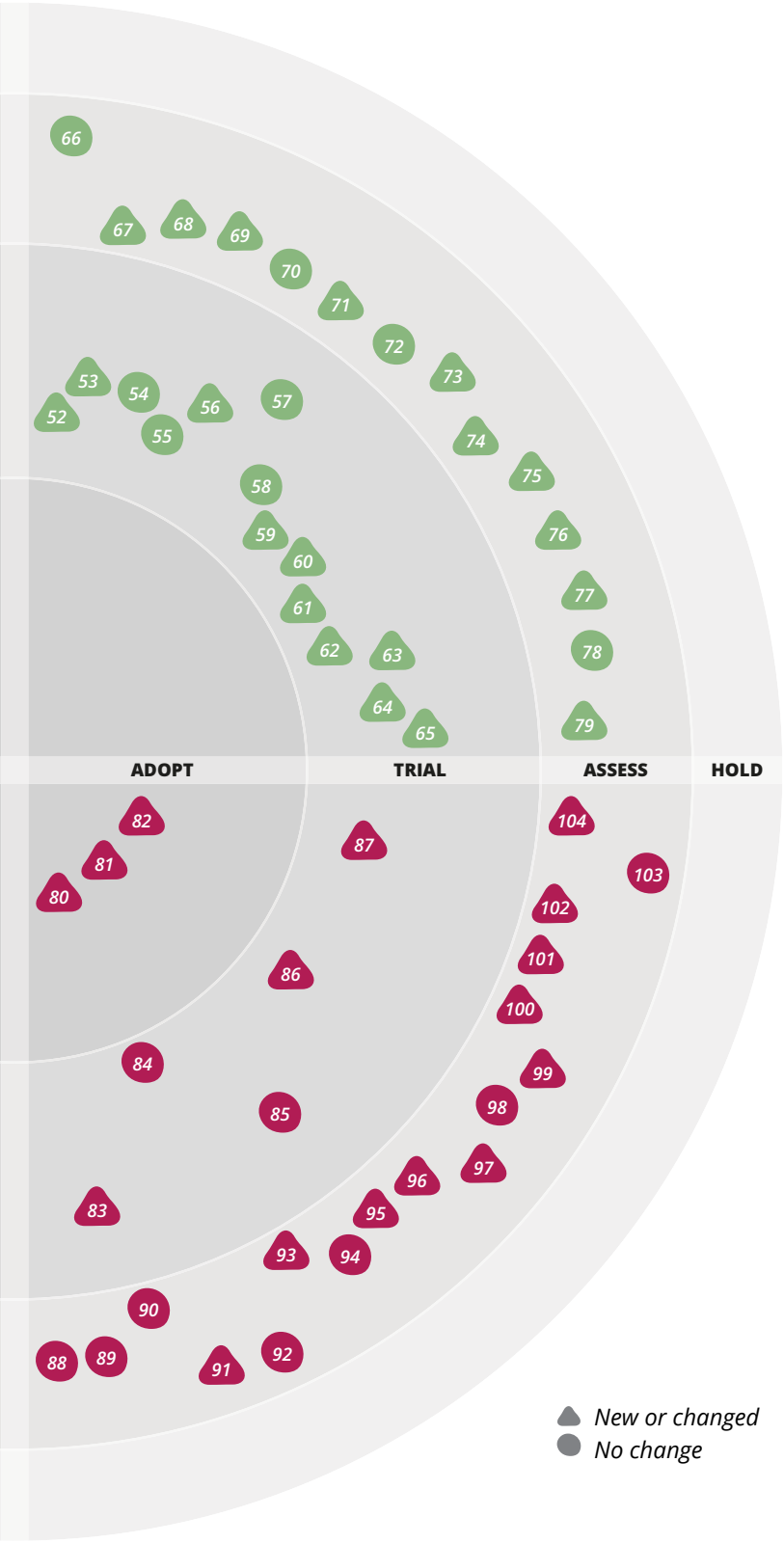
ASSESS
35. AWS Fargate *NEW*
36. Azure Service Fabric
37. Azure Stack *NEW*
38. Cloud Spanner
39. Corda
40. Cosmos DB
41. Godot *NEW*
42. Interledger *NEW*
43. Language Server Protocol
44. LoRaWAN
45. Mongoose OS *NEW*
46. Netlify
47. TensorFlow Serving
48. TICK Stack *NEW*
49. Web Bluetooth *NEW*
50. Windows Containers

HOLD
51. Overambitious API gateways



© ThoughtWorks, Inc. All Rights Reserved.

THE RADAR



TOOLS

ADOPT

TRIAL

52. Appium Test Distribution *NEW*
53. BackstopJS *NEW*
54. Buildkite
55. CircleCI
56. CVXPY *NEW*
57. gopass
58. Headless Chrome for front-end test
59. Helm *NEW*
60. Jupyter
61. Kong API Gateway
62. kops
63. Patroni *NEW*
64. WireMock *NEW*
65. Yarn

ASSESS

66. Apex
67. ArchUnit *NEW*
68. cfn_nag *NEW*
69. Conduit *NEW*
70. Cypress
71. Dependabot *NEW*
72. Flow
73. Headless Firefox *NEW*
74. nsp *NEW*
75. Parcel *NEW*
76. Scout2 *NEW*
77. Sentry *NEW*
78. Sonobuoy
79. Swashbuckle for .NET Core *NEW*

HOLD

LANGUAGES & FRAMEWORKS

ADOPT

80. AssertJ
81. Enzyme
82. Kotlin

TRIAL

83. Apollo *NEW*
84. CSS Grid Layout
85. CSS Modules
86. Hyperledger Composer *NEW*
87. OpenZeppelin *NEW*

ASSESS

88. Android Architecture Components
89. Atlas and BeeHive
90. Clara rules
91. Flutter *NEW*
92. Gobot
93. Hyperapp *NEW*
94. PyTorch
95. Rasa *NEW*
96. Reactor *NEW*
97. RIBs *NEW*
98. Solidity
99. SwiftNIO *NEW*
100. Tensorflow Eager Execution *NEW*
101. TensorFlow Lite *NEW*
102. troposphere *NEW*
103. Truffle
104. WebAssembly *NEW*

HOLD

© ThoughtWorks, Inc. All Rights Reserved.

TECHNIQUES

ADOPT
1. Lightweight Architecture Decision Records

TRIAL
2. Applying product management to internal platforms
3. Architectural fitness function
4. Autonomous bubble pattern
5. Chaos Engineering
6. Domain-scoped events *NEW*
7. Hosted identity management as a service *NEW*
8. Micro frontends
9. Pipelines for infrastructure as code
10. Polycloud

ASSESS
11. BeyondCorp *NEW*
12. Embedded mobile mocks *NEW*
13. Ethereum for decentralized applications
14. Event streaming as the source of truth
15. GraphQL for server side resource aggregation *NEW*
16. Infrastructure configuration scanner *NEW*
17. Jupyter for automated testing *NEW*
18. Log level per request *NEW*
19. Security Chaos Engineering *NEW*
20. Service mesh
21. Sidecars for endpoint security
22. The three Rs of security

HOLD
23. Generic cloud usage *NEW*
24. Recreating ESB antipatterns with Kafka

It's important to remember that encapsulation applies to events and event-driven architectures just as it applies to other areas of software. In particular, think about the scope of an event and whether we expect it to be consumed within the same application, the same domain or across an entire organization. A **DOMAIN-SCOPED EVENT** will be consumed within the same domain as it's published, as such we expect the consumer to have access to a certain context, resources or references in order to act on the event. If the consumption is happening more widely within an organization, the contents of the event might well need to be different, and we need to take care not to "leak" implementation details that other domains then come to depend upon.

Identity management is a critical platform component. External users on mobile apps need to be authenticated, developers need to be given access to delivery infrastructure components, and microservices may need to identify themselves to other microservices. You should ask yourself whether identity management should be "self-hosted". In our experience, a **HOSTED IDENTITY MANAGEMENT AS A SERVICE** (SaaS) solution is preferable. We believe that top-tier hosted providers such as Auth0 and Okta can provide better uptime and security SLAs. That said, sometimes self-hosting the solution is a realistic decision, especially for enterprises that have the operational discipline and resources to do so safely. Large enterprise identity solutions typically offer a much more expansive range of capabilities such as centralized entitlements, governance reporting and separation of duties management among others. However, these concerns are typically more relevant for employee identities, especially in regulated enterprises with legacy systems.

Organizations are becoming more comfortable with the **POLYCLOUD** strategy — rather than going "all-in" with one provider, they are passing different types of workloads to different providers based on their own strategy. Some of them apply the best-of-breed approach, for example: putting standard services on AWS, but using Google for machine learning and data-oriented applications and Azure for Microsoft Windows applications. For some organizations this is a cultural and business decision. Retail businesses, for example, often refuse to store their data on Amazon and they distribute load to different providers based on their data. This is different to a cloud-agnostic strategy of aiming for portability across providers, which is costly and forces lowest-common-denominator thinking. Polycloud instead focuses on using the best match that each cloud provider offers.

© ThoughtWorks, Inc. All Rights Reserved.

Some organizations are doing away with implicitly trusted intranets altogether and treating all communication as if it was being transmitted through the public internet.

(BeyondCorp)

Previously in the Radar, we've discussed the rise of the perimeterless enterprise. Now, some organizations are doing away with implicitly trusted intranets altogether and treating all communication as if it was being transmitted through the public internet. A set of practices, collectively labeled **BEYONDCORP**, have been described by Google engineers in a set of publications. Collectively, these practices — including managed devices, 802.1x networking and standard access proxies protecting individual services — make this a viable approach to network security in large enterprises.

When developing mobile applications, our teams often find themselves without an external server for testing apps. Setting up an over-the-wire mock may be a good fit for this particular problem. Developing the HTTP mocks and compiling them into the mobile binary for testing — **EMBEDDED MOBILE MOCKS** — enables teams to test their mobile apps when disconnected and with no external dependencies. This technique may require creating an opinionated library based on both the networking library used by the mobile app and your usage of the underlying library.

One pattern that comes up again and again when building microservice-style architectures is how to handle the aggregation of many resources server-side. In recent years, we've seen the emergence of a number of patterns such as Backend for Frontend (BFF) and tools such as Falcor to address this. Our teams have started using **GRAPHQL FOR SERVER-SIDE RESOURCE AGGREGATION** instead. This differs from the usual mode of using GraphQL where clients directly query a GraphQL server. When using this technique, the services continue to expose RESTful APIs but under-the-hood aggregate services use GraphQL resolvers as the implementation for stitching resources from other services. This technique simplifies the internal implementation of aggregate services or BFFs by using GraphQL.

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.

We're seeing some interesting reports of using **JUPYTER FOR AUTOMATED TESTING**. The ability to mix code, comments and output in the same document reminds us of FIT, FitNesse and Concordion. This flexible approach is particularly useful if your tests are data heavy or rely on some statistical analysis such as performance testing. Python provides all the power you need, but as tests grow in complexity, a way to manage suites of notebooks would be helpful.

One problem with observability in a highly distributed microservices architecture is the choice between logging everything — and taking up huge amounts of storage space — or randomly sampling logs and potentially missing important events. Recently, we've noticed a technique that offers a compromise between these two solutions. Set the **LOG LEVEL PER REQUEST** via a parameter passed in through the tracing header. Using a tracing framework, possibly based on the OpenTracing standard, you can pass a correlation id from service to service in a single transaction. You can even inject other data, such as the desired log level, at the initiating transaction and pass it along with the tracing information. This ensures that the additional data collected corresponds to a single user transaction as it flows through the system. This is also a useful technique for debugging, since services might be paused or otherwise modified on a transaction-by-transaction basis.

© ThoughtWorks, Inc. All Rights Reserved.

We deliberately introduce false positives into production networks and other infrastructure to check whether procedures in place are capable of identifying security failures under controlled conditions.

(Security Chaos Engineering)

We've previously talked about the technique of Chaos Engineering in the Radar and the Simian Army suite of tools from Netflix that we've used to run experiments to test the resilience of production infrastructure. **SECURITY CHAOS ENGINEERING** broadens the scope of this technique to the realm of security. We deliberately introduce false positives into production networks and other infrastructure — build-time dependencies, for example — to check whether procedures in place are capable of identifying security failures under controlled conditions. Although useful, this technique should be used with care to avoid desensitizing teams to security problems.

Increasingly, we're seeing organizations prepare to use multiple clouds — not to benefit from individual provider's strengths, though, but to avoid vendor "lock-in" at all costs.

(Generic cloud usage)

The major cloud providers continue to add new features to their clouds at a rapid pace, and under the banner of Polycloud we've suggested using multiple clouds in parallel, to mix and match services based on the strengths of each provider's offerings. Increasingly, we're seeing organizations prepare to use multiple clouds — not to benefit from individual provider's strengths, though, but to avoid vendor "lock-in" at all costs. This, of course, leads to **GENERIC CLOUD USAGE**, only using features that are present across all providers, which reminds us of the lowest common denominator scenario we saw 10 years ago when companies avoided many advanced features in relational databases in an effort to remain vendor neutral. The problem of lock-in is real. However, instead of treating it with a sledgehammer approach, we recommend looking at this problem from the perspective of exit costs and relate those to the benefits of using cloud-specific features.

© ThoughtWorks, Inc. All Rights Reserved.

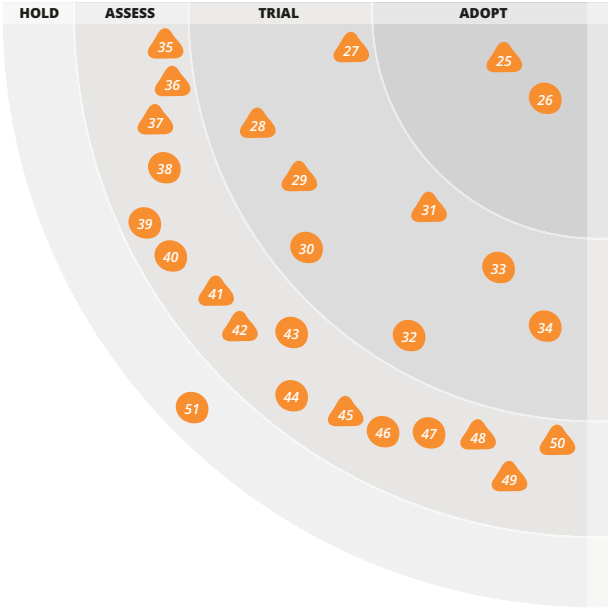
PLATFORMS

ADOPT
25. .NET Core
26. Kubernetes

TRIAL
27. Azure
28. Contentful *NEW*
29. EMQ *NEW*
30. Flood IO
31. GKE
32. Google Cloud Platform
33. Keycloak
34. WeChat

ASSESS
35. AWS Fargate *NEW*
36. Azure Service Fabric
37. Azure Stack *NEW*
38. Cloud Spanner
39. Corda
40. Cosmos DB
41. Godot *NEW*
42. Interledger *NEW*
43. Language Server Protocol
44. LoRaWAN
45. Mongoose OS *NEW*
46. Netlify
47. TensorFlow Serving
48. TICK Stack *NEW*
49. Web Bluetooth *NEW*
50. Windows Containers

HOLD
51. Overambitious API gateways



Our teams have confirmed that **.NET CORE** has reached a level of maturity that makes it the default for .NET server applications. The open source .NET Core framework enables the development and deployment of .NET applications on Windows, macOS and Linux with first-class cross-platform tooling. Microsoft provides blessed Docker images which make it easy to deploy .NET Core applications in a containerized environment. Positive directions in the community and feedback from our projects indicate that .NET Core is the future for .NET development.

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 modelling 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 by default to deliver media assets and JSON documents, good support for localization, and the ability — albeit with some effort — to integrate with Auth0.

© ThoughtWorks, Inc. All Rights Reserved.

EMQ is a scalable open source multiplatform MQTT broker. It's written in Erlang/OTP for higher performance, handling millions of concurrent connections. It supports multiple protocols including MQTT, MQTT Sensor Networks, CoAP as well as WebSockets, making it suitable for both IoT and mobile devices. We've started using EMQ in our projects and have enjoyed its ease of installation and use, its ability to route messages to different destinations including Kafka and PostgreSQL, as well as its API-driven approach for its monitoring and configuration.

While the software development ecosystem is converging on Kubernetes as the orchestration platform for containers, running Kubernetes clusters remains operationally complex. Google Kubernetes Engine (**GKE**) is a managed Kubernetes solution for deploying containerized applications that alleviates the operational overhead of running and maintaining Kubernetes clusters. Our teams have had a good experience using GKE, with the platform doing the heavy lifting of applying security patches, monitoring and auto-repairing the nodes, and managing multicluster and multiregion networking. In our experience, Google's API-first approach in exposing platform capabilities, as well as using industry standards such as OAuth for service authorisation, improve the developer experience. It's important to consider that GKE is under rapid development with many of its APIs in beta release which, despite the developers' best efforts to abstract consumers from underlying changes, can impact you. We're expecting continuous improvement around maturity of infrastructure as code with Terraform on GKE and similar tools.

AWS FARGATE is a recent entry into the docker-as-a-service space, currently limited to the US-East-1 region. For teams using AWS Elastic Container Service (ECS), AWS Fargate is a good alternative without having to manage, provision and configure any underlying EC2 instances or clusters. Fargate allows defining (ECS or EKS – ECS for Kubernetes) tasks as a Fargate type, and they will run on the AWS Fargate infrastructure. If you like the focus on business functionality that AWS Lambda gives you, Fargate is the closest you can get when applications can't be deployed as single functions.

AZURE SERVICE FABRIC is a distributed systems platform built for microservices and containers. It can act as a PaaS with its reliable services, or like a container orchestrator with its ability to manage containers. What distinguishes Service Fabric though are programming models such as Reliable Actors built on top of reliable services. When it comes to IoT use cases, for example, Reliable Actors offers some compelling advantages — in addition to the reliability and platform benefits of being on Service Fabric, you also get its state management and replication capabilities. In keeping with continued focus on open source software (OSS), Microsoft will be transitioning Service Fabric to an open development process on Github. All this makes Azure Service Fabric worth trialling — particularly for organizations who are invested in the .NET framework.

Microsoft adds an interesting offering as a middle ground between full-featured public clouds and simple on-premises virtualization.

(Azure Stack)

Cloud computing brings significant benefits over self-hosted virtualized solutions but sometimes data simply cannot leave an organization's premises, usually for latency or regulatory reasons. For European companies, the current political climate also raises more concerns about placing data in the hands of US-based entities. With **AZURE STACK**, Microsoft adds an interesting offering as a middle ground between full-featured public clouds and simple on-premises virtualization: a slimmed-down version of the software that runs Microsoft's Azure Global cloud is combined with a rack of preconfigured commodity hardware from the usual suspects like HP and Lenovo, providing an organization with the core Azure experience on premises. By default, support is split between Microsoft and the hardware vendors (and they promise to cooperate), but system integrators can offer complete Azure Stack solutions, too.

As AR and VR continue to gain traction, we continue to explore tools with which we can create immersive virtual worlds. Our positive experience with Unity, one of the two major gaming engines, led us to feature it in previous Radars. We still like Unity but are also excited about **GODOT**, a relatively new entrant to the field. Godot is open source software and although not as fully featured as the big commercial engines, it comes with a more modern software design and less clutter. Offering C# and Python further lowers the barrier to entry for developers outside the gaming industry. Godot version 3, released earlier this year, adds support for VR and support for AR is on the horizon.

© ThoughtWorks, Inc. All Rights Reserved.

Most people may know the "Internet of money" through Bitcoin. In fact, this idea can be traced to the early stages of the Web. HTTP even reserved a status code for digital payment.

(Interledger)

Most people may know the "Internet of money" through Bitcoin. In fact, this idea can be traced to the early stages of the Web. HTTP even reserved a status code for digital payment. The challenging part of this idea is to transfer value between different ledgers in different entities. Blockchain technology promotes this idea through building a distributed shared ledger. The current challenge is how to achieve interoperability between different blockchain ledgers and interoperability with traditional centralized ledgers. **INTERLEDGER** is a protocol to connect different ledgers. This protocol uses connectors and a cryptographic mechanism such as HTLC to route secure payments across ledgers. It's not hard to join the payment network through its suites. Interledger was first initiated by Ripple and is now steadily developed by a W3C community group.

With an accelerated growth of connected embedded devices and wider accessibility of hardware, **MONGOOSE OS** fills a noticeable gap for embedded software developers: the gap between Arduino firmware suitable for prototyping and bare-metal microcontrollers' native SDKs. Mongoose OS is a microcontroller operating system that comes with a set of libraries and a development framework to support typical Internet of Things (IoT) applications with connectivity to generic MQTT servers and popular IoT cloud platforms such as Google Cloud IoT Core and AWS IoT by default. In fact, Google recommends a Mongoose starter kit for its Cloud IoT Core. We've had a seamless experience using Mongoose OS in our embedded projects building connected workspaces. We especially liked its built-in security at the individual device level and OTA firmware updates, among other features. At the time of writing, only a limited number of microcontrollers and boards are supported with more popular ARM-based microcontrollers still under development.

TICK STACK is a platform composed of open source components which makes collection, storage, graphing and alerting on-time series data such as metrics and events easy. The components of the TICK Stack 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, the 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. This stack is backed by InfluxData and needs the enterprise version for features such as DB clustering, but 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.

WEB BLUETOOTH allows us to control any Bluetooth Low Energy device directly from the browser. This allows us to target scenarios that previously could only be solved with a native app. The specification is published by the Web Bluetooth Community Group and describes an API to discover and communicate with devices over the Bluetooth 4 wireless standard. Right now, Chrome is the only major browser which currently supports this specification. With Physical Web and Web Bluetooth, we now have other avenues for getting users to interact with devices without them having to install yet another app on their phone. This is an exciting space which is worth keeping an eye on.

Microsoft is catching up in the container space with **WINDOWS CONTAINERS** enabling running Windows applications as containers on Windows-based environments. At the time of writing, Microsoft provides two Windows OS images as Docker containers — Windows Server 2016 Server Core and Windows Server 2016 Nano Server — that can run as a Windows Server Container with Docker. Our teams have started using Windows containers in scenarios where build agents and similar containers have been working successfully. Microsoft is aware that there's room for improvements such as decreasing the large image sizes and enriching ecosystem support and documentation.

© ThoughtWorks, Inc. All Rights Reserved.

TOOLS

ADOPT

TRIAL
52. Appium Test Distribution *NEW*
53. BackstopJS *NEW*
54. Buildkite
55. CircleCI
56. CVXPY *NEW*
57. gopass
58. Headless Chrome for front-end test
59. Helm *NEW*
60. Jupyter
61. Kong API Gateway
62. kops
63. Patroni *NEW*
64. WireMock *NEW*
65. Yarn

ASSESS
66. Apex
67. ArchUnit *NEW*
68. cfn_nag *NEW*
69. Conduit *NEW*
70. Cypress
71. Dependabot *NEW*
72. Flow
73. Headless Firefox *NEW*
74. nsp *NEW*
75. Parcel *NEW*
76. Scout2 *NEW*
77. Sentry *NEW*
78. Sonobuoy
79. Swashbuckle for .NET Core *NEW*

HOLD

We've featured Appium in the Radar in the past. It's one of the most popular mobile test automation frameworks. As we scale our test suite, being able to run our tests in parallel against an array of devices is key in having short feedback loops. **APPIUM TEST DISTRIBUTION** solves this problem very effectively with its ability to run tests in parallel as well as run the same tests on multiple devices. Among other things, it distinguishes itself by its ability to add and remove devices in which tests run without any manual setup required and with its ability to run tests on remote devices. We've used it in a few projects at ThoughtWorks over the last couple of years and it worked very well for us.

We've been enjoying **BACKSTOPJS** for visual regression testing of web applications. The configurable viewports and ability to adjust tolerances are particularly useful, as is the visual comparison tool, which makes it easier to spot minor variations. It has good scriptability and the option to run in Headless Chrome, PhantomJS and SlimerJS. We find it particularly helpful when running it against living component style guides.

It is surprising how many problems can be expressed as mathematical optimizations and often convex problems that can be efficiently solved.
(CVXPY)

It's surprising how many problems can be expressed as mathematical optimization problems and often convex problems that can be efficiently solved. **CVXPY** is an open source Python-embedded modeling language for convex optimization problems. It's maintained by academics at Stanford University and offers a batteries-included install for several open source and commercial solvers. The documentation includes many examples which should inspire developers to use it. It's particularly useful for prototyping solutions even though commercially licensed solvers, such Gurobi or IBM CPLEX, may be required. In most cases though, it suffices by itself. However, the same group has written many extension packages such as DCCP and related software such as CVXOPT based on recent advances in optimization.

© ThoughtWorks, Inc. All Rights Reserved.

HELM is a package manager for Kubernetes. The set of Kubernetes resources that together define an application is packaged as charts. These charts can describe a single resource, such as a Redis pod, or a full stack of a web application: HTTP servers, databases and caches. Helm, by default, comes with a repository of curated Kubernetes applications that are maintained in the official charts repository. It's also easy to set up a private chart repository for internal usage. Helm has two components: a command line utility called Helm and a cluster component called Tiller. Securing a Kubernetes cluster is a wide and nuanced topic, but we highly recommend setting up Tiller in a role-based access control (RBAC) environment. We've used Helm in a number of client projects and it's dependency management, templating and hook mechanism has greatly simplified the application lifecycle management in Kubernetes.

Over the last couple of years, we've noticed a steady rise in the popularity of analytics notebooks. These are Mathematica-inspired applications that combine text, visualization and code in a living, computational document.

(Jupyter)

Over the last couple of years, we've noticed a steady rise in the popularity of analytics notebooks. These are Mathematica-inspired applications that combine text, visualization and code in a living, computational document. Increased interest in machine learning — along with the emergence of Python as the programming language of choice for practitioners in this field — has focused particular attention on Python notebooks, of which **JUPYTER** seems to be gaining the most traction among ThoughtWorks teams. People seem to keep finding creative uses for Jupyter beyond a simple analytics tool. For example, see Jupyter for automated testing.

Kong is an open source API gateway which also comes as an enterprise product integrating with proprietary API analytics and a developer portal. Kong can be deployed, in a variety of configurations, as an edge API gateway, as an internal API proxy, or even as a sidecar in a service mesh configuration. OpenResty, through its Nginx modules, provides a strong and performant foundation, with Lua plugins for extensions. Kong can either use PostgreSQL for single-region deployments or Cassandra for multiregion configurations. Our developers have enjoyed Kong's high performance, its API-first approach (which enables automation of its configuration) and its ease of deployment as a container. **KONG API GATEWAY**, unlike overambitious API gateways, has a smaller set of features but it implements the essential set of API gateway capabilities such as traffic control, security, logging, monitoring and authentication.

KOPS is a command line tool for creating and managing high-availability production Kubernetes clusters. kops has become our go-to tool to self-manage Kubernetes clusters on AWS, not the least because of its rapidly growing open source community. It also supports installing, upgrading and managing Kubernetes clusters on Google Cloud. Our experience with kops on Google, however, is very limited because of our preference for GKE, the managed Kubernetes offering. We recommend using kops in reusable scripts to create infrastructure as code. We're interested to see how kops continues to evolve to support managed Kubernetes clusters such as EKS, Amazon's own managed Kubernetes service.

PATRONI is a template for PostgreSQL high availability. Born out of the need to provide automatic failure for PostgreSQL, Patroni is a Python-based PostgreSQL controller that leverages a distributed configuration store (such as etcd, ZooKeeper, or Consul) to manage the state of the PostgreSQL cluster. Patroni supports both streaming and synchronous replication models and provides a rich set of REST APIs for dynamic configuration of the PostgreSQL cluster. If you want to achieve high availability in a distributed PostgreSQL setup, you have to consider many edge cases, and we like the fact that Patroni provides a template to achieve most of the common use cases.

A key driver for architectures based on microservices is independent evolvability of services. For example, when two services depend on each other, the testing process for one usually involves stubs and mocks for the other one. These can be written by hand, but as with mocking in unit tests, a framework helps developers focus on the actual test scenario. We have known of **WIREMOCK** for a while but we've preferred running tests with mountebank. Over the past year, though, WireMock has really caught up and we now recommend it as a good alternative.

© ThoughtWorks, Inc. All Rights Reserved.

YARN is a fast, reliable and secured package manager for JavaScript. Using a lock file and a deterministic algorithm, Yarn is able to guarantee that an installation that worked on one system will work exactly the same way on any other system. By efficiently queuing up requests, Yarn maximizes network utilization and as a result we've seen faster package downloads. Yarn continues to be our tool of choice for JavaScript package management in spite of the latest improvements in npm (version 5).

ARCHUNIT is a Java testing library for checking architecture characteristics such as package and class dependencies, annotation verification and even layer consistency. The fact that it runs as unit tests, within your existing test setup, pleases us, even though it's available for Java architectures only. 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.

The cloud and continuous delivery had a dramatic effect on infrastructure security. When following infrastructure as code, the entire infrastructure — which includes networks, firewalls and accounts — is defined in scripts and configuration files, and with Phoenix Servers and Environments, the infrastructure is recreated in each deployment, often many times a day. In such a scenario, testing the infrastructure after it's created is neither sufficient nor feasible. A tool that helps address this problem is **CFN_NAG**. It scans the CloudFormation templates used with AWS for patterns that may indicate insecure infrastructure, and it does so before the infrastructure is created. Running a tool such as cfn_nag in a build pipeline is fast and it can detect a number of problems before they even reach a cloud environment.

CONDUIT is a lightweight service mesh for Kubernetes. Conduit embraces the out-of-process architecture with data plane proxy written in Rust and a control plane in Go. The data plane proxy runs as a sidecar for all TCP traffic in the Kubernetes cluster and the control plane runs in a separate namespace in Kubernetes exposing REST APIs to control the behavior of the data plane proxy. By proxying all requests, Conduit provides a wealth of metrics for monitoring and observability of interactions in the service mesh for HTTP, HTTP/2 and gRPC traffic. Even though Conduit is relatively new to this space, we recommend it because it's simple to install and operate.

Keeping dependencies up to date is a chore, but it's important to manage upgrades frequently and incrementally. We want the process to be as painless and automated as possible. Our teams have often hand-rolled scripts to automate parts of the process; now, however, we integrate commercial offerings to do that work. **DEPENDABOT** is a service that integrates with your GitHub repositories and automatically checks your project dependencies for new versions. When required, Dependabot will open a pull request with upgraded dependencies. Using features of your CI server, you can automatically test upgrades for compatibility and automatically merge compatible upgrades to master. There are alternatives to Dependabot, including Renovate for JavaScript projects and Depfu for JavaScript and Ruby projects. Our teams, however, recommend Dependabot because of its multilanguage support and ease of use.

When developing front-end applications, we've mentioned Headless Chrome as a better alternative to PhantomJS for front-end testing in a previous edition of the Radar. Now we suggest assessing **HEADLESS FIREFOX** as a viable option in this area. In the same way as Headless Chrome, Firefox in a headless mode runs the browser without the visible UI components, executing the UI tests suite much faster.

NSP is a command line tool to identify known vulnerabilities in Node.js applications. By running the check command on the root of a Node.js project, nsp generates the vulnerabilities report by checking against the published advisories. nsp provides a way to customize the check command to hide all vulnerabilities below the given CVSS score or exit with an error code if at least one finding has a CVSS score above the given value. Once the advisories are saved through the gather command, nsp can also be used in offline mode.

PARCEL is a web application bundler similar to Webpack or Browserify. We've featured Webpack previously in our Radar and it continues to be a great tool. Parcel distinguishes itself from its rivals through developer experience and speed. It has all the standard bundling features and provides true zero-configuration experience, making it really easy to get started with and use. It has fast bundle times and beats its competitors in many benchmarks. Parcel has gained a lot of community interest and is worth keeping an eye on.

© ThoughtWorks, Inc. All Rights Reserved.

SCOUT2 is a security auditing tool for AWS environments. Instead of manually navigating through web pages, you can rely on Scout2 to fetch all the configuration data of an AWS environment for you; it even generates an attack surface report. Scout2 ships with preconfigured rules and can be easily extended to support more services and test cases. Since Scout2 only performs AWS API calls to fetch configuration data and identify security gaps, it is not necessary to complete and submit the AWS Vulnerability / Penetration Testing Request Form.

SENTRY is an error-tracking tool that helps monitor and fix errors in real time. Error tracking and management tools such as Sentry distinguish themselves from traditional logging solutions such as the ELK Stack in their focus on discovering, investigating and fixing errors. Sentry has been around for some time and is quite popular — error-tracking tools are increasingly useful with the current focus on "mean time to recovery". Sentry — with its integration options with Github, Hipchat, Heroku, Slack, among other platforms — enables us to keep a close eye on our apps. It can provide error notifications following a release, enable us to track whether new commits actually fix the issue and alert us if an issue comes back due to a regression.

In the current state of technology services, exposing RESTFul APIs is increasingly adopted and API documentation is very important for consumers.

(Swashbuckle for .NET Core)

In the current state of technology services, exposing RESTFul APIs is increasingly adopted and API documentation is very important for consumers. In this space, Swagger has been largely used across teams and we would like to highlight **SWASHBUCKLE FOR .NET CORE**. Swashbuckle for .NET Core is a tool that generates living API documentation in Swagger, based on the code for .NET Core projects. When using it, you can also explore and test operations of APIs through its UI.

© ThoughtWorks, Inc. All Rights Reserved.

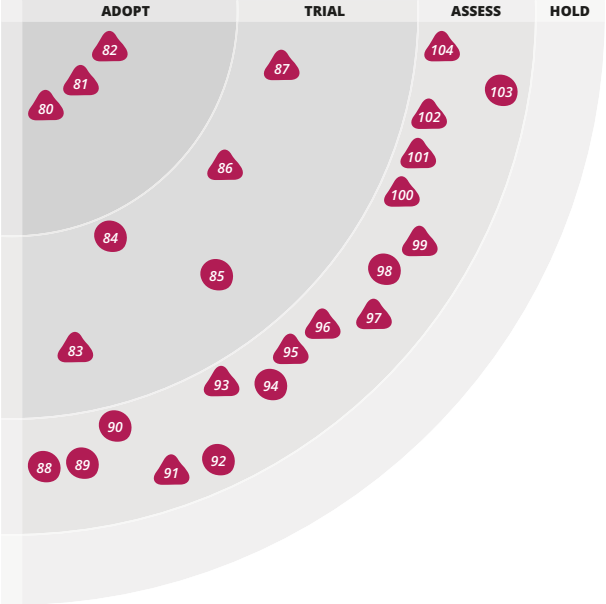
LANGUAGES & FRAMEWORKS

ADOPT
80. AssertJ
81. Enzyme
82. Kotlin

TRIAL
83. Apollo *NEW*
84. CSS Grid Layout
85. CSS Modules
86. Hyperledger Composer *NEW*
87. OpenZeppelin *NEW*

ASSESS
88. Android Architecture Components
89. Atlas and BeeHive
90. Clara rules
91. Flutter *NEW*
92. Gobot
93. Hyperapp *NEW*
94. PyTorch
95. Rasa *NEW*
96. Reactor *NEW*
97. RIBs *NEW*
98. Solidity
99. SwiftNIO *NEW*
100. Tensorflow Eager Execution *NEW*
101. TensorFlow Lite *NEW*
102. troposphere *NEW*
103. Truffle
104. WebAssembly *NEW*

HOLD



ASSERTJ is a Java library that provides a fluent interface for assertions, which makes it easy to convey intent within test code. AssertJ gives readable error messages, soft assertions and improved collections and exception support. Many of our teams choose AssertJ as their default assertion library instead of JUnit combined with Java Hamcrest.

ENZYME has become the defacto standard for unit testing React UI components. Unlike many other snapshot-based testing utilities, Enzyme enables you to test without doing on-device rendering, which results in faster and more granular testing. This is a contributing factor in our ability to massively reduce the amount of functional testing we find we have to do in React applications. In many of our projects it's used within a unit testing framework such as Jest.

KOTLIN has experienced an accelerated rate of adoption and rapid growth of tooling support. Some of the reasons behind its popularity are its concise syntax, null safety, ease of transition from Java and interoperability with other JVM-based languages in general, and that it doubles as a great introductory language to functional programming. With JetBrains adding the ability to compile Kotlin to native binaries on multiple platforms, as well as transpile to JavaScript, we believe it has the potential of much wider use by the larger community of mobile and native application developers. Although at the time of writing, some of the tooling such as static and coverage code analysis have yet to mature, given our experience of using Kotlin in many production applications, we believe Kotlin is ready for general adoption.

© ThoughtWorks, Inc. All Rights Reserved.

Since it was first introduced in the Radar, we've seen a steady adoption of GraphQL, particularly as a remote interface for a Backend for Frontend (BFF). As they gain more experience, our teams have reached consensus on Apollo, a GraphQL client, as the preferred way to access GraphQL data from a React application. Although the **APOLLO** project also provides a server framework and a GraphQL gateway, the Apollo client simplifies the problem of binding UI components to data served by any GraphQL backend. Notably, Apollo is used by Amazon AWS in their recent launch of the new AWS AppSync service.

The Hyperledger project has grown into a broader collaboration and now contains a series of subprojects. It supports Blockchain implementations for different purposes; for example, Burrow is dedicated to build a permissioned Ethereum and Indy is more focused on digital identity. Among these platforms, Fabric is the most mature one. Most of time when people talk about adopting Hyperledger they are actually thinking about Hyperledger Fabric. However, the programming abstraction of chaincode is relatively low level given it manipulates the state of the ledger directly. Moreover, it always takes a lot of time to set up infrastructure before writing the first line of blockchain code. **HYPERLEDGER COMPOSER**, which builds on top of Fabric, accelerates the process of turning ideas into software. Composer provides DSLs to model business assets, define access control and build a business network. By using Composer you could quickly validate your idea through a browser without setting up any infrastructure. Just remember that the Composer itself isn't Blockchain — you still need to deploy it on Fabric.

Security is the cornerstone of the blockchain economy.

(OpenZeppelin)

Security is the cornerstone of the blockchain economy. In the last issue of the Radar, we highlighted the importance of testing and auditing smart contracts dependencies. **OPENZEPPELIN** is a framework to help build secure smart contracts in Solidity. The team behind OpenZeppelin summed up a series of pitfalls and best practices around smart contracts' security and embedded these experiences into the source code. The framework is well reviewed and validated by the open source community. We recommend the use of OpenZeppelin instead of writing your own implementation of the ERC20/ERC721 token. OpenZeppelin is also integrated with Truffle.

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.

Given the number of JavaScript application frameworks we've featured in the Radar over the years we asked ourselves, do we really need to call out another one? We decided that **HYPERAPP** is worth a look because of its minimalist approach. It has a very small footprint, less than 1KB, and yet covers all the essential functionality for writing a web application. This is only possible with an elegant design that reduces everything to the absolute minimum, which in turn makes it easier to understand and use the framework. Despite being relatively new, it has attracted a good-size community and we recommend to at least consider it when picking a framework for a new application.

RASA is a new entrant in the area of chatbots. Instead of using a simple decision tree it uses neural networks to map intent and internal state to a response. Rasa integrates with natural language processing solutions such as spaCy; and, unlike other solutions we've featured in the Radar, Rasa is open source software and can be self-hosted, which makes it a viable solution when ownership of data is of concern. Our experiences with using Rasa Stack for an internal application have been positive.

REACTOR is a library for building non-blocking applications on the JVM — version 8 and above — based on the Reactive Streams specification. Reactive programming emphasizes moving from imperative logic to asynchronous, non-blocking and functional style code, especially when dealing with external resources. Reactor implements the reactive stream specification and provides two publisher APIs — Flux (0 to N elements) and Mono (0 or 1 element) — to effectively model push-based stream processing. Reactor project is well suited for microservices architecture and offers back pressure–ready network engines for HTTP, WebSockets, TCP and UDP traffic.

© ThoughtWorks, Inc. All Rights Reserved.

RIBs — which is short for router, interactor and builder — is a cross-platform architecture mobile framework from Uber. The key idea of RIBs is to decouple business logic from the view tree, and thus ensure the app is driven by business logic. This is actually an application of Clean Architecture in mobile application development. By applying consistent architecture patterns across native Android and iOS, RIBs provides clear statement management and good testability. We advise putting business logic in the back-end service rather than leak it into the view, so if you do have a complicated mobile application, RIBs can help manage this complexity.

We're in favor of asynchronous and reactive styles of programming especially for network I/O-bound distributed systems. Reactive libraries often sit on top of a lower level nonblocking communication framework such as Netty. Recently **SWIFTNIO**, an open source nonblocking networking framework from Apple, has grabbed our attention. SwiftNIO is similar to Netty but written in Swift. It's currently supported on MacOS and Ubuntu and implements HTTP as a higher-level protocol. We're excited to see the usage of this framework and integration of it into higher-level application frameworks and other protocols.

In the last issue we featured PyTorch, a deep-learning modeling framework that allows an imperative programming style. Now **TENSORFLOW EAGER EXECUTION** provides this imperative style in TensorFlow by enabling execution of modeling statements outside of the context of a session. This improvement could provide the ease of debugging and finer-grained model control of PyTorch with the widespread popularity and performance of TensorFlow models. The feature is still quite new so we're anxious to see how it performs and how it'll be received by the TensorFlow community.

TENSORFLOW LITE is the designated successor of TensorFlow Mobile, which we mentioned in our previous Radar. Like Mobile it is a lightweight solution tuned and optimized for mobile devices (Android and iOS). We expect the standard use case to be the deployment of pretrained models into mobile apps but TensorFlow Lite also supports on-device learning which opens further areas of application.

We're trying out **TROPOSPHERE** as a way of defining the infrastructure as code on AWS for our projects where AWS CloudFormation is used 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 binary format designed to run in the browser at near native speeds, and opens up the range of languages you can use to write front-end functionality.

(WebAssembly)

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.

© ThoughtWorks, Inc. All Rights Reserved.

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

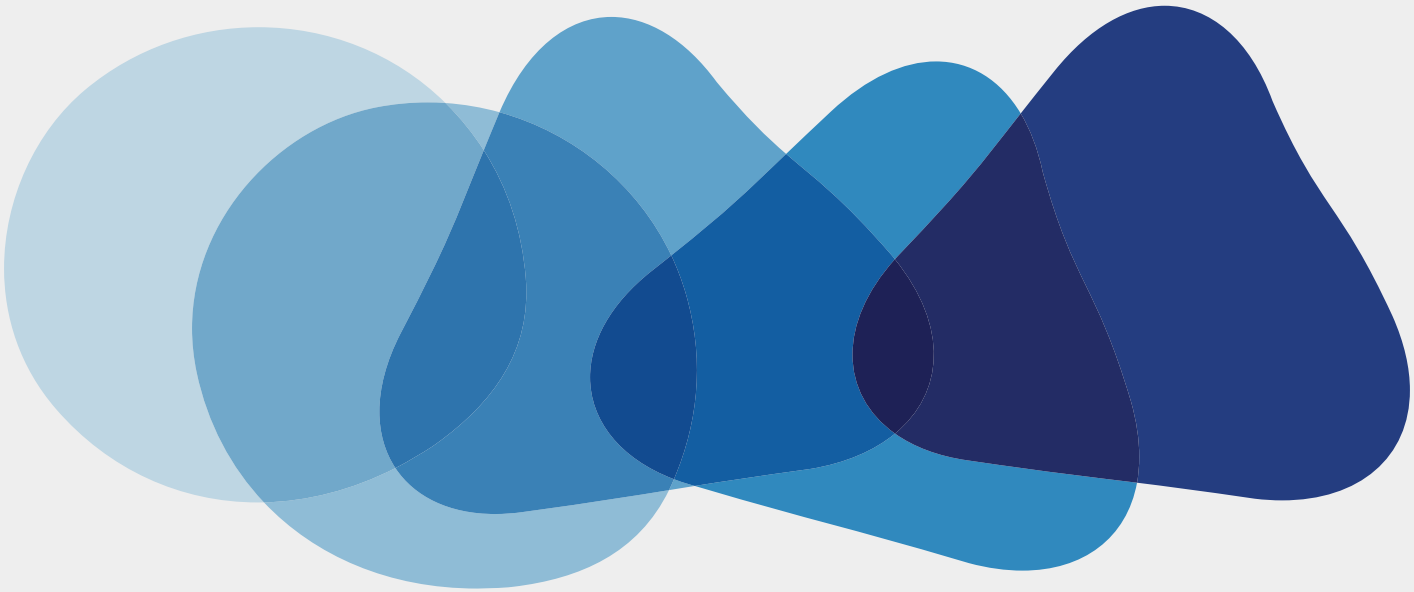
SUBSCRIBE NOW

thght.works/Sub-EN

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 organisations striving in the same direction.

Founded over 20 years ago, ThoughtWorks has grown to a company of over 4500 people, including a products division which 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

#TWTechRadar