ThoughtWorks®

TECHNOLOGY RADAR *APRIL '16*

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

thoughtworks.com/radar

WHAT'S NEW?

Here are the themes highlighted in this edition:

OPEN SOURCE AS A VIRTUOUS BY-PRODUCT

Some of the most influential software appearing on our radar comes from companies whose first mandate isn't to create software tools. Several of our radar entries come from Facebook, not considered a traditional software development toolmaker. Unlike in the past, today many companies open source their important software assets—to attract new recruits and credentialize themselves. This creates a virtuous feedback loop: Innovative open source attracts good developers who are in turn more likely to innovate. As a side effect, these companies' frameworks and libraries are some of the most influential in the industry. This represents a big shift in the software development ecosystem and is further proof of the efficacy of open source software ... in the right context (our advice about Web Scale Envy still stands).

PARSING THE PAAS PUZZLE

Many large organizations see the Cloud and Platform as a Service (PaaS) as an obvious way to standardize infrastructure, ease deployment and operations, and make developers more productive. But it's still early days, the definition of PaaS remains nebulous, and many PaaS approaches are incomplete or suffer from the immaturity of supporting frameworks and tools. Some PaaS solutions make it harder to do things more easily done with plain Infrastructure as a Service (IaaS), such as using a custom Service Locator or complex network topology, and the jury is still out on whether a "Containers as a Service" approach will provide similar value with more flexibility. We see many companies implementing an off-the-shelf PaaS or gradually building their own, with varying degrees of success. We suspect that any PaaS built today will not be an end state but rather part of an evolutionary path. Enterprise migration to Cloud and PaaS, while bringing many benefits, has difficulties and challenges, particularly around overall pipeline design and tooling. Consumers of these technologies should seek the inflection point that indicates "ready for prime time" for their context and should avoid coupling too tightly to the implementation details of their PaaS.

DOCKER, DOCKER, DOCKER!

Containerization, and Docker in particular, has proven hugely beneficial as an application-management technique, rationalizing deployment between environments and simplifying the "it works here but not there" class of problems. We see a significant amount of energy focused on using Docker—and, particularly, the ecosystem surrounding it—beyond dev/test and all the way into production. Docker containers are used as the "unit of scaling" for many PaaS and "data center OS" platforms, giving Docker even more momentum. As it matures as both a development and production environment, people are paying more attention to containerization, its side effects and its implications.

OVER-REACTIVE?

Reactive programming—where components react to changes in data that are propagated to them rather than use imperative wiring—has become extremely popular, with reactive extensions available in almost all programming languages. User interfaces, in particular, are commonly written in a reactive style, and many ecosystems are settling on this paradigm. While we like the pattern, overuse of event-based systems complicates program logic, making it difficult to understand; developers should use this style of programming judiciously. It is certainly popular: We added a significant number of reactive frameworks and supporting tools on this Radar.

CONTRIBUTORS

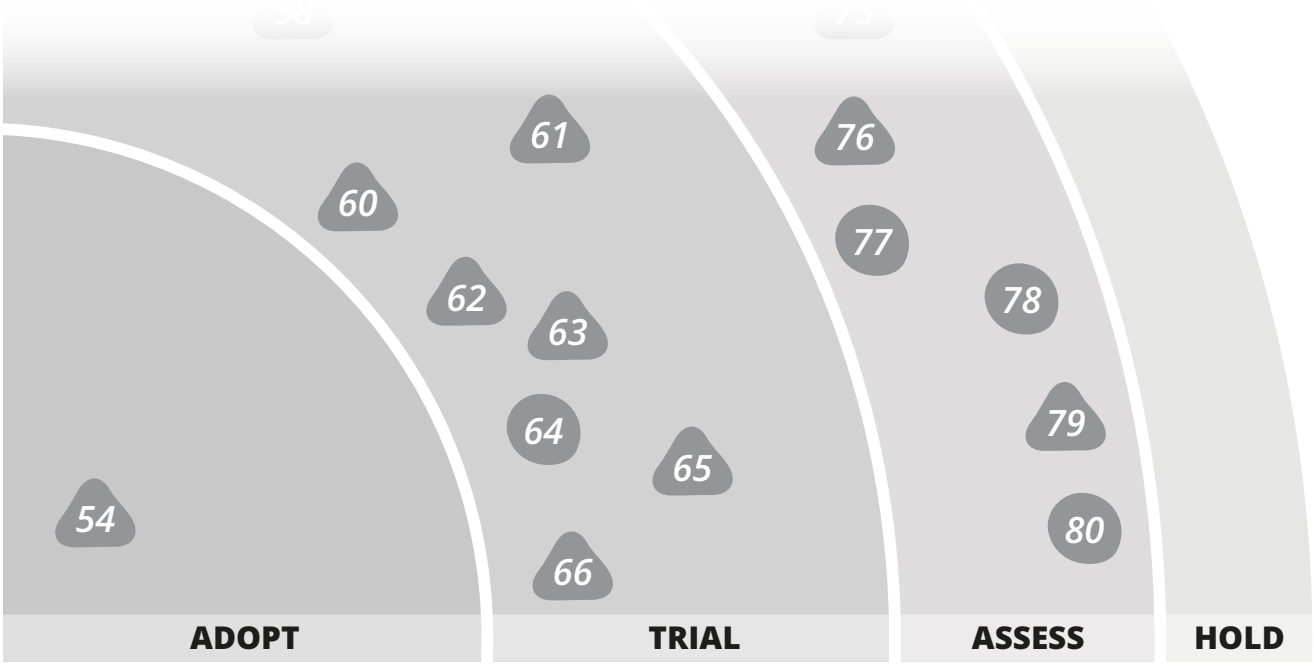
The Technology Radar is prepared by the ThoughtWorks Technology Advisory Board, comprised of:

Rebecca Parsons (CTO)	Dave Elliman	Ian Cartwright	Rachel Laycock
Martin Fowler(Chief Scientist)	Erik Doernenburg	James Lewis	Sam Newman
Anne J Simmons	Evan Bottcher	Jonny LeRoy	Scott Shaw
Badri Janakiraman	Fausto de la Torre	Mike Mason	Srihari Srinivasan
Brain Leke	Hao Xu	Neal Ford	Thiyagu Palanisamy

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. Decoupling deployment from release
2. Products over projects
3. Threat Modeling

TRIAL
4. BFF - Backend for frontends
5. Bug bounties
6. Data Lake
7. Event Storming
8. Flux
9. Idempotency filter
10. iFrames for sandboxing
11. NPM for all the things
12. Phoenix Environments
13. QA in production
14. Reactive architectures

ASSESS
15. Content Security Policies new
16. Hosted IDE's
17. Hosting PII data in the EU new
18. Monitoring of invariants
19. OWASP ASVS new
20. Serverless architecture new
21. Unikernels new
22. VR beyond gaming new

HOLD
23. A single CI instance for all teams new
24. Big Data envy new
25. Gitflow
26. High performance envy/web scale envy
27. SAFe™

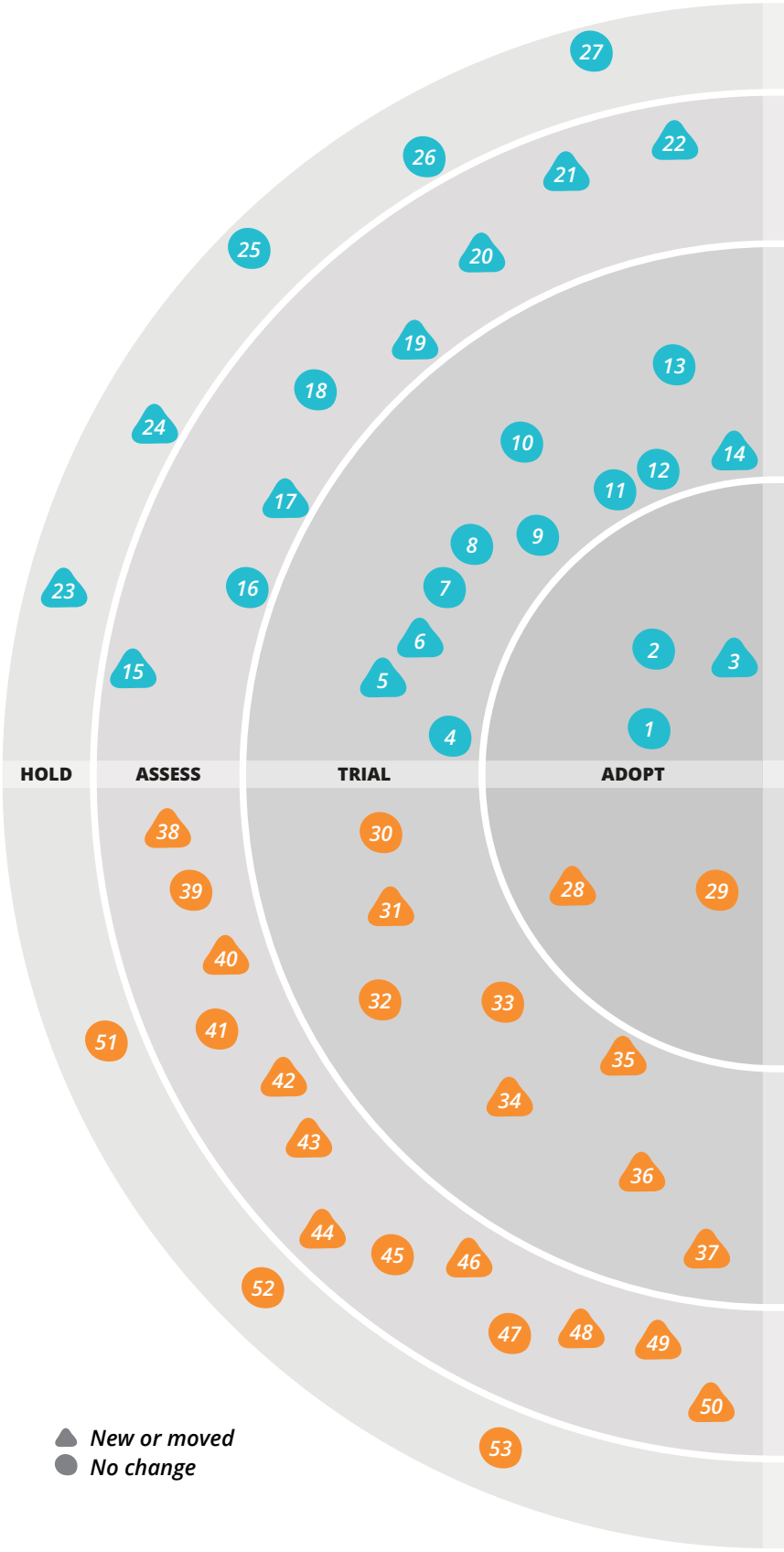
PLATFORMS

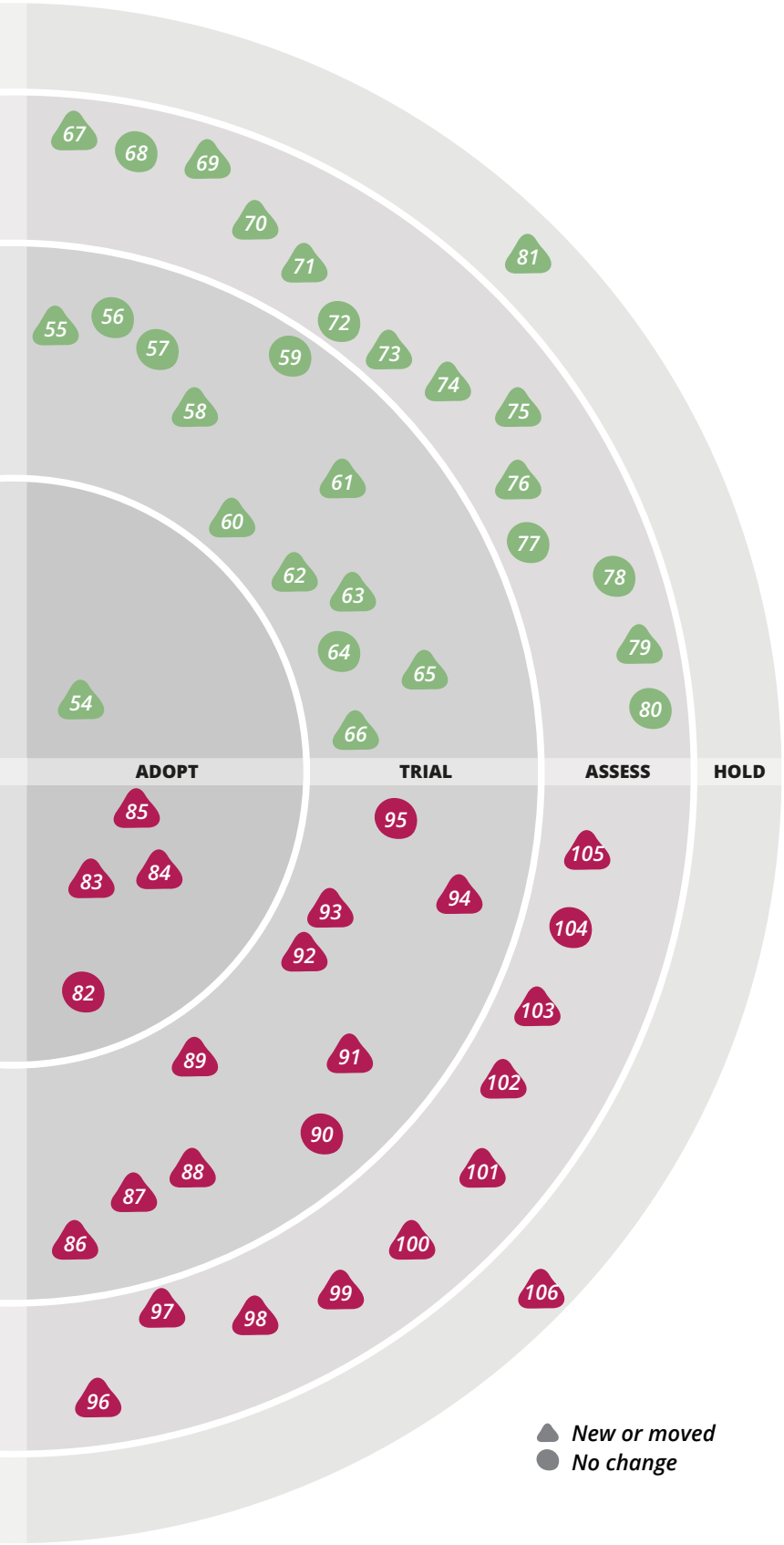
ADOPT
28. Docker
29. TOTP Two-Factor Authentication

TRIAL
30. Apache Mesos
31. AWS Lambda
32. H2O
33. HSTS
34. Kubernetes
35. Linux security modules
36. Pivotal Cloud Foundry new
37. Rancher

ASSESS
38. Amazon API Gateway new
39. AWS ECS
40. Bluetooth Mesh new
41. Ceph
42. Deflect new
43. ESP8266 new
44. MemSQL new
45. Mesosphere DCOS
46. Nomad new
47. Presto
48. Realm new
49. Sandstorm new
50. TensorFlow new

HOLD
51. Application Servers
52. Over-ambitious API Gateways
53. Superficial private cloud





THE RADAR

TOOLS

ADOPT
54. Consul

TRIAL
55. Apache Kafka
56. Browsersync
57. Carthage
58. Gauge
59. GitUp
60. Let's Encrypt
61. Load Impact new
62. OWASP Dependency-Check new
63. Serverspec new
64. SysDig
65. Webpack new
66. Zipkin

ASSESS
67. Apache Flink new
68. Concourse CI
69. Gitrob
70. Grasp new
71. HashiCorp Vault new
72. ievms
73. Jepsen new
74. LambdaCD new
75. Pinpoint new
76. Pitest new
77. Prometheus
78. RAML
79. Repsheet new
80. Sleepy Puppy

HOLD
81. Jenkins as a deployment pipeline new

LANGUAGES & FRAMEWORKS

ADOPT
82. ES6
83. React.js
84. Spring Boot
85. Swift

TRIAL
86. Butterknife new
87. Dagger new
88. Dapper new
89. Ember.js
90. Enlive
91. Fetch new
92. React Native
93. Redux new
94. Robolectric new
95. SignalR

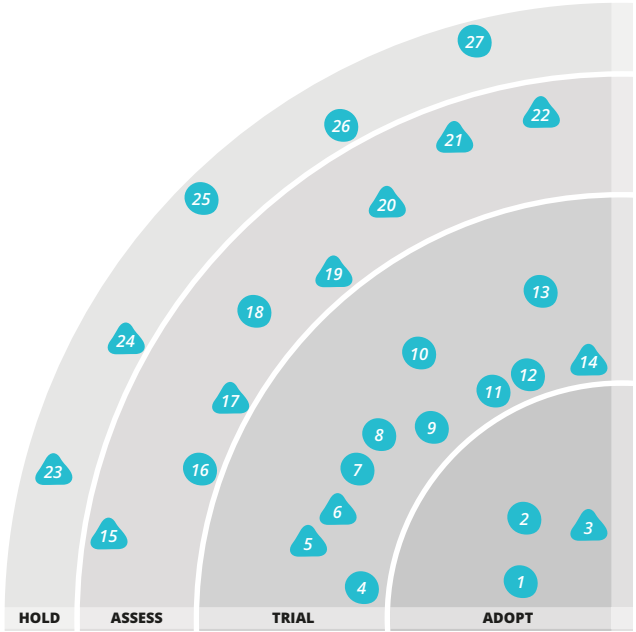
ASSESS
96. Alamofire new
97. AngularJS
98. Aurelia new
99. Cylon.js new
100. Elixir
101. Elm
102. GraphQL new
103. Immutable.js new
104. OkHttp
105. Recharts new

HOLD
106. JSPatch new

TECHNIQUES

With the number of high-profile security breaches in the past months, software development teams no longer need convincing that they must place an emphasis on writing secure software and dealing with their users' data in a responsible way. The teams face a steep learning curve, though, and the vast number of potential threats—ranging from organized crime and government spying to teenagers who attack systems "for the lulz"—can be overwhelming. **Threat Modeling** provides a set of techniques that help you identify and classify potential threats early in the development process. It is important to understand that it is only part of a strategy to stay ahead of threats. When used in conjunction with techniques such as establishing cross-functional security requirements to address common risks in the technologies a project uses and using automated security scanners, threat modeling can be a powerful asset.

The use of **bug bounties** continues to grow in popularity for many organizations, including enterprises and notable government bodies. A bug-bounty program encourages participants to identify potentially damaging vulnerabilities in return for reward or recognition. Companies like HackerOne and Bugcrowd offer services to help organizations manage this process more easily, and we're seeing these services gather adoption.



A **Data Lake** is an immutable data store of largely unprocessed "raw" data, acting as a source for data analytics. While the technique can clearly be misused, we have used it successfully at clients, hence motivating its move to trial. We continue to recommend other approaches for operational collaborations, limiting the use of the data lake to reporting, analytics and feeding data into data marts.

We see continued adoption and success of **reactive architectures**, with reactive language extensions and reactive frameworks being very popular (we added several such blips in this edition of the Radar). User interfaces, in particular, benefit greatly from a reactive style of programming. Our caveats last time still hold true: Architectures based on asynchronous message passing introduce complexity and make the overall system harder to understand—it's no longer possible to simply read the program code and understand what the system does. We recommend assessing the performance and scalability needs of your system before committing to this architectural style.

We are finding **Content Security Policies** to be a helpful addition to our security toolkit when dealing with websites that pull assets from mixed contexts. The policy defines a set of rules about where assets can come from (and whether to allow inline script tags). The browser then refuses to load or execute JavaScript, CSS or images that violate those rules. When used in conjunction with good practices, such as output encoding, it provides good mitigation for XSS attacks. Interestingly, the optional endpoint for posting JSON reports of violations is how Twitter discovered that ISPs were injecting HTML or JavaScript into their pages.

ADOPT
1. Decoupling deployment from release
2. Products over projects
3. Threat Modeling

TRIAL
4. BFF - Backend for frontends
5. Bug bounties
6. Data Lake
7. Event Storming
8. Flux
9. Idempotency filter
10. iFrames for sandboxing
11. NPM for all the things
12. Phoenix Environments
13. QA in production
14. Reactive architectures

ASSESS
15. Content Security Policies
16. Hosted IDE's
17. Hosting PII data in the EU
18. Monitoring of invariants
19. OWASP ASVS
20. Serverless architecture
21. Unikernels
22. VR beyond gaming

HOLD
23. A single CI instance for all teams
24. Big Data envy
25. Gitflow
26. High performance envy/web scale envy
27. SAFe™

TECHNIQUES *continued*

In a number of countries around the world, we see government agencies seeking broad access to private, personally identifiable information (PII). In the EU, the highest court has invalidated the Safe Harbor framework, and Privacy Shield, its successor, is expected to be challenged too. At the same time, the use of cloud computing is increasing, 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, protected by the most progressive privacy laws, by **Hosting PII in the EU**.

As more development teams incorporate security earlier in the development life cycle, figuring out requirements to limit security risks can seem like a daunting task. Few people have the extensive technical knowledge needed to identify all the risks that an application might face, and teams might struggle just trying to decide where to begin. Relying on frameworks such as OWASP's **ASVS** (Application Security Verification Standard) can help make this easier. Although somewhat lengthy, it contains a thorough list of requirements categorized by functions such as authentication, access control, and error handling and logging, which can be reviewed as needed. It is also helpful as a resource for testers when it comes time to verify software.

Serverless architecture replaces long-running virtual machines with ephemeral compute power that comes into existence on request and disappears immediately after use. Examples include Firebase and AWS Lambda. Use of this architecture can mitigate some security concerns such as security patching and SSH access control, and can make much more efficient use of compute resources. These systems cost very little to operate and can have inbuilt scaling features (this is especially true for AWS Lambda). An example architecture could be a JavaScript app with static assets served by a CDN or S3 coupled with AJAX calls served by the API Gateway and Lambda. While serverless architectures have significant benefits, there are drawbacks too: Deploying, managing and sharing code across services is more complex, and local or offline testing is more difficult if not impossible.

With the continued rise to domination of the container model led by Docker adoption, we think it's worth calling attention to the continued rapid development in the **Unikernel** space. Unikernels are single-purpose library operating systems that can be compiled down from high-level languages to run directly on the hypervisors used by commodity cloud platforms. They promise a number of advantages over containers, not least their superfast startup time and very small attack surface area. Many are still at the research-project phase—Drawbridge from Microsoft Research, MirageOS and HaLVM amongst others—but we think the ideas are very interesting and combine nicely with the technique of serverless architecture.

The idea of virtual reality has been around for more than 50 years, and with successive improvements of computing technology many ideas have been hyped and explored. We believe that we're reaching a tipping point now. Modern graphics cards provide sufficient compute power to render detailed, realistic scenes in high resolutions, and at the same time at least two consumer-oriented VR headsets (the HTC Vive and Facebook's Oculus Rift) are coming to market. These headsets are affordable, they have high-resolution displays, and they eliminate perceivable motion-tracking lag, which was causing issues such as headaches and nausea before. The headsets are mainly targeted at enthusiast video gaming, but we are convinced that they will open many possibilities for **VR beyond gaming**, particularly as the low-fi approaches, such as Google Cardboard, are driving greater awareness.

There might be the impression that it's easier to manage a **single CI (Continuous Integration) instance for all teams** because it gives them a single configuration and monitoring point. But a bloated instance that is shared by every team in an organization can cause a lot of damage. We have found that problems like build timeouts, configuration conflicts and gigantic build queues appear more frequently. Having this single point of failure can interrupt the work of many teams. Carefully consider the trade-off between these pitfalls and having a single point of configuration. In organizations with multiple teams, we recommend having CI instances distributed by teams, with enterprise decisions based not on the single CI installation but on defining guidelines about the instances' selection and configuration.

TECHNIQUES *continued*

While we've long understood the value of Big Data to better understand how people interact with us, we've noticed an alarming trend of **Big Data envy**: organizations using complex tools to handle "not-really-that-big" Data. Distributed map-reduce algorithms are a handy technique for large data sets, but many data sets we see could easily fit in a single-node relational or graph database. Even if you do have more data than that, usually the best thing to do is to first pick out the data you need, which can often then be processed on such a single node. So we urge that before you spin up your clusters, take a realistic assessment of what you need to process, and if it fits—maybe in RAM—use the simple option.

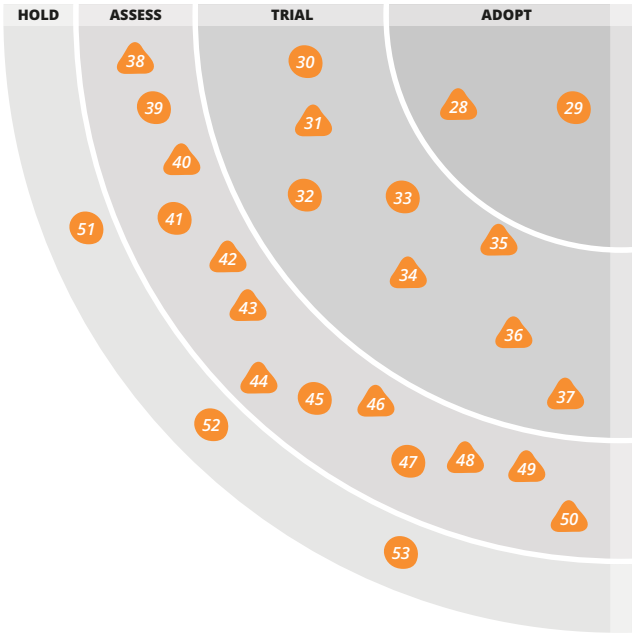
PLATFORMS

We remain excited about **Docker** as it evolves from a tool to a complex platform of technologies. Development teams love Docker, as the Docker image format makes it easier to achieve parity between development and production, making for reliable deployments. It is a natural fit in a microservices-style application as a packaging mechanism for self-contained services. On the operational front, Docker support in monitoring tools (Sensu, Prometheus, cAdvisor, etc.), orchestration tools (Kubernetes, Marathon, etc.) and deployment-automation tools reflect the growing maturity of the platform and its readiness for production use. A word of caution, though: There is a prevalent view of Docker and Linux containers in general as being "lightweight virtualization," but we would not recommend using Docker as a secure process-isolation mechanism, though we are paying attention to the introduction of user namespaces and seccomp profiles in version 1.10 in this regard.

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 to produce highly scalable systems with invisible infrastructure. We have run into significant problems using Java for Lambda functions, with erratic latencies up to several seconds as the Lambda container is started. We recommend sticking with JavaScript or Python for the time being.

Kubernetes is Google's answer to the problem of deploying containers into a cluster of machines, which is becoming an increasingly common scenario. It is not the solution used by Google internally but an open source project that originated at Google and has seen a fair number of external contributions. Since we mentioned Kubernetes on the previous Radar, our initial positive impressions have been confirmed, and we are seeing successful use of Kubernetes in production at our clients.

In earlier versions of the Radar, we have highlighted the value of **Linux security modules**, talking about how they enable people to think about server hardening as a part of their development workflow. More recently, with LXC and Docker containers now shipping with default AppArmor profiles on certain Linux distributions, it has forced the hand of many teams to understand how these tools work. In the event that teams use container images to run any process that they did not themselves create, these tools help them assess questions about who has access to what resources on the shared host and the capabilities that these contained services have, and be conservative in managing levels of access.



ADOPT
28. Docker
29. TOTP Two-Factor Authentication

TRIAL
30. Apache Mesos
31. AWS Lambda
32. H2O
33. HSTS
34. Kubernetes
35. Linux security modules
36. Pivotal Cloud Foundry
37. Rancher

ASSESS
38. Amazon API Gateway
39. AWS ECS
40. Bluetooth Mesh
41. Ceph
42. Deflect
43. ESP8266
44. MemSQL
45. Mesosphere DCOS
46. Nomad
47. Presto
48. Realm
49. Sandstorm
50. TensorFlow

HOLD
51. Application Servers
52. Over-ambitious API Gateways
53. Superficial private cloud

PLATFORMS *continued*

The PaaS space has seen a lot of movement since we last mentioned Cloud Foundry in 2012. While there are various distributions of the open source core, we have been impressed by the offering and ecosystem assembled as **Pivotal Cloud Foundry**. While we expect continued convergence between the unstructured approach (Docker, Mesos, Kubernetes, etc.) and the more structured and opinionated buildpack style offered by Cloud Foundry and others, we see real benefit for organizations that are willing to accept the constraints and rate of evolution to adopt a PaaS. Of particular interest is the speed of development that comes from the simplification and standardization of the interaction between development teams and platform operations.

The emerging Containers as a Service (CaaS) space is seeing a lot of movement and provides a useful option between basic IaaS (Infrastructure as a Service) and more opinionated PaaS (Platform as a Service). While **Rancher** creates less noise than some other players, we have enjoyed the simplicity that it brings to running Docker containers in production. It can run stand-alone as a full solution or in conjunction with tools like Kubernetes.

Amazon API Gateway is Amazon's offering enabling developers to expose API services to Internet clients, offering 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 over-ambitious API gateways, but at this stage Amazon's offering appears to be lightweight enough to avoid those problems.

While many deployments of smart devices rely on Wi-Fi connectivity, we have been seeing success with **Bluetooth Mesh** networks that don't necessitate a hub or gateway. With better energy usage than Wi-Fi and better smartphone adoption than ZigBee, Bluetooth LE deployed as a self-healing mesh provides interesting new approaches for connecting local device-area networks. We are still waiting for the formal approach to emerge from the Bluetooth SIG but have already had successful deployments. We particularly like the lack of infrastructure required to stand up a decentralized network but still retain the option to "progressively enhance" the system with the addition of a gateway and cloud services.

Deflect is an open source service protecting NGOs, activist and independent media companies from DDoS attacks. Similar to a commercial CDN, it uses distributed reverse-proxy caching and also hides your server IP addresses and blocks public access to admin URLs. Particular effort is put in to combat the botnets typically used for extrajudicial censoring of independent voices.

Our growing ranks of hardware hackers have been excited by the **ESP8266** Wi-Fi microcontroller. Rather than a specific technology innovation, it is the combination of low price point and small form factor that has sparked an inflection point in people's thinking about what is now feasible to achieve with custom hardware devices. Its main characteristics are: Wi-Fi capabilities (it can act as station, access point or a combination of both), low power, open hardware, Arduino SDK programmability, Lua programmability, huge community support and low cost compared with other IoT modules.

As Moore's Law predicts, we continue to increase the capacity of computer systems and reduce their cost, and so new processing techniques become possible that only a few years ago would have seemed out of reach. One of these techniques is the in-memory database: Instead of using slow disks or relatively slow SSDs to store data, we can keep it in memory for high performance. One such in-memory database, **MemSQL**, is making waves because it is horizontally scalable across a cluster and provides a familiar SQL-based query language. MemSQL also connects to Spark for analytics against real-time data, rather than stale data in a warehouse.

HashiCorp continues to turn out interesting software. The latest to catch our attention is **Nomad**, which is competing in the ever-more-populated scheduler arena. Major selling points include not just being limited to containerized workloads, and operating in multi–data center / multiregion deployments.

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, and our teams have enjoyed using it. Note that migrations are not quite as straightforward as the Realm documentation would have you believe. Still, Realm has us excited, and we suggest you take a look.

PLATFORMS *continued*

For people who want the benefit of cloud-based collaboration tools but don't want to inadvertently "become the product" of a major cloud provider, **Sandstorm** provides an interesting open source alternative with the potential for self-hosting. Of particular interest is the isolation approach, whereby containerization is applied per document rather than per application, and syscall whitelisting is added to further secure the sandbox.

Google's **TensorFlow** is an open source machine-learning platform that can be used for everything from research through to production and will run on hardware from a mobile CPU all the way to a large GPU compute cluster. It's an important platform because it makes implementing deep-learning algorithms much more accessible and convenient. Despite the hype, though, TensorFlow isn't really anything new algorithmically: All of these techniques have been available in the public domain via academia for some time. It's also important to realize that most businesses are not yet doing even basic predictive analytics and that jumping to deep learning likely won't help make sense of most data sets. For those who do have the right problem and data set, however, TensorFlow is a useful toolkit.

TOOLS

We have moved **Consul**, the service-discovery tool supporting both DNS- and HTTP-based discovery mechanisms, into Adopt. It goes beyond other discovery tools by providing customizable health checks for registered services, ensuring that unhealthy instances are marked accordingly. More tools have emerged to work with Consul to make it even more powerful. Consul Template enables configuration files to be populated with information from Consul, making things like client-side load balancing using mod_proxy much easier. In the world of Docker, registrator can automatically register Docker containers as they appear with Consul with extremely little effort, making it much easier to manage container-based setups. You should still think long and hard about whether you need a tool like this or whether something simpler will do, but if you decide you need service discovery, you won't go far wrong with Consul.

Many organizations are now looking closely at new data architectures that capture information as immutable sequences of events at scale. **Apache Kafka** continues to build momentum as an open source messaging framework that provides a solution for publishing ordered event feeds to large numbers of independent, lightweight consumers. Configuring Kafka is nontrivial, but our teams are reporting positive experiences with the framework.

Gauge is a lightweight cross-platform test-automation tool. Specifications are written in free-form Markdown so test cases can be written in the business language, as opposed to using the more common but restrictive "given-when-then" format. Language and IDE support are implemented as plugins to a single core implementation, allowing testers to use the same IDEs as the rest of the team, with powerful capabilities such as autocompletion and refactoring. This tool, open sourced by ThoughtWorks, also supports parallel execution out of the box for all supported platforms.

Let's Encrypt first appeared on the Radar last edition, and since December 2015 this project has moved its

beta status from private to public, meaning users will no longer be required to have an invitation in order to try it. Let's Encrypt grants access to a simpler mechanism to obtain and manage certificates for a larger set of users who are seeking a way to secure their websites. It also promotes a big step forward in terms of security and privacy. This trend has already begun within ThoughtWorks, and many of our projects now have certificates verified by Let's Encrypt.

Load Impact is a SaaS load-testing tool that can generate highly realistic loads of up to 1.2 million concurrent users. Record and playback web interactions using a Chrome plugin simulate network connections for mobile or desktop users and generate load from up to 10 different locations around the world. While not the only on-demand load-testing tool we've used—we also like BlazeMeter—our teams were very enthusiastic about Load Impact.

ADOPT
54. Consul

TRIAL
55. Apache Kafka
56. Browsersync
57. Carthage
58. Gauge
59. GitUp
60. Let's Encrypt
61. Load Impact
62. OWASP Dependency-Check
63. Serverspec
64. SysDig
65. Webpack
66. Zipkin

ASSESS
67. Apache Flink
68. Concourse CI
69. Gitrob
70. Grasp
71. HashiCorp Vault
72. ievms
73. Jepsen
74. LambdaCD
75. Pinpoint
76. Pitest
77. Prometheus
78. RAML
79. Repsheet
80. Sleepy Puppy

HOLD
81. Jenkins as a deployment pipeline

TOOLS *continued*

In a world full of libraries and tools that simplify the life of many software developers, deficiencies in their security have become visible and have increased the vulnerability surface in the applications that use them. **OWASP Dependency-Check** automatically identifies potential security problems in the code, checking if there are any known publicly disclosed vulnerabilities, then using methods to constantly update the database of public vulnerabilities. Dependency-Check has some interfaces and plugins to automate this verification in Java and .NET (which we have used successfully) as well as Ruby, Node.js and Python.

In the past we have included automated Provisioning Testing as a recommended technique, and in this issue we highlight **Serverspec** as a popular tool for implementing those tests. Although this tool is not new, we are seeing its use become more common as more cross-functional delivery teams take on responsibility for infrastructure provisioning. Serverspec is built on the Ruby library RSpec and comes with a comprehensive set of helpers for asserting that server configuration is correct.

Webpack has solidified itself as our go-to JavaScript module bundler. With its ever-growing list of loaders, it provides a single dependency tree for all your static assets, allowing flexible manipulation of JavaScript, CSS, etc. and minimizing what needs to be sent to the browser and when. Of particular relevance is the smooth integration among AMD, CommonJS and ES6 modules and how it has enabled teams to work in ES6 and seamlessly transpile (using Babel) to earlier versions for browser compatibility. Many of our teams also value Browserify, which covers a similar space but is more focused on making Node.js modules available for client-side use.

Development on **Zipkin** has continued apace, and since the middle of 2015 it has moved to the openzipkin/zipkin organization at GitHub. There are now bindings for Python, Go, Java, Ruby, Scala and C#; and there are Docker images available for those wanting to get started quickly. We still like this tool. There is an active and growing community around usage of it, and implementation is getting easier. If you need a way of measuring the end-to-end latency of many logical requests, Zipkin continues to be a strong choice.

Apache Flink is a new-generation platform for scalable distributed batch and stream processing. At its core is a streaming data-flow engine. It also supports 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. While it hasn't reached version 1.0, it has raised significant community interest due to innovations in stream processing, memory handling, state management and simplicity of configuration.

Attackers continue to use automated software to crawl public GitHub repositories to find AWS credentials and spin up EC2 instances to mine Bitcoins or for other nefarious purposes. Although adoption of tools like git-crypt and Blackbox to safely store secrets such as passwords and access tokens in code repositories is increasing, it is still all too common that secrets are stored unprotected. It is also not uncommon to see project secrets accidentally checked in to developers' personal repositories. **Gitrob** can help minimize the damage of exposing secrets. It scans an organization's GitHub repositories, flagging all files that might contain sensitive information that shouldn't have been pushed to the repository. The current release of the tool has some limitations: It can only be used to scan public GitHub organizations and their members, it doesn't inspect the contents of files, it doesn't review the entire commit history, and it fully scans all repositories each time it is run. Despite these limitations, it can be a helpful reactive tool to help alert teams before it is too late. It should be considered a complementary approach to a proactive tool such as Talisman.

We had our collective minds blown by a little JavaScript command-line refactoring tool called **Grasp**. Providing a rich set of selectors and operating against the abstract syntax tree, it is leagues ahead of fiddling with sed and grep. A useful addition to the toolkit in our ongoing quest to treat JavaScript as a first-class language.

Having a way to securely manage secrets is increasingly becoming a huge project issue. The old idea of just having a file with secrets or environment variables is becoming hard to manage, especially in environments with multiple applications like microservices or microcontainer environments, where the applications need to access a multitude of secrets. **HashiCorp Vault** is a promising tool that tries to solve the problem by providing mechanisms for securely accessing secrets through an unified interface. It has some features that make life easier, such as encryption and automatically generating secrets for known tools, among others.

TOOLS *continued*

With the growth in usage of NoSQL data stores, and the growth in popularity of polyglot approaches to persistence, teams now have many choices when it comes to storing their data. While this has brought many advantages, product behavior with flaky networks can introduce subtle (and not so subtle) issues that are often not well understood, even in some cases by the product developers themselves. The **Jepsen** toolkit and accompanying blog have become the de-facto reference for anyone looking to understand how different database and queuing technologies react under adverse conditions. Crucially, the approach to testing, which includes clients in the transactions, shines a spotlight on possible failure modes for many teams building microservices.

LambdaCD provides teams with a way to define Continuous Delivery pipelines in Clojure. This brings the benefits of Infrastructure as code to the configuration of CD servers: source-control management, unit testing, refactoring and code reuse. In the "pipelines as code" space, LambdaCD stands out for being lightweight, self-contained and fully programmable, allowing teams to work with their pipelines in the same way that they do with their code.

Teams using the Phoenix Server or Phoenix Environment techniques have found little in the way of support from Application Performance Management (APM) tools. Their licensing models, based on long-running, limited amounts of tin, and their difficulty in dealing with ephemeral hardware, have meant that they are often more trouble than they are worth. However, distributed systems need monitoring, and at some point many teams recognize the need for an APM tool. We think **Pinpoint**, an open source tool in this space, is worth investigating as an alternative to AppDynamics and Dynatrace. Pinpoint is written in Java, with plugins available for many servers, databases and frameworks. While we think you can go a long way using a combination of other lightweight open source tools—Zipkin, for example—if you are in the market for an APM, Pinpoint is worth considering.

Pitest is a test coverage analysis tool for Java that uses a mutation-testing technique. Traditional test coverage analysis tends to measure the number of lines that are executed by your tests. It is therefore only able to identify code that is definitely not tested. Mutation testing, on the other hand, tries to test the quality of those lines that are executed by your test code and yet might contain general errors. Several problems can be spotted this way, helping the team to measure and grow a healthy test suite. Most of such tools tend to be slow and difficult to use, but Pitest has proven to have better performance, is easy to set up, and is actively supported.

Attacks on web properties using bots are becoming more sophisticated. Identifying these bad actors and their behaviors is the goal of the **Repsheet** project. It's a plugin for either Apache or NGINX that records user activity, fingerprints actors using predefined and user-defined rules, and then allows action to be taken, including the ability to block offensive actors. It includes a utility that visualizes current actors; this puts the ability to manage bot-based threats in the hands of team members, increasing security awareness for teams. We like this since it's a good example of a simple tool solving a very real but often invisible problem—bot-based attacks.

We know we're in perilous territory here, since we build a competing tool, but we feel we have to address a persistent problem. Continuous Integration tools like CruiseControl and Jenkins are valuable for software development, but as your build process gets more complex it requires something beyond just Continuous Integration: It requires a deployment pipeline. We frequently see people trying to use **Jenkins as a Deployment Pipeline** with the aid of plugins, but our experience is that these quickly become a tangle. Jenkins 2.0 introduces "Pipeline as Code" but continues to model pipelines using plugins and fails to change the core Jenkins product to model pipelines directly. In our experience, tools that are built around a first-class representation of deployment pipelines are much more suitable, and this is what drove us to replace CruiseControl with GoCD. Today we see several products that embrace deployment pipelines, including ConcourseCI, LambdaCD, Spinnaker, Drone and GoCD.

LANGUAGES & FRAMEWORKS

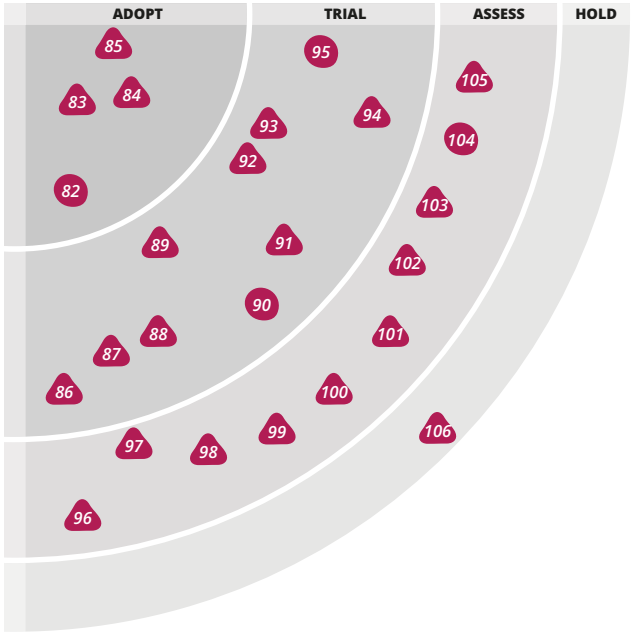
In the avalanche of front-end JavaScript frameworks, **React.js** stands out due to its design around a reactive data flow. Allowing only one-way data binding greatly simplifies the rendering logic and avoids many of the issues that commonly plague applications written with other frameworks. We're seeing the benefits of React.js on a growing number of projects, large and small, while at the same time we continue to be concerned about the state and the future of other popular frameworks like AngularJS. This has led to React.js becoming our default choice for JavaScript frameworks.

A lot of work has gone into **Spring Boot** to reduce complexity and dependencies, which largely alleviates our previous reservations. If you live in a Spring ecosystem and are moving to microservices, Spring Boot is now the obvious choice. For those not in Springland, Dropwizard is also worthy of serious consideration.

Swift is now our default choice for development in the Apple ecosystem. With the release of Swift 2, the language approached a level of maturity that provides the stability and performance required for most projects. A good number of libraries that support iOS development—SwiftyJSON, Quick, etc.—are now migrated over to Swift, which is where the rest of the applications should follow. Swift has now been open sourced, and we are seeing a community of developers dedicated to continuously improving development in iOS.

Butterknife is a field and method binding view-injection library. It allows the injection of arbitrary objects, views and listeners, thereby ensuring cleaner code with reduced glue code for Android development. With Butterknife, multiple views can be grouped into a list or array with common actions applied to the views simultaneously, without heavy reliance on XML configurations. Our project teams have used this library and benefited from its simplicity and ease of use.

With the increased need for Android-based applications, **Dagger** offers a fully static, compile-time dependency-injection framework. Dagger's strictly generated implementation and nonreliance on reflection-based



solutions addresses many of the performance and development issues, thereby making it suitable for Android development. With Dagger, there is full traceability with easy debugging because the entire call stack for provision and creation is made available.

Dapper is a minimal, lightweight ORM of sorts for .NET. Rather than trying to write the SQL queries for you, Dapper maps SQL queries to dynamic objects. Though it's not brand new, Dapper has steadily gained acceptance from ThoughtWorks teams working in .NET. For the C# programmer, it removes some of the drudgery of mapping relational queries to objects while still allowing complete control over the SQL or stored procedures.

Ember.js has developed further support based on project experiences and is clearly a strong contender in the field of JavaScript application frameworks. Ember is praised for its developer experience, with far fewer surprises than other frameworks such as AngularJS. The Ember CLI build tooling, convention-over-configuration approach and ES6 support also gain positive feedback.

ADOPT
82. ES6
83. React.js
84. Spring Boot
85. Swift

TRIAL
86. Butterknife
87. Dagger
88. Dapper
89. Ember.js
90. Enlive
91. Fetch
92. React Native
93. Redux
94. Robolectric
95. SignalR

ASSESS
96. Alamofire
97. AngularJS
98. Aurelia
99. Cylon.js
100. Elixir
101. Elm
102. GraphQL
103. Immutable.js
104. OkHttp
105. Recharts

HOLD
106. JSPatch

LANGUAGES & FRAMEWORKS *continued*

Our teams are moving away from JQuery or raw XHR for remote JavaScript calls and instead are using the new Fetch API and the **Fetch** polyfill in particular. The semantics remain similar but have cleaner support for promises and CORS support. We are seeing this as the new de-facto approach.

We are seeing continued success with **React Native** for rapid cross-platform mobile development. Despite some churn as it undergoes continuing development, the advantages of trivial integration between native and nonnative code and views, the rapid development cycle (instant reload, chrome debugging, Flexbox layout) and general growth of the React style is winning us over. As with many frameworks, care needs to be taken to keep your code well structured, but diligent use of a tool like Redux really helps here.

Redux is a great, mature tool that has helped many of our teams reframe how they think about managing state in client-side apps. Using a Flux-style approach, it enables a loosely coupled state-machine architecture that's easy to reason about. We've found it a good companion to some of our favored JavaScript frameworks, such as Ember and React.

In the Android application-development world, **Robolectric** is a unit-testing framework that has been used by multiple teams within our technical community. It offers the best option among those available for writing real unit tests that extend or interact directly with Android components and support JUnit tests. We caution, though, that because it is an implementation of the Android SDK, there might be device-specific issues for some tests that pass in Robolectric. To manually mock all the Android dependencies, ensuring only test of the system-in-test will require a lot of complex code, and this framework addresses this effectively.

Networking and decoding in iOS applications have been a difficult endeavor for many years. There have been many libraries and attempts to solve this ongoing problem. It looks as though **Alamofire** is the most robust and developer-friendly library to handle decoding JSON. It was written by the same creator as its Objective-C counterpart (AFNetworking), which was used at great length during the Objective-C days.

While we have delivered many successful projects using **AngularJS** and are seeing an acceleration of adoption in corporate settings, we have decided to move Angular back to Assess on this edition of the Radar. This move is intended as a note of caution: React.js and Ember offer strong alternatives; the migration path from Angular version 1 to version 2 is causing uncertainty; and we see some organizations adopting the framework without really thinking through whether a single-page application fits their needs. We have passionate internal debates about this topic but have certainly seen codebases become overly complex from a combination of two-way binding and inconsistent state-management patterns. We believe that rather than requiring that a solid framework be jettisoned, these issues can be solved through careful design and use of Redux or Flux from the outset.

Aurelia is considered the next-generation JavaScript client framework and was written using a modern version of JavaScript: ECMAScript 2016. Aurelia was created by Rob Eisenberg, the creator of Durandal. He left the Angular 2.0 core team to dedicate his time to this project. The great thing about Aurelia is that it's highly modular, contains simple small libraries and is designed to be customized easily. Aurelia follows the pattern of convention over configuration, which enables easier production and consumption of modules, but there are no strong conventions that you have to adhere to. Aurelia has a large community, and in the project website you can learn more by using the tutorials.

The intersection between IoT devices and the JavaScript ecosystem offers interesting possibilities. **Cylon.js** is a JavaScript library for building interfaces for robotics and the Internet of Things, which has excited our technical community. It offers support for 50+ platform devices, as well as general-purpose input/output support with a shared set of drivers provided by the cylon-gpio module. Control of the devices is then possible through a web browser interface.

We continue to see a lot of excitement from people using the **Elixir** programming language. 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 have been prompted to reconsider **Elm** because of the rapid adoption of Redux framework. Elm—the original inspiration for Redux—offers the view componentization and reactiveness of React.js along with the predictable state of Redux in a compiled, strongly typed functional language. Elm is written in Haskell and has a Haskell-like syntax but compiles down to HTML, CSS and JavaScript for the browser. JavaScript programmers rushing to embrace React.js and Redux might want to also consider Elm as a type-safe alternative for some applications.

When we look at REST implementations in the wild, we frequently see REST misused to naively retrieve object graphs through chatty interactions between client and server. Facebook's **GraphQL** is an interesting alternative to REST that might be a better approach for this very common use case. As a protocol for remotely retrieving object graphs, GraphQL has received enormous attention recently. One of GraphQL's most interesting features is its consumer-oriented nature: The structure of a response is driven entirely by the client, not the server. This decouples the consumer and forces the server to obey Postel's law. Client implementations are now available in many programming languages, but we have seen a flurry of interest of Facebook's Relay, a JavaScript framework that was designed to support the React.js stateless component model.

Immutability is often emphasized in the functional programming paradigm, and most languages have the ability to create immutable objects, which cannot 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. Our teams have had value using this library for tracking mutation and maintaining state, and it is a library we encourage developers to investigate, especially when it's combined with the rest of the Facebook stack.

We've been enjoying how **Recharts** integrates D3 charts into React.js in a clean and declarative manner.

Many iOS developers are using **JSPatch** to dynamically patch their apps. When a JSPatch-enabled app runs, it loads a chunk of JavaScript (potentially via an insecure HTTP connection) and then bridges to the main Objective-C application code to change behavior, fix bugs, and so on. While convenient, we think monkey-patching live apps is a bad idea and should be avoided. When doing any amount of incremental patching, it's very important that your testing process matches what end users will experience, in order to properly validate functionality. An alternative approach is to use React Native for the app and AppHub and CodePush to push small updates and new features.

ThoughtWorks is a software company and community of passionate, purpose-led individuals that specialize in software consulting, delivery and products. We think disruptively to deliver technology to address our clients' toughest challenges, all while seeking to revolutionize the IT industry and create positive social change. We make pioneering tools for software teams who aspire to be great. Our products help organizations continuously improve and deliver quality software for their most critical needs. Founded over 20 years ago, ThoughtWorks has grown from a small group in Chicago to a company of over 3500 people spread across 35 offices in 12 countries: Australia, Brazil, Canada, China, Ecuador, Germany, India, Singapore, South Africa,Turkey, the United Kingdom, and the United States.

ThoughtWorks®