ThoughtWorks®

TECHNOLOGY RADAR *NOV '16*

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

thoughtworks.com/radar

#TWTechRadar

WHAT'S NEW?

Here are the themes highlighted in this edition:

"DOCKER AS PROCESS, PAAS AS MACHINE, MICROSERVICES ARCHITECTURE AS PROGRAMMING MODEL"

The microservices style of architecture highlights rising abstractions in the developer world because of containerization and the emphasis on low coupling, offering a high level of operational isolation. Developers can think of a container as a self-contained process and the PaaS as the common deployment target, using the microservices architecture as the common style. Decoupling the architecture allows the same for teams, cutting down on coordination cost among silos. Its attractiveness to both developers and DevOps has made this the de facto standard for new development in many organizations.

INTELLIGENT EMPOWERMENT

Long-time R&D topics like machine learning and artificial intelligence suddenly have practical applications through frameworks like Nuance Mix and TensorFlow. Developers can download frameworks that range from NLP to machine learning libraries. We happily observe that companies frequently open source sophisticated libraries and tools in this space that would have been stratospherically expensive and therefore restricted a decade ago, making them available to a wide audience of developers. Many factors have evolved and combined to make new tools possible: commodity computing, targeting specific hardware like GPUs and cloud resources. Perhaps your Big Data hoarding is starting to pay off....

THE HOLISTIC EFFECT OF TEAM STRUCTURE

Team structure has always had a large impact on a wide variety of software development subjects, and has become an area of increased focus given foundations such as self-service PaaS and microservices. Companies now favor product thinking over projects; tech companies are popularizing the "you build it, you run it" style of team autonomy, and we're seeing the same product thinking applied to enterprise projects. When restructuring teams yields better results, it illustrates once again that software development is still mostly a communication problem. Building cross-functional teams increases the beneficial surface area of communication across traditionally segregated job roles, which in turn removes friction imposed by artificial structures like silos.

AR AND VR EASING TOWARD MAINSTREAM

We see augmented and virtual reality (AR/VR) generating business interest, both technologies that were once relegated merely to games and novelty. Whereas chasing virtual cartoons brought AR to public attention via mobile SDKs, hardware such as the Oculus Rift, HTC Vive and Microsoft HoloLens is maturing to the point that early adopters can reap benefits without fumbling with immature technology. Although software platforms like OpenVR and Unity have long been mature, new natural language processing (NLP) tools like Nuance Mix and hardware that provides natural interactions will have a huge impact on the adoption of AR and VR. We are already running VR and AR labs in our offices to explore future applications like remote interactions or retail wayfinding. Our experiments demonstrate VR as a surprisingly powerful medium for more empathetic remote collaboration and storytelling due to its ability to bypass abstraction and immerse users directly in an experience. However, we will see significant challenges in the creation and delivery of VR and AR content as the skills and capabilities lag behind the pace of hardware, particularly in the enterprise.

CONTRIBUTORS

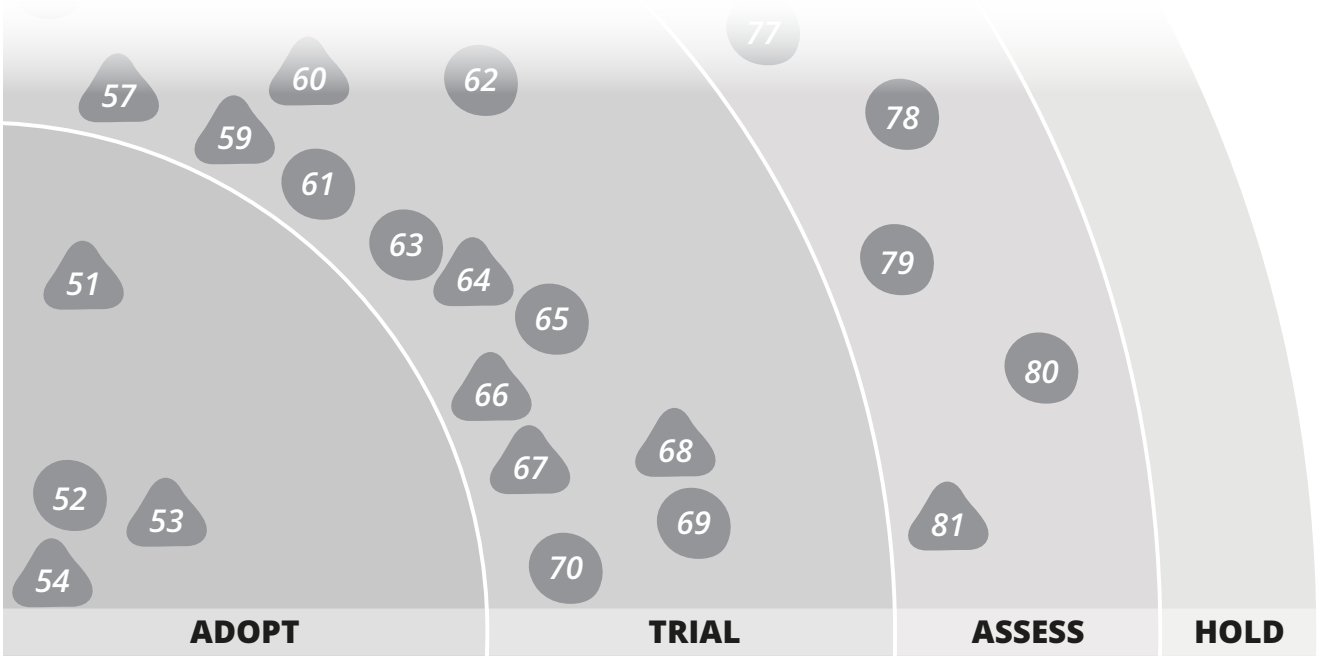
The Technology Radar is prepared by the ThoughtWorks Technology Advisory Board, comprised of:

Rebecca Parsons (CTO)
Martin Fowler (Chief Scientist)
Anne J Simmons
Badri Janakiraman
Bharani Subramaniam
Camilla Falconi Crispim
Erik Doernenburg
Evan Bottcher
Fausto de la Torre
Hao Xu
Ian Cartwright
James Lewis
Jiaxing Chen
Jonny LeRoy
Marco Valtas
Mike Mason
Neal Ford
Rachel Laycock
Scott Shaw
Srihari Srinivasan
Zhamak Dehghani

ABOUT THE TECHNOLOGY 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 CIOs to developers. 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. The rings are:



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

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.

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

Proceed with caution.

Items that are new or have had significant changes since the last radar are represented as triangles, while items that have not moved are represented as circles. We are interested in far more items than we can reasonably fit into a document this size, so we fade many items from the last radar to make room for the new items. Fading an item does not mean that we no longer care about it.

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

THE RADAR

TECHNIQUES

ADOPT
1. Consumer-driven contract testing
2. Pipelines as code new
3. Threat Modeling

TRIAL
4. APIs as a product new
5. Bug bounties
6. Data Lake
7. Hosting PII data in the EU
8. Lightweight Architecture Decision Records new
9. Reactive architectures
10. Serverless architecture

ASSESS
11. Client-directed query new
12. Container security scanning new
13. Content Security Policies
14. Differential privacy new
15. Micro frontends new
16. OWASP ASVS
17. Unikernels
18. VR beyond gaming

HOLD
19. A single CI instance for all teams
20. Anemic REST new
21. Big Data envy
22. Cloud lift and shift

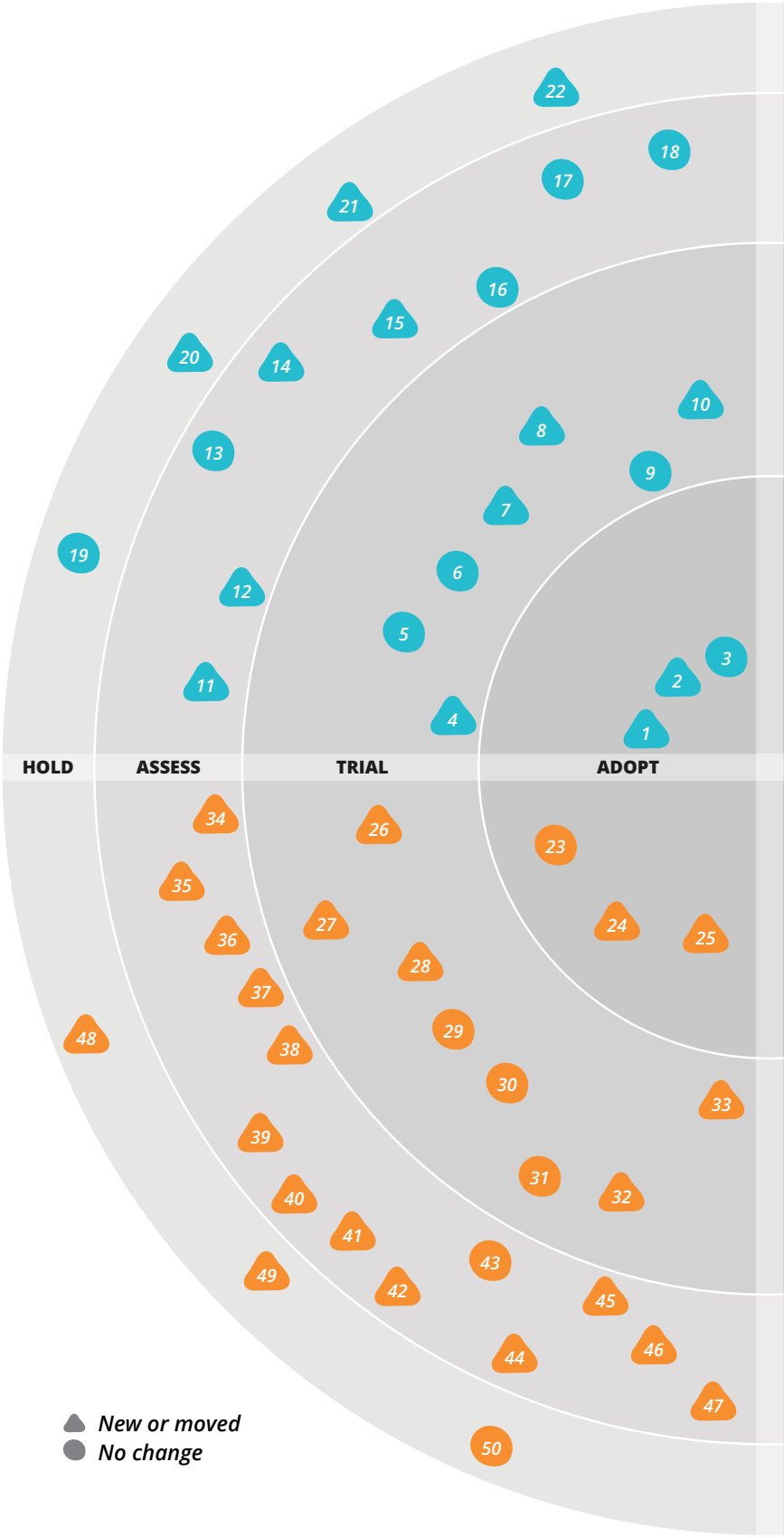
PLATFORMS

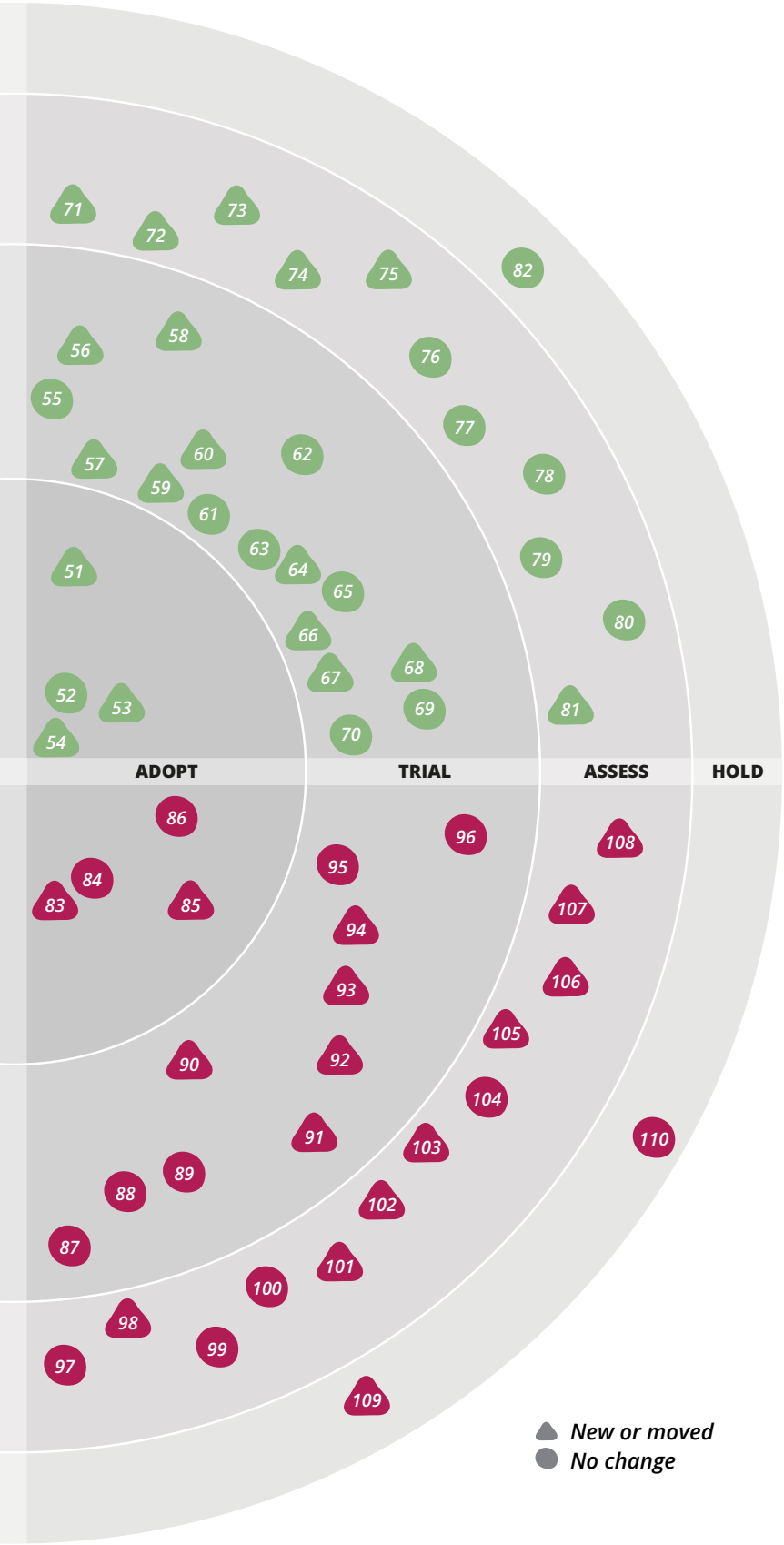
ADOPT
23. Docker
24. HSTS
25. Linux security modules

TRIAL
26. Apache Mesos
27. Auth0 new
28. AWS Lambda
29. Kubernetes
30. Pivotal Cloud Foundry
31. Rancher
32. Realm
33. Unity beyond gaming new

ASSESS
34. .NET Core
35. Amazon API Gateway
36. Apache Flink
37. AWS Application Load Balancer new
38. Cassandra carefully new
39. Electron new
40. Ethereum new
41. HoloLens new
42. IndiaStack new
43. Nomad
44. Nuance Mix new
45. OpenVR new
46. Tarantool new
47. wit.ai new

HOLD
48. CMS as a platform
49. Overambitious API gateway
50. Superficial private cloud





THE RADAR

TOOLS

ADOPT
51. Babel new
52. Consul
53. Grafana new
54. Packer

TRIAL
55. Apache Kafka
56. Espresso
57. fastlane new
58. Galen new
59. HashiCorp Vault
60. JSONassert new
61. Let's Encrypt
62. Load Impact
63. OWASP Dependency-Check
64. Pa11y new
65. Serverspec
66. Talisman new
67. Terraform
68. tmate new
69. Webpack
70. Zipkin

ASSESS
71. Android-x86 new
72. axios new
73. Bottled Water new
74. Clojure.spec new
75. FBSnapshotTestcase new
76. Grasp
77. LambdaCD
78. Pinpoint
79. Pitest
80. Repsheet
81. Scikit-learn new

HOLD
82. Jenkins as a deployment pipeline

LANGUAGES & FRAMEWORKS

ADOPT
83. Ember.js
84. React.js
85. Redux
86. Spring Boot

TRIAL
87. Butterknife
88. Dagger
89. Dapper
90. Elixir
91. Enzyme new
92. Immutable.js
93. Phoenix new
94. Quick and Nimble new
95. React Native
96. Robolectric

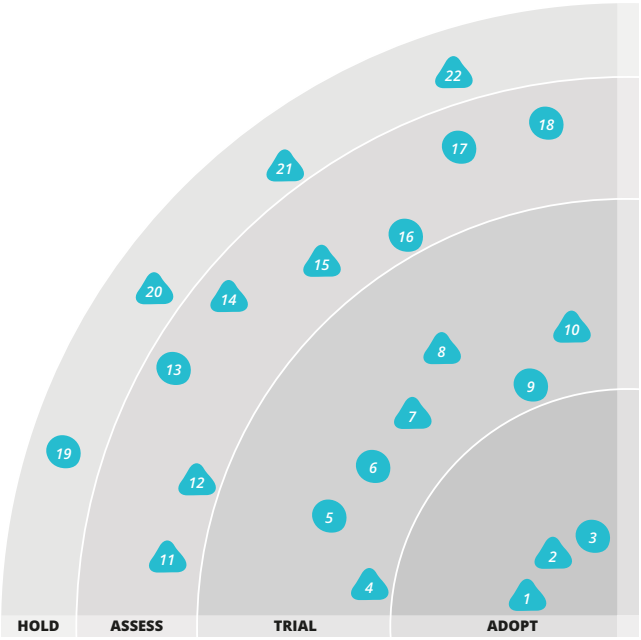
ASSESS
97. Aurelia
98. ECMAScript 2017 new
99. Elm
100. GraphQL
101. JuMP new
102. Physical Web new
103. Rapidoid new
104. Recharts
105. ReSwift new
106. Three.js new
107. Vue.js new
108. WebRTC new

HOLD
109. AngularJS
110. JSPatch

TECHNIQUES

We've decided to bring **consumer-driven contract testing** back from the archive for this edition even though we had allowed it to fade in the past. The concept isn't new, but with the mainstream acceptance of microservices, we need to remind people that consumer-driven contracts are an essential part of a mature microservice testing portfolio, enabling independent service deployments. But in addition, we want to point out that consumer-driven contract testing is a technique and an attitude that requires no special tool to implement. We love frameworks like Pact because they make proper contract tests easier to implement in certain contexts. But we have noticed a tendency for teams to focus on the framework rather than on the general practice. Writing Pact tests is not a guarantee that you are creating consumer-driven contracts; likewise, in many situations you should be creating good consumer-driven contracts even where no pre-built testing tool exists.

Teams are pushing for automation across their environments, including their development infrastructure. **Pipelines as code** is defining the deployment pipeline through code instead of configuring a running CI/CD tool. LambdaCD, Drone, GoCD and Concourse are examples that allow usage of this technique. Also, configuration automation tools for CI/CD systems like GoMatic can be used to treat the deployment pipeline as code—versioned and tested.



Businesses have wholeheartedly embraced APIs as a way to expose business capabilities to both external and internal developers. APIs promise the ability to experiment quickly with new business ideas by recombining core capabilities. But what differentiates an API from an ordinary enterprise integration service? One difference lies in treating **APIs as a product**, even when the consumer is an internal system. Teams that build APIs should understand the needs of their customers and make the product compelling to them. Products are also improved, maintained and supported over the long term. They should have an owner who advocates for the customer and strives for continual improvement. Products are actively maintained and supported, easy to find and easy to use. In our experience, a product orientation is the missing ingredient that makes the difference between ordinary enterprise integration and an agile business built on a platform of APIs.

In a number of countries, we see government agencies seeking broad access to private, personally identifiable information (PII). The increased use of public cloud solutions makes it more difficult for organizations to protect the data entrusted to them by their users while also respecting all relevant laws. The European Union has some of the most progressive privacy laws, and all the major cloud providers—Amazon, Google and Microsoft—offer multiple data centers and regions within the European Union. Therefore, we recommend that companies, especially those with a global user base, assess the feasibility of a safe haven for their users' data by **hosting PII data in the EU**. Since we wrote about this technique in the last Radar, we have rolled out a new internal system that handles sensitive information relating to all our employees, and we have chosen to host it in a data center located in the European Union.

ADOPT
1. Consumer-driven contract testing
2. Pipelines as code
3. Threat Modeling

TRIAL
4. APIs as a product
5. Bug bounties
6. Data Lake
7. Hosting PII data in the EU
8. Lightweight Architecture Decision Records
9. Reactive architectures
10. Serverless architecture

ASSESS
11. Client-directed query
12. Container security scanning
13. Content Security Policies
14. Differential privacy
15. Micro frontends
16. OWASP ASVS
17. Unikernels
18. VR beyond gaming

HOLD
19. A single CI instance for all teams
20. Anemic REST
21. Big Data envy
22. Cloud lift and shift

TECHNIQUES *continued*

Although much documentation can be replaced with highly readable code and tests, in a world of evolutionary architecture it's important to record certain design decisions for the benefit of future team members and for external oversight. **Lightweight Architecture Decision Records** is a technique for capturing important architectural decisions along with their context and consequences. Although these items are often stored in a wiki or collaboration tool, we generally prefer storing them in source control with simple markup.

Serverless architecture is an approach that replaces long-running virtual machines with ephemeral compute power that comes into existence on request and disappears immediately after use. Since the last Radar, we have had several teams put applications into production using a "serverless" style. Our teams like the approach; it's working well for them and we consider it a valid architectural choice. Note that serverless doesn't have to be an all-or-nothing approach: some of our teams have deployed a new chunk of their systems using serverless while sticking to a traditional architectural approach for other pieces.

Although many problems that people encounter with RESTful approaches to APIs can be attributed to the anemic REST antipattern, some use cases warrant exploration of other approaches. In particular, organizations that have to support a long tail of client applications (and thus a likely proliferation of API versions even if they employ consumer-driven contracts)—and have a large portion of their APIs supporting the endless-list style of activity feeds—may hit some limits in RESTful architectures. These can sometimes be mitigated by employing the **client-directed query** approach to client-server interaction. We see this approach being successfully used in both GraphQL and Falcor, where clients have more control over both the contents and the granularity of the data returned to them. This does put more responsibility onto the service layer and can still lead to tight coupling to the underlying data model, but the benefits may be worth exploring if well-modeled RESTful APIs aren't working for you.

The container revolution instigated by Docker has massively reduced the friction in moving applications between environments but at the same time has blown a rather large hole in the traditional controls over what can go to production. The technique of **container security scanning** is a necessary response to this threat vector. Docker now provides its own security scanning tools, as does CoreOS, and we've also had success with the CIS Security Benchmarks. Whichever approach you take, we believe the topic of automated container security validation is of high value and a necessary part of PaaS thinking.

It has long been known that "anonymized" bulk data sets can reveal information about individuals, especially when multiple data sets are cross-referenced together. With increasing concern over personal privacy, some companies—including Apple and Google—are turning to differential privacy techniques in order to improve individual privacy while retaining the ability to perform useful analytics on large numbers of users. **Differential privacy** is a cryptographic technique that attempts to maximize the accuracy of statistical queries from a database while minimizing the chances of identifying its records. These results can be achieved by introducing a low amount of "noise" to the data, but it's important to note that this is an ongoing research area. Apple has announced plans to incorporate differential privacy into its products—and we wholeheartedly applaud its commitment to customers' privacy—but the usual Apple secrecy has left some security experts scratching their heads. We continue to recommend Datensparsamkeit as an alternative approach: simply storing the minimum data you actually need will achieve better privacy results in most cases.

We've seen significant benefit from introducing microservice architectures, which have allowed teams to scale delivery of independently deployed and maintained services. However, teams have often struggled to avoid the creation of front-end monoliths—large and sprawling browser applications that are as difficult to maintain and evolve as the monolithic server-side applications we've abandoned. We're seeing an approach emerge that our teams call **micro frontends**. In this approach, a web application is broken up by its pages and features, with each feature being owned end-to-end by a single team. Multiple techniques exist to bring the application features—some old and some new—together as a cohesive user experience, but the goal remains to allow each feature to be developed, tested and deployed independently from others. The BFF - backend for frontends approach works well here, with each team developing a BFF to support its set of application features.

TECHNIQUES *continued*

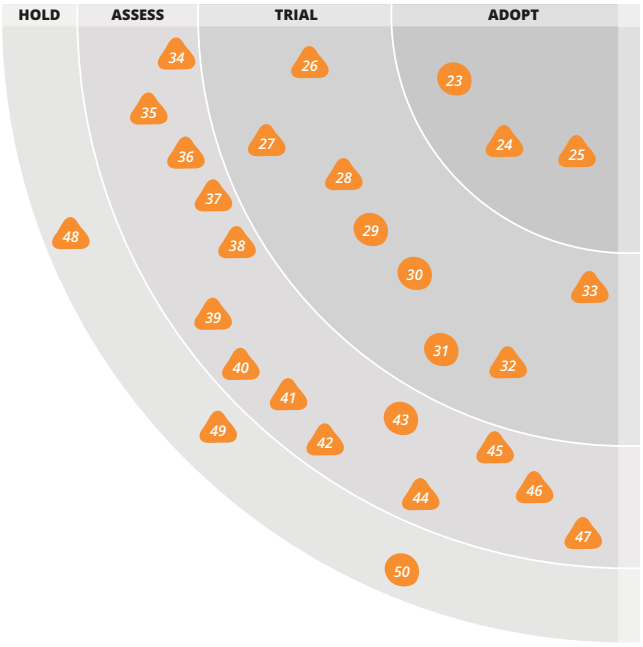
With the increasing popularity of the BFF - Backend for frontends pattern and use of one-way data-binding frameworks like React.js, we've noticed a backlash against REST-style architectures. Critics accuse REST of causing chatty, inefficient interactions among systems and failing to adapt as client needs evolve. They offer frameworks such as GraphQL or Falcor as alternative data-fetch mechanisms that let the client specify the format of the data returned. But in our experience, it isn't REST that causes these problems. Rather, they stem from a failure to properly model the domain as a set of resources. Naively developing services that simply expose static, hierarchical data models via templated URLs result in an **anemic REST** implementation. In a richly modeled domain, REST should enable more than simple repetitive data fetching. In a fully evolved RESTful architecture, business events and abstract concepts are also modeled as resources, and the implementation should make effective use of hypertext, link relations and media types to maximize decoupling between services. This antipattern is closely related to the Anemic Domain Model pattern and results in services that rank low in Richardson Maturity Model. We have more advice for designing effective REST APIs in our Insights article.

We continue to see organizations chasing "cool" technologies, taking on unnecessary complexity and risk when a simpler choice would be better. One particular theme is using distributed, Big Data systems for relatively small data sets. This behavior prompts us to put **Big Data envy** on hold once more, with some additional data points from our recent experience. The Apache Cassandra database promises massive scalability on commodity hardware, but we have seen teams overwhelmed by its architectural and operational complexity. Unless you have data volumes that require a 100+ node cluster, we recommend against using Cassandra. The operational team you'll need to keep the thing running just isn't worth it. While creating this edition of the Radar, we discussed several new database technologies, many offering "10x" performance improvements over existing systems. We're always skeptical until new technology—especially something as critical as a database—has been properly proven. Jepsen provides analysis of database performance under difficult conditions and has found numerous bugs in various NoSQL databases. We recommend maintaining a healthy dose of skepticism and keeping an eye on sites such as Jepsen when you evaluate database tech.

As more organizations are choosing to deploy applications in the cloud, we're regularly finding IT groups that are wastefully trying to replicate their existing data center management and security approaches in the cloud. This often comes in the form of firewalls, load balancers, network proxies, access control, security appliances and services that are extended into the cloud with minimal rethinking. We've seen organizations build their own orchestration APIs in front of the cloud providers to constrain the services that can be utilized by teams. In most cases these layers serve only to cripple the capability, taking away most of the intended benefits of moving to the cloud. In this edition of the Radar, we've chosen to rehighlight **cloud lift and shift** as a technique to avoid. Organizations should instead look more deeply at the intent of their existing security and operational controls, and look for alternative controls that work in the cloud without creating unnecessary constraints. Many of those controls will already exist for mature cloud providers, and teams that adopt the cloud can use native APIs for self-serve provisioning and operations.

PLATFORMS

HTTP Strict Transport Security (**HSTS**) is a now widely supported policy that allows websites to protect themselves from downgrade attacks. A downgrade attack in the context of HTTPS is one that can cause users of your site to fall back to HTTP rather than HTTPS, allowing for further attacks such as man-in-the-middle attacks. With HSTS, the server sends a header that informs the browser that it should only use HTTPS to access the website. Browser support is now widespread enough that this easy-to-implement feature should be added to any site using HTTPS. Mozilla's Observatory can help identify this and other useful headers and configuration options that improve security and privacy. When implementing HSTS, it is critical to verify that all resources load properly over HTTPS, because once HSTS is turned on, there is (almost) no turning back until the expiry time. The directive to include subdomains should be added but, again, a thorough verification that all subdomains support secure transport is required.



Application whitelisting has proven to be one of the most effective ways to mitigate cyber intrusion attacks. A convenient way to implement this widely recommended practice is through **Linux security modules**. With SELinux or AppArmor included by default in most Linux distributions, and with more comprehensive tools such as Grsecurity readily available, we have moved this technology into the Adopt ring in this edition. These tools help teams assess questions about who has access to what resources on shared hosts, including contained services. This conservative approach to access management will help teams build security into their SDLC processes.

We've continued to have positive experiences deploying the **Apache Mesos** platform to manage cluster resources for highly distributed systems. Mesos abstracts out underlying computing resources such as CPU and storage, aiming to provide efficient utilization while maintaining isolation. Mesos includes Chronos for distributed and fault-tolerant execution of scheduled jobs, and Marathon for orchestrating long-running processes in containers.

We have a growing belief that for most scenarios it is rarely worth rolling your own authentication code. Outsourced identity management speeds up delivery, reduces mistakes and tends to enable a faster response to newly discovered vulnerabilities. **Auth0** has particularly impressed us in this field for its ease of integration, range of protocols and connectors supported, and rich management API.

ADOPT
23. Docker
24. HSTS
25. Linux security modules

TRIAL
26. Apache Mesos
27. Auth0
28. AWS Lambda
29. Kubernetes
30. Pivotal Cloud Foundry
31. Rancher
32. Realm
33. Unity beyond gaming

ASSESS
34. .NET Core
35. Amazon API Gateway
36. Apache Flink
37. AWS Application Load Balancer
38. Cassandra carefully
39. Electron
40. Ethereum
41. HoloLens
42. IndiaStack
43. Nomad
44. Nuance Mix
45. OpenVR
46. Tarantool
47. wit.ai

HOLD
48. CMS as a platform
49. Overambitious API gateway
50. Superficial private cloud

PLATFORMS *continued*

Our teams continue to enjoy using **AWS Lambda** and are beginning to use it to experiment with serverless architectures, combining Lambda with the API Gateway. We do recommend that Lambda functions contain only a moderate amount of code. Ensuring the quality of a solution based on a tangle of many large Lambda functions is difficult, and such a solution may not be cost-effective. For more complex needs, deployments based on containers or VMs are still preferable. In addition, we have run into significant problems using Java for Lambda functions, with erratic latencies up to several seconds as the Lambda container is started. Of course, you can sidestep this issue by using JavaScript or Python, and if Lambda functions do not contain a lot of code, the choice of programming language should not matter too much.

Realm is a database designed for use on mobile devices, with its own persistence engine to achieve high performance. Realm is marketed as a replacement for SQLite and Core Data. Note that migrations are not quite as straightforward as the Realm documentation would have you believe. However, more and more teams are choosing Realm as the persistence mechanism in production environments for mobile applications.

After experiencing years of growth as a platform for game development, **Unity** has recently become the platform of choice for VR and AR application development. Whether you're creating a fully immersive world for the Oculus or HTC Vive headsets, a holographic layer for your newly spatial enterprise application or an AR feature set for your mobile app, Unity likely provides what you need to both prototype it and get it ready for prime time. Many of us at ThoughtWorks believe that VR and AR represent the next significant shift in the computing platform, and for now, Unity is the single most important tool in the toolbox we use to develop for this change. We've used Unity to develop all our VR prototypes, as well as AR functionality for headsets and phone/tablet applications.

.NET Core is an open source modular product for creating applications that can be easily deployed in Windows, macOS and Linux. .NET Core makes it possible to build cross-platform web applications using ASP.NET Core with a set of tools, libraries and frameworks—another choice for microservices architecture. The community around .NET Core and other related projects has been growing. New tools have appeared and evolved quickly, such as Visual Studio Code. There are Docker images based on both Linux and Windows (Nano Server) with .NET Core that simplify applying a microservice architecture. CoreCLR and CoreFX appeared in the Radar in the past. However, a few months ago Microsoft announced the release of .NET Core 1.0, the first stable version. We see good new opportunities, changes and a vibrant community as reasons to keep assessing this product.

Amazon API Gateway is Amazon's offering enabling developers to expose API services to Internet clients. It offers the usual API gateway features like traffic management, monitoring, authentication and authorization. Our teams have been using this service to front other AWS capabilities like AWS Lambda as part of serverless architectures. We continue to monitor for the challenges presented by overambitious API gateways, but at this stage Amazon's offering appears to be lightweight enough to avoid those problems.

Interest continues to build for **Apache Flink**, a new-generation platform for scalable distributed batch and stream processing. At the core of Apache Flink is a streaming data-flow engine, with support for tabular (SQL-like), graph-processing and machine learning operations. Apache Flink stands out with feature rich capabilities for stream processing: event time, rich streaming window operations, fault tolerance and exactly-once semantics. The project shows significant ongoing activity, with the latest release (1.1) introducing new datasource/sink integrations as well as improved streaming features.

Amazon recently launched the **AWS Application Load Balancer** (ALB), a direct replacement for Elastic Load Balancers introduced back in 2009. ALB supports Layer 7 traffic inspection and is built to support modern cloud architecture. If you're building a microservices-based system using ECS, the new load balancers will directly understand container hosting and scaling, with multiple containers and ports per EC2 instance. Content-based routing allows segmentation of requests onto groups of target servers, along with independent scaling of those groups. Health checks performed by the load balancers are much improved, with the ability to capture detailed metrics about application performance. We like everything that we see here, and teams have begun to report successful usage of ALB.

PLATFORMS *continued*

Apache's Cassandra database is a powerful, scalable Big Data solution for storing and processing large amounts of data, often using hundreds of nodes split over multiple worldwide locations. It's a great tool and we like it, but too often we see teams run into trouble using it. We recommend using **Cassandra carefully**. Teams often misunderstand the use case for Cassandra, attempting to use it as a general-purpose data store when in fact it is optimized for fast reads on large data sets based on predefined keys or indexes. Its dependence on the storage schema can also make it difficult to evolve over time. Cassandra also has significant operational complexity and some rough edges, so unless you absolutely need the scaling it provides, a simpler solution is usually better. If you don't need Cassandra's specific use-case and scaling characteristics, you might just be choosing it out of Big Data envy. Careful use of Cassandra will include extensive automated testing, and we're happy to recommend CassandraUnit as part of your testing strategy.

Electron is a solid framework for building native desktop clients using web technologies such as HTML, CSS and JavaScript. Teams can leverage their web know-how to deliver polished cross-platform desktop clients without spending time learning another set of technologies.

The hype seems to have peaked for blockchain and cryptocurrencies, as evidenced by the previous firehose-scale announcements in this area slowing to a trickle, and we expect some of the more speculative efforts to die out over time. One of the blockchains, **Ethereum**, is making good progress and is worth watching. Ethereum is a public blockchain with a built-in programming language that allows "smart contracts" to be built into it. These are algorithmic movements of "ether" (the Ethereum cryptocurrency) in response to activity happening on the blockchain. R3Cev, the consortium building blockchain tech for banks, built its first proof of concept on Ethereum. Ethereum has been used to build a Distributed Autonomous Organization (DAO)—one of the first "algorithmic corporations"—although a recent heist of $150m worth of Ether demonstrates that the blockchain and cryptocurrencies are still the Wild West of the technology world.

In the **HoloLens**, Microsoft has delivered the first truly usable AR headset. Not only is it a beautiful piece of industrial design and an eminently comfortable device to wear, but it also clearly demonstrates the promise of AR for the enterprise via its gorgeous optics and deep Windows 10 integration. We expect HoloLens to be the first AR platform on which we deliver substantial application functionality to our clients in the near term, and we look forward to its evolution as it gains broader traction.

IndiaStack is a set of Open APIs designed with the goal of transforming India from a data-poor to a data-rich country. The stack emphasizes layered innovation by specifying a minimal set of APIs and encourages the rest of the ecosystem to build custom applications on top of these APIs. Aadhaar serves as one of the foundation layers, providing authentication services for more than a billion Indian citizens. In addition, there are services to provide paperless transactions through digital signatures (eSign), unified online payment (UPI) and an electronic consent layer (e-KYC) to securely provide Aadhaar details to service providers. We believe in the Open API–driven initiative to bring digital innovation, and the design principles behind IndiaStack could be used as a change agent for other regions/countries.

Nuance Mix is a framework for natural language processing from the company that created the speech-to-text technology behind Dragon Speaking and the first roll-out of Siri. This framework supports the creation of grammars that allow for free-form user interaction via voice. The developer defines a domain-specific grammar that the framework can train itself to understand. The outcomes are responses to user input that identify the user's intents and interaction concepts. At first, it is limited to phrases close to the ones used to train it, but over time it can start to identify meaning from more divergent phrasing. Though it is still in beta, the accuracy from early exploration has been compelling, and the eventual product is one to watch for application forms that could benefit from hands-free user interaction—including mobile, IoT, AR, VR and interactive spaces.

OpenVR is the underlying SDK in making many of the VR head-mounted displays (HMDs) work with Unity and will likely keep growing in importance. Much of the VR work at ThoughtWorks was built on top of OpenVR, because it will run on any HMD, unlike the other SDKs. Though it is not open source, it is free via the license. The Oculus SDK is more restrictive in its licensing and only works on Oculus devices. OSVR, while truly open source, doesn't seem to have as much adoption yet. If you're going to develop a VR application and target as many devices as possible—and not use Unity or Unreal to develop them—OpenVR is the most concrete and pragmatic solution right now.

PLATFORMS *continued*

Tarantool is an open source NoSQL solution that combines database and cache into one entity and provides APIs for writing application logic in Lua. Both in-memory and disk-based engines are supported, and users can create multiple indexes (HASH, TREE, RTREE, BITSET) based on their use cases. The data itself is stored in MessagePack format and uses the same protocol to communicate between clients and server. Tarantool supports write-ahead logs, transactions and asynchronous master-master replication. We are happy with the architectural decision of embracing single-writer policy and cooperative multitasking to handle concurrent connections.

Hype surrounding machine intelligence has reached a crescendo, but as with Big Data, useful frameworks and tools are waiting to be discovered among all the hot air. One such tool is **wit.ai**, a SaaS platform that allows developers to create conversational interfaces using natural language processing (NLP). Wit works with either text or speech inputs, helps developers manage conversational intent and allows custom business logic to be implemented using JavaScript. The system is free for commercial and noncommercial use and encourages the creation of open applications. Be aware that you must agree to let Wit use your data in order to improve the service and for its own analysis, so read the terms and conditions carefully. Another contender in this space is the Microsoft Bot Framework, but it's available only in limited preview form as of this writing. As with most things Microsoft, we expect the Bot Framework to evolve quickly, so it's worth keeping an eye on.

We are seeing too many organizations run into trouble as they attempt to use their **CMS as a platform** for delivering large and complex digital applications. This is often driven by the vendor-fueled hope of bypassing unresponsive IT organizations and enabling the business to drag and drop changes directly to production. While we are very supportive of providing content producers with the right tools and workflows, for applications with complex business logic we tend to recommend treating your CMS as a component of your platform (often in a hybrid or headless mode) cooperating cleanly with other services, rather than attempting to implement all of your functionality in the CMS itself.

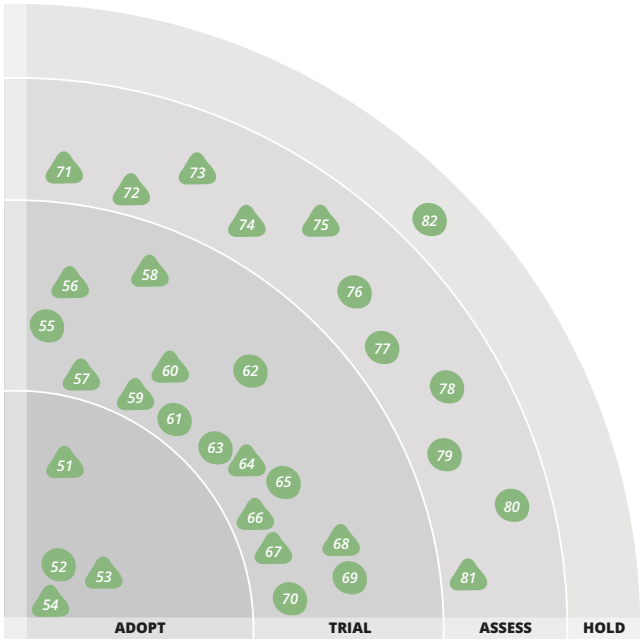
One of our regular complaints is about business smarts implemented in middleware, resulting in transport software with ambitions to run critical application logic. Vendors in the highly competitive API gateway market continue to add features that differentiate their products. This results in **overambitious API gateway** products whose functionality—on top of what is essentially a reverse proxy—encourages designs that are difficult to test and deploy. API gateways can provide utility in dealing with some generic concerns—for example, authentication and rate-limiting—but any domain smarts such as data transformation or rule processing should live in applications or services where they can be controlled by product teams working closely with the domains they support.

TOOLS

Babel.js has become the default compiler for writing next-generation JavaScript. Its ecosystem is really taking off, thanks to its restructured plugin system. It allows developers to write ES6 (and even ES7) code that runs in the browser or in the server without sacrificing backward compatibility for older browsers, and with very little configuration. It has first-class support for different build-and-test systems, which makes integration with any current workflow simple. It is a great piece of software that has become the main driver of ES6 (and ES7) adoption and innovation.

When combining modern techniques and architecture styles, such as microservices, DevOps and QA in production, development teams need increasingly sophisticated monitoring. Simply looking a graphs of disk usage and CPU utilization is not sufficient anymore, and many teams collect application and business-specific metrics using tools such a Graphite and Kibana. **Grafana** makes it easy to create useful and elegant dashboards for data from a number of sources. A particularly useful feature allows timescales of different graphs to be synchronized, which helps with spotting correlations in the underlying data. The templating system that is being added shows a lot promise and will likely make managing sets of similar services even easier. Based on its strengths, Grafana has become our default choice in this category.

Machine images have become a staple of modern deployment pipelines, and there are a number of tools and techniques to create the images. Because of its comprehensive feature set and the positive experiences we've had with it, we recommend **Packer** over the alternatives. We also recommend against trying to write custom scripts to do what Packer does out of the box.



At the top of the testing pyramid for Android application development, our teams are increasingly using **Espresso** as the functional-testing tool. Its small-core API hides the messy implementation details and helps in writing concise tests, with faster and reliable test execution. With Espresso, you can run automated UI tests simulating user interactions within a single target app on both emulators and real devices across different Android versions.

fastlane is our go-to tool for automating most of the boring activities involved in getting iOS and Android mobile apps built, tested, documented and provisioned. Simple configuration, a range of tooling and multiple pipelines make this a key ingredient in doing continuous delivery for mobile.

ADOPT
51. Babel
52. Consul
53. Grafana
54. Packer

TRIAL
55. Apache Kafka
56. Espresso
57. fastlane
58. Galen
59. HashiCorp Vault
60. JSONassert
61. Let's Encrypt
62. Load Impact
63. OWASP Dependency-Check
64. Pa11y
65. Serverspec
66. Talisman
67. Terraform
68. tmate
69. Webpack
70. Zipkin

ASSESS
71. Android-x86
72. axios
73. Bottled Water
74. Clojure.spec
75. FBSnapshotTestcase
76. Grasp
77. LambdaCD
78. Pinpoint
79. Pitest
80. Repsheet
81. Scikit-learn

HOLD
82. Jenkins as a deployment pipeline

TOOLS *continued*

Testing that layout and styling of responsive websites is working as expected across various form factors can be a slow and often manual process. **Galen** helps ease this problem by providing a simple language, running on top of Selenium, that allows you to specify expectations for the appearance of your website in various screen sizes. Although Galen suffers from the typical brittleness and speed issues of any end-to-end testing approach, we have found benefit in the early feedback on design issues.

Having a way to securely manage secrets is increasingly becoming a huge project issue. The old practice of keeping secrets in a file or in environment variables is becoming hard to manage, especially in environments with multiple applications and large numbers of microservices. **HashiCorp Vault** addresses the problem by providing mechanisms for securely accessing secrets through a unified interface. It has served us well on a number of projects, and our teams liked how easy it was to integrate Vault with their services. Storing and updating secrets is a bit cumbersome, because it relies on a command-line tool and a fair amount of discipline from the team.

More projects are emitting and consuming information formatted as JSON. Writing tests in Java for JSON can be laborious. **JSONassert** is a small library to help write smaller tests dealing with JSON by simplifying assertions and providing better error messages.

Pa11y is an automatic accessibility tester that can run from the command line and be embedded into a build pipeline. Our teams have had success using Pa11y on a highly dynamic site by first creating a static HTML version, then running the accessibility tests against that. For many systems—especially government websites—accessibility testing is a requirement, and Pa11y makes it all a lot easier.

With the maturity of tools such as Vault, there is no longer an excuse for storing secrets in code repositories, particularly since this often ends up being the soft underbelly of important systems. We've previously mentioned repository-scanning tools such as Gitrob, but we are now pushing proactive tools such as (the ThoughtWorks-created) **Talisman**, which is a prepush hook for Git that scans commits for secrets matching predefined patterns.

With **Terraform**, you can manage cloud infrastructure by writing declarative definitions. The configuration of the servers instantiated by Terraform is usually left to tools like Puppet, Chef or Ansible. We like Terraform because the syntax of its files is quite readable and because it supports a number of cloud providers while making no attempt to provide an artificial abstraction across those providers. Following our first, more cautious, mention of Terraform almost two years ago, it has seen continued development and has evolved into a stable product that has proven its value in our projects. The issue with state file management can now be sidestepped by using what Terraform calls a "remote state backend." We've successfully used Consul for that purpose.

Pair programming is an essential technique for us, and—given that we're seeing more and more teams whose members are distributed across multiple locations—we have experimented with a number of tools to support remote pairing. We certainly liked ScreenHero but are concerned about its future. For teams that don't rely on a graphical IDE, using **tmate** for pairing has turned out to be a great solution. tmate is a fork of the popular tmux tool, and compared to tmux for remote pairing, the setup is much easier. Compared to graphical screen-sharing solutions, the bandwidth and resource requirements are modest, and it obviously never suffers from blurry screens. Teams can also set up their own server, thus retaining full control of the privacy and integrity of the solution.

Android-x86 is a port of the Android open source project to x86 platforms. The project started by hosting various patches from the community for x86 support but then later created its own codebase to provide support for different x86 platforms. We have seen significant time savings by utilizing Android-x86 in our CI servers instead of emulators for hermetic UI testing. However, for UI-specific tests targeting a particular device resolution—simulating low memory, bandwidth and battery—it is better to stick with emulators.

Our teams have had success with **axios**, a promises-based HTTP client in JavaScript that they describe as "better than Fetch." The project has lots of endorsements and activity on GitHub, and it gets a thumbs-up from us.

TOOLS *continued*

With the growth of interest in streaming data architectures and the downstream data lakes they feed, we have seen an increased reliance on "change data capture" tooling to connect transactional data stores to stream-processing systems. **Bottled Water** is a welcome addition to this field, converting changes in PostgreSQL's write-ahead log into Kafka events. One downside of this approach, however, is that you are tied to low-level database events rather than the higher-level business events we recommend as the foundation for an event-oriented architecture.

One of those perpetual developer debates involves language typing: How much is just right? Clojure, the dynamically typed functional Lisp on the JVM, added a new entry into this discussion that blurs the lines. **Clojure.spec** is a new facility built into Clojure that allows developers to wrap type and other verification criteria around data structures, such as allowable value ranges. Once they are established, Clojure uses these specifications to provide a slew of benefits: generated tests, validation, destructuring of data structures and others. Clojure.spec is a promising way to have the benefits of types and ranges where developers need them but not everywhere.

Testing the visual portion of iOS applications can be painful, slow and flakey, which is why we're happy to include **FBSnapshotTestcase** in our toolkit. It automates taking, storing and diff-ing snapshots of UI components so you can keep your interfaces pixel-perfect. Since it runs as a unit test (in the simulator), it is faster and more reliable than functional-testing approaches.

Scikit-learn is an increasingly popular machine-learning library written in Python. It provides a robust set of machine-learning models such as clustering, classification, regression and dimensionality reduction, and a rich set of functionality for companion tasks like model selection, model evaluation and data preparation. Since it is designed to be simple, reusable in various contexts and well documented, we see this tool accessible even to nonexperts to explore the machine-learning space.

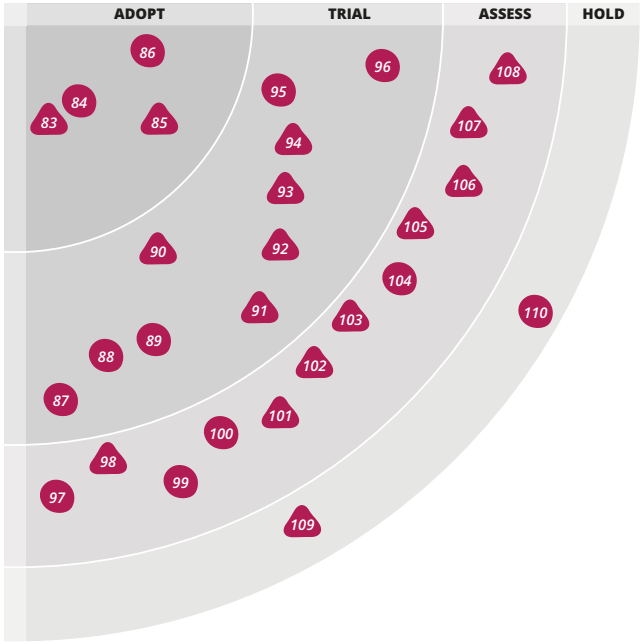
LANGUAGES & FRAMEWORKS

If you are faced with building a single-page application (SPA) and trying to choose a framework to build with, **Ember.js** has emerged as a leading choice. Our teams praise Ember for its highly productive developer experience, with far fewer surprises than other frameworks such as AngularJS. The Ember CLI build tooling is a haven in the storm of JavaScript build tools, and the Ember core team and community are highly active and responsive.

With the increasing complexity of single-page JavaScript applications, we have seen a more pressing need to make client-side state management predictable. **Redux**, with its three principles of restrictions for updating state, has proven to be invaluable in a number of projects we have implemented. Getting Started with Redux and idiomatic Redux tutorials are a good starting point for new and experienced users. Its minimal library design has spawned a rich set of tools, and we encourage you to check out the redux-ecosystem-links project for examples, middleware and utility libraries. We also particularly like the testability story: Dispatching actions, state transitions and rendering can be unit-tested separately from one another and with minimal amounts of mocking.

Interest in the **Elixir** programming language continues to build. Increasingly, we see it used in serious projects and hear feedback from developers who find its Actor model to be robust and very fast. Elixir, which is built on top of the Erlang virtual machine, is showing promise for creating highly concurrent and fault-tolerant systems. Elixir has distinctive features such as the Pipe operator, which allows developers to build a pipeline of functions as you would in the UNIX command shell. The shared byte code allows Elixir to interoperate with Erlang and leverage existing libraries while supporting tools such as the Mix build tool, the IEx interactive shell and the ExUnit unit-testing framework.

We've been enjoying the rapid component-level UI testing that **Enzyme** provides for React.js applications. Unlike many other snapshot-based testing frameworks,



Enzyme allows 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.

Immutability is often emphasized in the functional programming paradigm, and most languages have the ability to create immutable objects—objects that can't be changed once created. **Immutable.js** is a library for JavaScript that provides many persistent immutable data structures, which are highly efficient on modern JavaScript virtual machines. Immutable.js objects are, however, not normal JavaScript objects, so references to JavaScript objects from immutable objects should be avoided. More teams are using this library for tracking mutation and maintaining state in production. We recommend that developers investigate this library, especially when it's combined with the rest of the Facebook stack.

ADOPT
83. Ember.js
84. React.js
85. Redux
86. Spring Boot

TRIAL
87. Butterknife
88. Dagger
89. Dapper
90. Elixir
91. Enzyme
92. Immutable.js
93. Phoenix
94. Quick and Nimble
95. React Native
96. Robolectric

ASSESS
97. Aurelia
98. ECMAScript 2017
99. Elm
100. GraphQL
101. JuMP
102. Physical Web
103. Rapidoid
104. Recharts
105. ReSwift
106. Three.js
107. Vue.js
108. WebRTC

HOLD
109. AngularJS
110. JSPatch

LANGUAGES & FRAMEWORKS *continued*

Some of our ThoughtWorks teams have had very positive experiences with **Phoenix**, a server-side web MVC framework written in Elixir. In addition to being streamlined and easy to use, Phoenix takes advantage of Elixir to be extremely fast. For some developers, Phoenix evokes the joy they experienced when first discovering Ruby and Rails. Although the ecosystem of libraries for Phoenix is not as extensive as for some more mature frameworks, it should benefit from the continuing success and growth of support for Elixir.

Most of our iOS teams are now using the **Quick and Nimble** pairing for their unit tests. In the RSpec family of behavior-driven development (BDD) testing tools, it provides very readable tests (with describe blocks) across Swift and Objective-C and has good support for asynchronous testing.

ECMAScript 2017—not to be confused with ES7 (a.k.a. ECMAScript 2016)—brings several noteworthy improvements to the language. Browsers are expected to implement this standard fully in the summer of 2017, but the Babel JavaScript compiler already supports a number of the features today. If you make extensive use of JavaScript and your codebase is under active development, we recommend that you add Babel to your build pipeline and begin using the supported features.

JuMP is a domain-specific language for mathematical optimizations in Julia. JuMP defines a common API called MathProgBase and enables users to write solver-agnostic code in Julia. Currently supported solvers include Artelys Knitro, Bonmin, Cbc, Clp, Couenne, CPLEX, ECOS, FICO Xpress, GLPK, Gurobi, Ipopt, MOSEK, NLopt and SCS. One other benefit is the implementation of automatic differentiation technique in reverse mode to compute derivatives so users are not limited to the standard operators like sin, cos, log and sqrt but can also implement their own custom objective functions in Julia.

We have been intrigued by the **Physical Web** standard created by Google. The idea of Physical Web is simple—beacons broadcast a URL—but the possibilities are broad. Basically, this is a way to annotate the physical world, tying objects and locations into the digital realm. The current transport mechanism is Eddystone URLs over Bluetooth LE, and sample clients are available. Although there are obvious security concerns with following randomly discovered links, we are most interested in use cases with customized clients where you can filter or proxy the URLs as required.

Rapidoid is a collection of web framework modules, including a fast low-level HTTP server implemented from scratch on top of Java NIO. Clever usage of off-heap input/output buffers, object pools and thread-local data structures provide Rapidoid an edge over other NIO-based servers like Netty. Being a fairly new project, Rapidoid has yet to implement a few features like built-in cache and SSL support; we suggest you check the roadmap for updates.

We are excited that the Redux paradigm has made its way to Swift-land in the form of **ReSwift**. We've found real benefits in the simplicity and readability of codebases once state and state changes are managed in a central place and common idiom. This also helps with building "offline first" applications.

Despite the fervor surrounding the spate of new headsets, we believe there are many VR and AR scenarios that make sense in the browser, particularly on mobile. Given this trend, we have seen an uptick in usage of **Three.js**, a powerful JavaScript visualization and 3D rendering framework. The growth in support for WebGL, which it is based on, has helped adoption, as has the vibrant community supporting this open source project.

In the ever-changing world of front-end JavaScript frameworks, **Vue.js** has gained a lot of ground as a lightweight alternative to AngularJS. It is designed to be a very flexible—and a less opinionated—library that offers a set of tools for building interactive web interfaces around concepts like modularity, components and reactive data flow. It has a low learning barrier, which makes it interesting for junior developers and beginners. Vue.js itself is not a full-blown framework; it is focused on the view layer only and therefore is easy to integrate with other libraries or existing projects.

Widespread adoption of AR/VR as a collaboration and communication medium requires a modern and readily available video streaming platform. **WebRTC** is an emerging standard for real-time communication between browsers that enables video streaming within commonly available web technologies. The range of browsers that support this standard is increasing, but Microsoft and Apple have been slow to adopt WebRTC in their proprietary browsers. If momentum continues to build, WebRTC could form the future foundation for AR/VR collaboration on the web.

AngularJS helped revolutionize the world of single-page JavaScript applications, and we have delivered many projects successfully with it over the years. However, we are no longer recommending it (v1) for teams starting fresh projects. We prefer the ramp-up speed and more maintainable codebases we are seeing with Ember and React, particularly in conjunction with Redux.

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 4000 people, including a products division which makes pioneering tools for software teams. ThoughtWorks has 40 offices across 14 countries: Australia, Brazil, Chile, China, Ecuador, Germany, India, Italy, Singapore, South Africa, Spain, Turkey, the United Kingdom and the United States.

ThoughtWorks®