ThoughtWorks®

TECHNOLOGY RADAR *VOL.16*

Insights into the technology and trends shaping the future

thoughtworks.com/radar

#TWTechRadar

CONTRIBUTORS

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

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

WHAT'S NEW?

Highlighted themes in this edition:

CONVERSATIONAL UI AND NATURAL LANGUAGE PROCESSING

watch the video (thght.works/ConUI)

Conversation—a new way to interact with applications—took the ecosystem by storm with tools such as Siri, Cortana, and Allo, and then extended into homes with devices such as Amazon Echo and Google Home.

Building conversational and natural language user interfaces, while presenting new challenges, has obvious benefits. The team behind the Echo intentionally omitted a screen, forcing them to rethink many human-machine interactions.

The conversational trend is not just limited to voice; as messaging apps have grown to dominate both phones and workplaces, we see conversations with other humans being supplemented by intelligent chatbots. As these platforms improve, they will learn to understand the context and intent of conversations, making interactions more lifelike and therefore more compelling.

The explosion of interest in the marketplace and mainstream media leads to a corresponding rise in developer interest in this new personal exocortex interaction mode.

INTELLIGENCE AS A SERVICE

watch the video (thght.works/IntSer)

A family of platforms burst onto the scene recently that we call *intelligence as a service*. These platforms encompass a wide variety of surprisingly powerful utilities from voice processing to natural language understanding, image recognition, and deep learning.

Capabilities that would have consumed costly resources a few years ago now appear as open source or SaaS platforms. It appears that the "cloud wars" have moved from competing on storage and compute to cognitive capabilities, as witnessed by the willingness to open-source previously differentiating tools such as Kubernetes and Mesos.

All the big players have offerings in this space, along with interesting niche players worth assessing. Although we still have reservations about the ethical and privacy implications of these services, we see great promise in utilizing these powerful tools in novel ways. Our clients are already investigating what new horizons they may expose by combining commodity cognition with intelligence about their own businesses.

DEVELOPER EXPERIENCE AS THE NEW DIFFERENTIATOR

watch the video (thght.works/DevExp)

User experience design has been a key differentiator for technology product companies for many years. Now the rapid rise of developer-facing tools and products, combined with the scarcity of engineering talent, is driving a similar focus on developer experience.

Increasingly, organizations evaluate cloud offerings based on the amount of engineering friction they reduce, treat APIs as products, and spin up teams focused on engineering productivity. At ThoughtWorks, we have always obsessed over efficient engineering practices and promoted tools and platforms that make developers' lives easier, so it excites us to see the industry beginning to adopt this approach.

Key techniques include: treating internal infrastructure as a product that needs to be compelling enough to compete with external offerings, focusing on self-service, understanding the developer ergonomics of the APIs you produce, containing "legacy in a box", and committing to ongoing empathetic user research of the developers using your services.

THE RISE OF PLATFORMS

watch the video (thght.works/RiseOTP)

The Radar themes emerge from observations and conversations during the vetting process; recently, while compiling the Radar, we've noticed the number of new entries in the Platforms quadrant. We think this is indicative of a broader trend in the software development ecosystem.

Notable Silicon Valley companies have illustrated how building a suitable platform can yield significant benefits. Part of their success comes from finding a useful level of encapsulation and capabilities. Increasingly, "platform thinking" appears across the ecosystem—from advanced capabilities highlighted on the Radar such as natural language, to infrastructure platforms such as Amazon.

Businesses are starting to think about platforms when exposing select capabilities via product-inspired APIs. Development teams think more in terms of building platforms for integration and improved developer experience. It seems the industry has finally latched onto a reasonable combination of packaging, convenience, and usefulness.

One definition that we like is that platforms should expose a self-service API and be easy to configure and provision within a team environment—which intersects nicely with another emerging theme, *developer experience as the new differentiator*. We expect to see further refinement in both the definition and capabilities of platforms in the near future.

PERVASIVE PYTHON

watch the video (thght.works/PerPyt)

Python is a language that keeps popping up in interesting places. Its ease of use as a general programming language, combined with its strong foundation in mathematical and scientific computing has historically led to its grassroots adoption by the academic and research communities. More recently, industry trends around AI commoditization and applications, combined with the maturity of Python 3, have helped bring new communities into the Python fold.

This edition of the Radar features a few Python libraries that have helped boost the ecosystem, including Scikit-learn in the machine learning domain; TensorFlow, Keras, and Airflow for smart data flow graphs; and spaCy which implements natural language processing to help empower conversationally aware APIs. Increasingly, we see Python bridging the gap between the scientists and engineers within organizations, loosening past prejudice against their favorite tools.

Architectural approaches such as microservices and containers have eased the execution of Python in production environments. Engineers can now deploy and integrate specialized Python code created by scientists through language- and technology-agnostic APIs. This fluidity is a great step toward a consistent ecosystem between researchers and engineers, in contrast to the de facto practice of translating specialized languages such as R to the production environments.

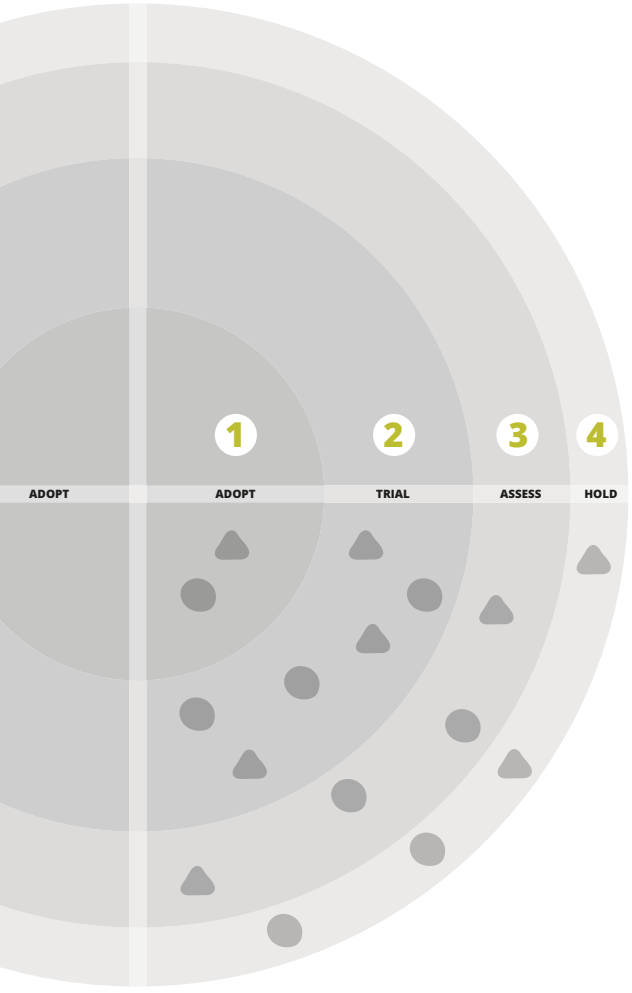
ABOUT THE RADAR

ThoughtWorkers are passionate about technology. We build it, research it, test it, open source it, write about it, and constantly aim to improve it—for everyone. Our mission is to champion software excellence and revolutionize IT. We create and share the ThoughtWorks Technology Radar in support of that mission. The ThoughtWorks Technology Advisory Board, a group of senior technology leaders in ThoughtWorks, creates the Radar. They meet regularly to discuss the global technology strategy for ThoughtWorks and the technology trends that significantly impact our industry.

The Radar captures the output of the Technology Advisory Board's discussions in a format that provides value to a wide range of stakeholders, from developers to CTOs. The content is intended as a concise summary.

We encourage you to explore these technologies for more detail. The Radar is graphical in nature, grouping items into techniques, tools, platforms, and languages & frameworks. When Radar items could appear in multiple quadrants, we chose the one that seemed most appropriate. We further group these items in four rings to reflect our current position on them.

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



RADAR AT A GLANCE

1 ADOPT

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

2 TRIAL

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

3 ASSESS

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

4 HOLD

Proceed with caution.

▲ NEW OR CHANGED

● NO CHANGE

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

! Our Radar is forward looking. To make room for new items, we fade items that haven't moved recently, which isn't a reflection on their value but rather our limited Radar real estate.

THE RADAR

TECHNIQUES

ADOPT
1. Pipelines as code

TRIAL
2. APIs as a product
3. Decoupling secret management from source code *NEW*
4. Hosting PII data in the EU
5. Legacy in a box *NEW*
6. Lightweight Architecture Decision Records
7. Progressive Web Applications *NEW*
8. Prototyping with InVision and Sketch *NEW*
9. Serverless architecture

ASSESS
10. Client-directed query
11. Container security scanning
12. Conversationally aware APIs *NEW*
13. Differential privacy
14. Micro frontends
15. Platform engineering product teams *NEW*
16. Social code analysis *NEW*
17. VR beyond gaming

HOLD
18. A single CI instance for all teams
19. Anemic REST
20. Big Data envy
21. CI theatre *NEW*
22. Enterprise-wide integration test environments *NEW*
23. Spec-based codegen *NEW*

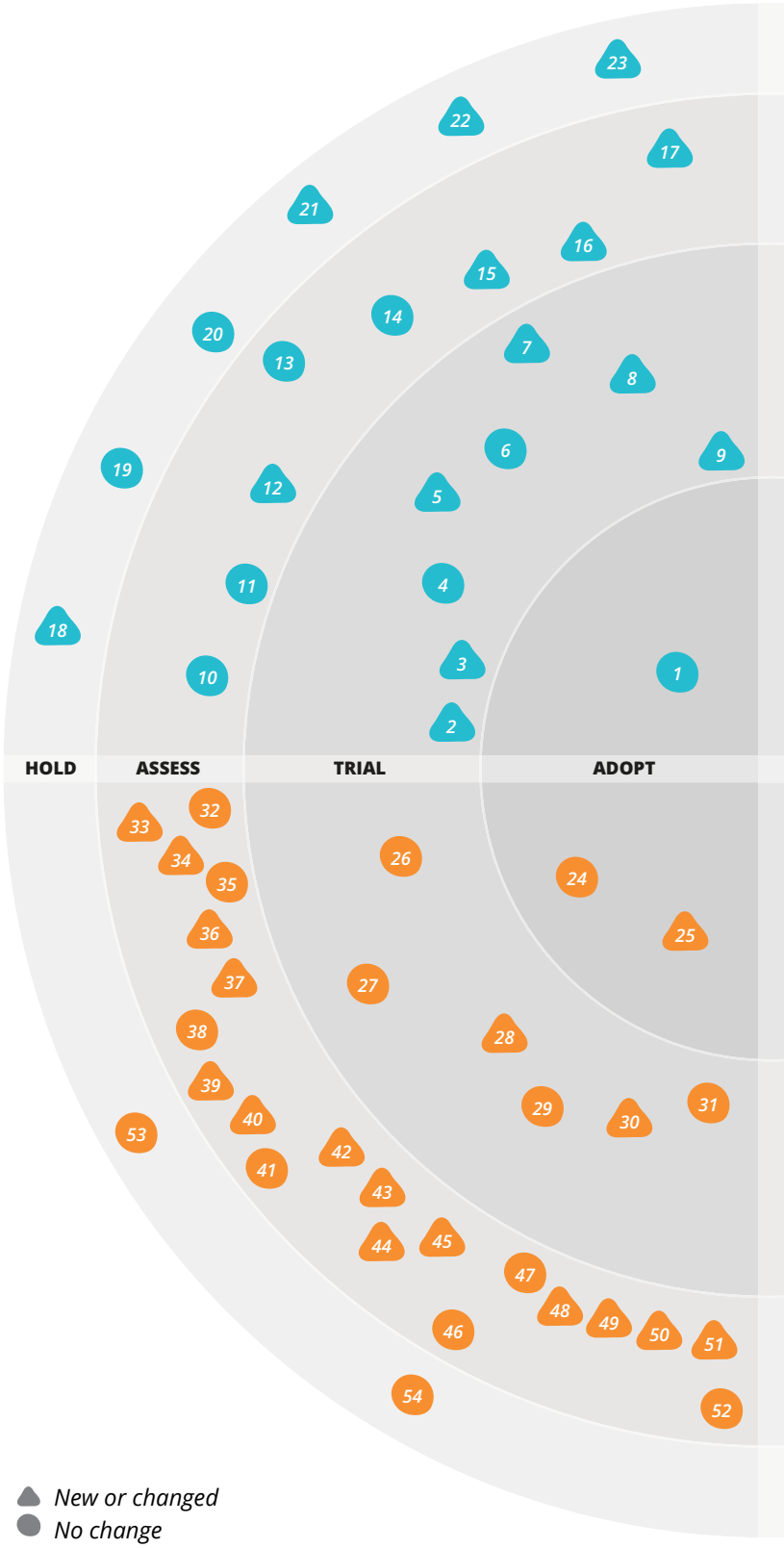
PLATFORMS

ADOPT
24. HSTS
25. Linux Security Modules

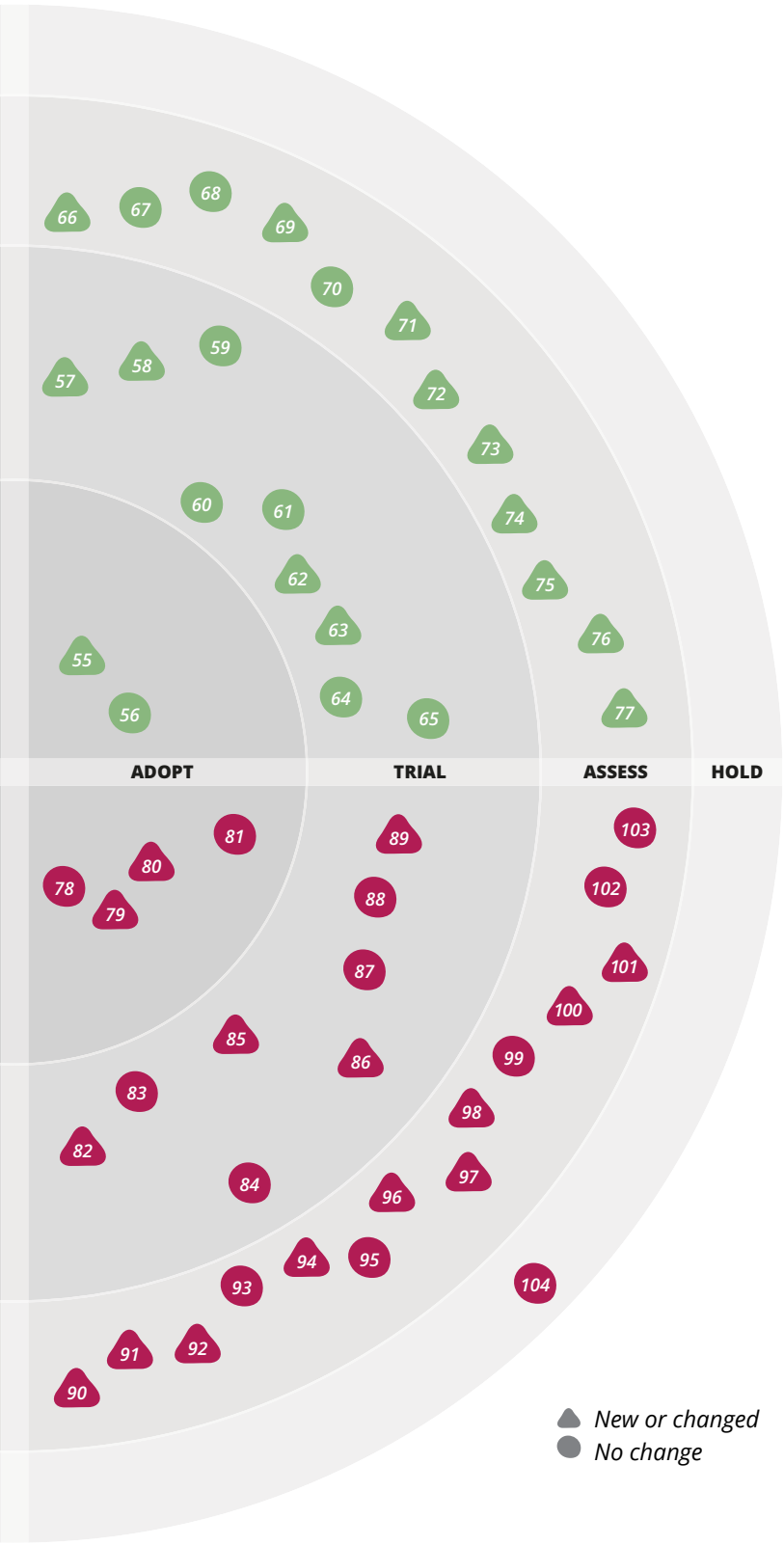
TRIAL
26. Apache Mesos
27. Auth0
28. AWS Device Farm *NEW*
29. AWS Lambda
30. OpenTracing *NEW*
31. Unity beyond gaming

ASSESS
32. .NET Core
33. Amazon API Gateway
34. api.ai *NEW*
35. Cassandra carefully
36. Cloud-based image comprehension *NEW*
37. DataStax Enterprise Graph *NEW*
38. Electron
39. Ethereum
40. Hyperledger *NEW*
41. IndiaStack
42. Kafka Streams *NEW*
43. Keycloak *NEW*
44. Mesosphere DCOS
45. Mosquitto *NEW*
46. Nuance Mix
47. OpenVR
48. PlatformIO *NEW*
49. Tango *NEW*
50. Voice platforms *NEW*
51. WebVR *NEW*
52. wit.ai

HOLD
53. CMS as a platform
54. Overambitious API gateways



THE RADAR



TOOLS

ADOPT
55. fastlane
56. Grafana

TRIAL
57. Airflow *NEW*
58. Cake and Fake *NEW*
59. Galen
60. HashiCorp Vault
61. Pa11y
62. Scikit-learn
63. Serverless Framework *NEW*
64. Talisman
65. Terraform

ASSESS
66. Amazon Rekognition *NEW*
67. Android-x86
68. Bottled Water
69. Claudia *NEW*
70. Clojure.spec
71. InSpec *NEW*
72. Molecule *NEW*
73. Spacemacs *NEW*
74. spaCy *NEW*
75. Spinnaker *NEW*
76. Testinfra *NEW*
77. Yarn *NEW*

HOLD

LANGUAGES & FRAMEWORKS

ADOPT
78. Ember.js
79. Python 3
80. ReactiveX
81. Redux

TRIAL
82. Avro *NEW*
83. Elixir
84. Enzyme
85. Hangfire *NEW*
86. Nightwatch *NEW*
87. Phoenix
88. Quick and Nimble
89. Vue.js

ASSESS
90. Angular 2 *NEW*
91. Caffe *NEW*
92. DeepLearning.scala *NEW*
93. ECMAScript 2017
94. Instana *NEW*
95. JuMP
96. Keras *NEW*
97. Knet.jl *NEW*
98. Kotlin *NEW*
99. Physical Web
100. PostCSS *NEW*
101. Spring Cloud *NEW*
102. Three.js
103. WebRTC

HOLD
104. AngularJS

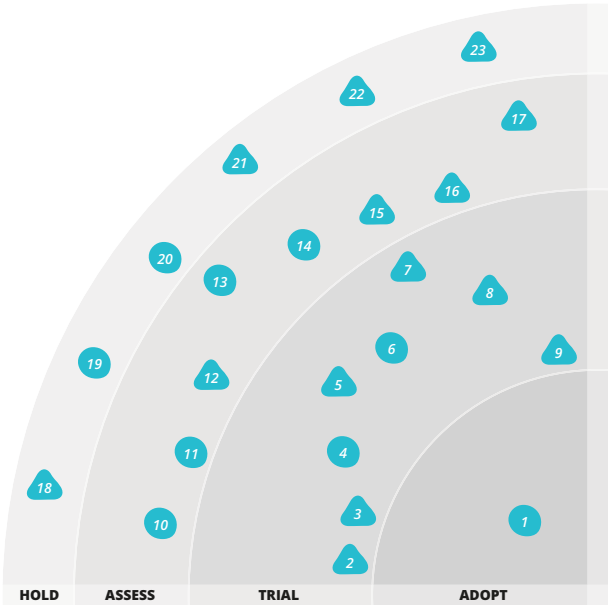
TECHNIQUES

ADOPT
1. Pipelines as code

TRIAL
2. APIs as a product
3. Decoupling secret management from source code *NEW*
4. Hosting PII data in the EU
5. Legacy in a box *NEW*
6. Lightweight Architecture Decision Records
7. Progressive Web Applications *NEW*
8. Prototyping with InVision and Sketch *NEW*
9. Serverless architecture

ASSESS
10. Client-directed query
11. Container security scanning
12. Conversationally aware APIs *NEW*
13. Differential privacy
14. Micro frontends
15. Platform engineering product teams *NEW*
16. Social code analysis *NEW*
17. VR beyond gaming

HOLD
18. A single CI instance for all teams
19. Anemic REST
20. Big Data envy
21. CI theatre *NEW*
22. Enterprise-wide integration test environments *NEW*
23. Spec-based codegen *NEW*



Companies 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 or fellow developer. Teams that build APIs should understand the needs of their customers and make the product compelling to them. Usability testing and UX research can lead to a better design and understanding of the API usage patterns and help bring a product mindset to APIs. APIs, like products, should be actively maintained and supported, and, easy to use. They should have an owner who advocates for the customer and strives for continual improvement. In our experience, 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 previous Radars issues we mentioned tools such as git-crypt and Blackbox that allow us to keep secrets safe inside the source code. **DECOUPLING SECRET MANAGEMENT FROM SOURCE CODE** is our way to remind technologists that there are other options for storing secrets. For example, HashiCorp vault, CI servers and configuration management tools provide mechanisms for storing secrets that are not linked to the source code of an application. Both approaches are viable and we recommend you use at least one of them in your projects.

Teams that build APIs should understand the needs of their customers and make the product compelling to them. Usability testing and UX research can lead to a better design and understanding of the API usage patterns and help bring a product mindset to APIs.

— APIs as a product

Working with legacy code, especially large monoliths, is one of the most unsatisfying, high-friction experiences for developers. Although we caution against extending and actively maintaining legacy monoliths, they continue to be dependencies in our environments, and developers often underestimate the cost and time required to develop against these dependencies. To help reduce the friction, developers have used virtualized machine images or container images with Docker containers to create immutable images of legacy systems and their configurations. The intent is to contain the **LEGACY IN A BOX** for developers to run locally and remove the need for rebuilding, reconfiguring or sharing environments. In an ideal scenario, teams that own legacy systems generate the corresponding boxed legacy images through their build pipelines, and developers can then run and orchestrate these images in their allocated sandbox more reliably. Although this approach has reduced the overall time spent by each developer, it has had limited success when the teams owning the downstream dependencies have been reluctant to create container images for others to use.

The increase in **PROGRESSIVE WEB APPLICATIONS** (PWAs) is the latest attempt to bring back the mobile web in response to users' "app fatigue". Originally proposed by Google in 2015, PWAs are web applications that take advantage of the latest technologies to combine the best of web and native mobile applications. Using a set of open standard technologies such as, service workers, the app manifest, and cache and push APIs, we can create applications that are platform independent and deliver app-like user experiences. This brings parity to web and native applications and helps mobile developers reach users beyond the walled garden of the app stores. Think of PWAs as websites that act and feel like native apps.

The combined use of InVision and Sketch has changed the way some people approach web application development. Although these are tools, it is really the technique of **PROTOTYPING WITH INVISION AND SKETCH** that makes this blip significant. Creating rich, clickable prototypes as the starting point for implementing front-end and back-end behavior helps speed up the development and eliminates churn in the implementation details. This combined use of these tools strikes the right balance between premature elaboration of visual detail and capturing early user feedback on the interactive experience.

A **SERVERLESS ARCHITECTURE** approach replaces long-running virtual machines with ephemeral compute power that comes into existence on request and disappears immediately after use. Our teams like the serverless approach; it's working well for us 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 AWS Lambda is almost synonymous with serverless, the other major cloud providers all have similar offerings, and we also recommend assessing niche players such as webtask.

Technologies such as Amazon Alexa, Google Voice and Siri have dramatically lowered the bar for voice-based interaction with software. However, a more conversational style of input (voice or text) can be hard to build on top of many existing APIs

— Conversationally aware APIs

Technologies such as Amazon Alexa, Google Voice and Siri have dramatically lowered the bar for voice-based interaction with software. However, a more conversational style of input (voice or text) can be hard to build on top of many existing APIs, especially when it comes to a more stateful style of interaction where a follow-up interaction needs to be aware of the overall conversational context. In this style of interaction, for example, we'd like to inquire about trains from Manchester to Glasgow and then being able to ask "What time is the first departure?" without having to give the context of the conversation again. Normally this context would be present in the initial response we send back to a browser, but in the case of voice interfaces we need to handle this context elsewhere. **CONVERSATIONALLY AWARE APIS** can be an example of the backend for front-end pattern where the front-end is a voice or chat platform. This type of API can handle the specifics of this style of interaction by

managing conversation states while calling underlying services on behalf of the voice front-end.

The adoption of cloud and DevOps, while increasing the productivity of teams who can now move more quickly with reduced dependency on centralized operations teams and infrastructure, also has constrained teams who lack the skills to self-manage a full application and operations stack. Some organizations have tackled this challenge by creating **PLATFORM ENGINEERING PRODUCT TEAMS**. These teams operate an internal platform which enables delivery teams to self-service deploy and operate systems with reduced lead time and stack complexity. The emphasis here is on API-driven self-service and supporting tools, with delivery teams still responsible for supporting what they deploy onto the platform. Organizations that consider establishing such a platform team should be very cautious not to accidentally create a separate DevOps team, nor should they simply relabel their existing hosting and operations structure as a platform.

SOCIAL CODE ANALYSIS enriches our understanding of the code quality by overlaying a developer's behavior with the structural analysis of the code. It uses data from version control systems, such as frequency and time of the change as well as the person making the change. You can choose to write your own scripts to analyze such data or use tools such as CodeScene. CodeScene can help you gain a better understanding of your software systems by identifying hotspots and complex, hard-to-maintain subsystems, coupling between distributed subsystems through temporal coupling, as well as the view of Conway's law in your organization. We believe that with technology trends such as distributed systems, microservices and distributed teams the social dimension of our code is vital in our holistic understanding of our systems' health.

The idea of virtual reality has been around for more than 50 years, and with successive advancements in computing technology many ideas have been hyped and explored. We believe that we've reached a tipping point. Reasonably affordable consumer-oriented VR headsets were shipped to the market last year, and modern graphics cards provide sufficient power to create immersive experiences with them. The headsets are mainly targeted at video game enthusiasts, but we're convinced that they'll open the doors to many possibilities for **VR BEYOND GAMING**. Teams without experience in building video games should not underestimate the effort and skill required to create good 3-D models and convincing textures.

The idea of virtual reality has been around for more than 50 years, and with successive advancements in computing technology many ideas have been hyped and explored. We believe that we've reached a tipping point.

— VR beyond gaming

We're compelled to caution, again, against creating **A SINGLE CI INSTANCE FOR ALL TEAMS**. While it's a nice idea in theory to consolidate and centralize Continuous Integration (CI) infrastructure, in reality we do not see enough maturity in the tools and products in this space to achieve the desired outcome. Software delivery teams which must use the centralized CI offering regularly have long delays depending on a central team to perform minor configuration tasks, or to troubleshoot problems in the shared infrastructure and tooling. At this stage, we continue to recommend that organizations limit their centralized investment to establishing patterns, guidelines and support for delivery teams to operate their own CI infrastructure.

We've long been advocates of continuous integration (CI), and we were pioneers in building CI server programs to automatically build projects on check-ins. Used well, these programs run as a daemon process on a shared project mainline that developers commit to daily. The CI server builds the project and runs comprehensive tests to ensure the whole software system is integrated and is in an always-releasable state, thus satisfying the principles of continuous delivery. Sadly, many developers simply set up a CI server and falsely assume they are "doing CI" when in reality they miss out on all the benefits. Common failure modes include: running CI against a shared mainline but with infrequent commits, so integration isn't really continuous; running a build with poor test coverage; allowing the build to stay red for long periods;

or running CI against feature branches which results in continuous isolation. The ensuing "**CI THEATRE**" might make people feel good, but would fail any credible CI certification test.

Sadly, many developers simply set up a CI server and falsely assume they are "doing CI" when in reality they miss out on all the benefits. The ensuing "CI THEATRE" might make people feel good, but would fail any credible CI certification test.

— CI theatre

When the enterprise-wide quarterly or monthly releases were considered best practice, it was necessary to maintain a complete environment for performing testing cycles prior to deployment to production. These **ENTERPRISE-WIDE INTEGRATION TEST ENVIRONMENTS** (often referred to as SIT or Staging) are a common bottleneck for continuous delivery today. The environments themselves are fragile and expensive to maintain, often with components that need manual configuration by a separate environment management team. Testing in the staging environment provides unreliable and slow feedback, and testing effort is duplicated with what can be performed on components in isolation. We recommend that organizations incrementally create an independent path to production for key components. Important techniques include contract testing, decoupling deployment from release, focus on mean time to recovery and testing in production.

Back in the days when SOAP held sway in the enterprise software industry, the practice of generating client code from WSDL specs was an accepted—even encouraged—practice. Unfortunately, the resulting code was often complex, untestable, difficult to modify and frequently didn't work across implementation platforms. With the advent of REST, we found it better to evolve API clients that use the tolerant reader pattern for extracting and processing only the fields needed. Recently we have observed a disturbing return to old habits with developers generating code from API specifications written in Swagger or RAML—a practice that we refer to as **SPEC-BASED CODEGEN**. Although such tools are very useful for driving the design of APIs and for extracting documentation, we caution against the tempting shortcut of simply generating client code directly from these specifications. The chances are that such code will be difficult to test and maintain.

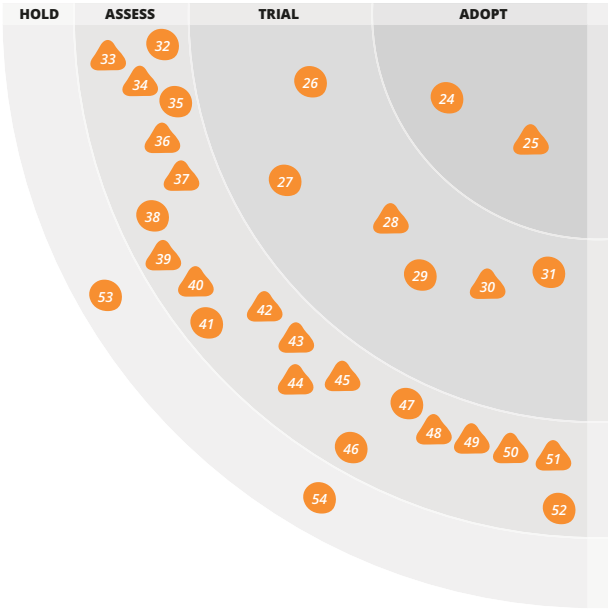
PLATFORMS

ADOPT
24. HSTS
25. Linux Security Modules

TRIAL
26. Apache Mesos
27. Auth0
28. AWS Device Farm *NEW*
29. AWS Lambda
30. OpenTracing *NEW*
31. Unity beyond gaming

ASSESS
32. .NET Core
33. Amazon API Gateway
34. api.ai *NEW*
35. Cassandra carefully
36. Cloud-based image comprehension *NEW*
37. DataStax Enterprise Graph *NEW*
38. Electron
39. Ethereum
40. Hyperledger *NEW*
41. IndiaStack
42. Kafka Streams *NEW*
43. Keycloak *NEW*
44. Mesosphere DCOS
45. Mosquitto *NEW*
46. Nuance Mix
47. OpenVR
48. PlatformIO *NEW*
49. Tango *NEW*
50. Voice platforms *NEW*
51. WebVR *NEW*
52. wit.ai

HOLD
53. CMS as a platform
54. Overambitious API gateways



The Principle of Least Privilege encourages us to restrict software components to access only the resources that they need. By default, however, a Linux process can do anything its running user can do—from binding to arbitrary ports to spawning new shells. The **LINUX SECURITY MODULES** (LSM) framework, which allows for security extensions to be plugged into the kernel, has been used to implement MAC on Linux. SELinux and AppArmor are the predominant and best-known LSM-compatible implementations that ship with the kernel. We recommend that teams learn to use one of these security frameworks (which is why we placed it in the Adopt ring). They 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.

AMAZON API GATEWAY enables developers to expose API services to Internet clients. It offers the usual API gateway features including traffic management, monitoring, authentication and authorization. Our teams have had positive experiences using this service to front AWS Lambda as part of serverless architectures. On the other hand, we have had more challenges using it as a more general purpose gateway to front HTTP/HTTPS endpoints running on EC2—where we have been stymied by a lack of interoperability with VPCs and difficulty in establishing client cert authentication with the gateway. Due to this mixed experience, we would like to advise teams to trial using AWS API Gateway with Lambda but assess suitability when using it in a more general setting.

The huge number of mobile devices makes it almost impossible for companies to test their mobile apps on all of them. Enter **AWS DEVICE FARM**, an app-testing service that enables you to run and interact with your Android, iOS and web apps on a wide variety of physical devices that are hosted in the cloud simultaneously. Detailed logs, performance graphs and screenshots are generated during each run to provide general and device-specific feedback. The service offers a lot of flexibility by allowing the state and configuration of each device to be altered in order to reproduce very specific test scenarios. Our teams are using AWS Device Farm to run end-to-end tests on devices with the largest install base for their apps.

As monolithic applications are being replaced with more complex microservice ecosystems, tracing requests across multiple services is becoming the norm.

— OpenTracing

As monolithic applications are being replaced with more complex (micro)service ecosystems, tracing requests across multiple services is becoming the norm. Luckily **OPENTRACING** is rapidly becoming the de facto standard for distributed tracing. Developed by Uber, Apple, Yelp and various other big players, it supports multiple tracers such as Zipkin, Instana, and Jaeger. OpenTracing currently provides vendor-neutral implementation in six languages: Go, JavaScript, Java, Python, Objective-C and C++.

In parallel with the recent surge of chatbots and voice platforms, we've seen a proliferation of tools and platforms such as **API.AI** that provide a service to extract intent from text and management of conversational flow that you can hook into. Recently acquired by Google, this "natural-language-understanding as a service" offering competes with other players in this space such as wit.ai and Amazon's Lex.

Image comprehension used to be a dark art and required a team of onsite data scientists. In recent years, however, we've come closer to solving problems such as image and facial classification/categorization, facial comparisons, facial landmark identification, and facial recognition. **CLOUD-BASED IMAGE COMPREHENSION** provides access to machine-learning capabilities through services such as Amazon Rekognition, Microsoft Computer Vision API and Google Cloud Vision API which can supplement AR applications and anything involving photo tagging and classification.

We've had some early successes with **DATASTAX ENTERPRISE GRAPH** (DSE Graph) for handling large graph databases. Built on top of Cassandra, DSE Graph targets the type of large data sets where our longtime favorite Neo4j begins to show some limitations. This scale has its trade-offs; for example, you lose the ACID transactions and run-time schema-free nature of Neo4j, but access to the underlying Cassandra tables, the integration of Spark for analytical workloads, and the powerful TinkerPop/Gremlin query language make this an option worth considering.

The hype seems to have peaked for blockchains and cryptocurrencies, as evidenced by the slowdown of previous firehose-scale announcements in this area, and we expect some of the more speculative efforts to die out over time. One of the blockchains, **ETHEREUM**, while not universally popular among diehard blockchain aficionados, appears in increasing numbers in new initiatives. Ethereum is a public blockchain with a built-in programming language allowing developers to build "smart contracts", which 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 proofs 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 $150 million in the ether demonstrates that the blockchains and cryptocurrencies are still the Wild West of the technology world.

HYPERLEDGER is a platform built around blockchain technologies. It consists of a blockchain implementation named Fabric and other associated tools. Disregarding the hype surrounding blockchain, our teams have found it easy to get started with these tools. The fact that it is an open source platform supported by the Linux Foundation also adds to our excitement about Hyperledger.

KAFKA STREAMS is a lightweight library for building streaming applications. It's been designed with the goal of simplifying stream processing enough to make it easily accessible as a mainstream application programming model for asynchronous services. It can

be a good alternative in scenarios where you want to apply a stream processing model to your problem without embracing the complexity of running a cluster (usually introduced by full-fledged stream processing frameworks).

In a microservices or any other distributed architecture, one of the most common needs is to secure the services or APIs through authentication and authorization features. This is where Keycloak comes in. **KEYCLOAK** is an open source identity and access management solution that makes it easy to secure applications or microservices with little to no code. Out of the box, it supports single sign-on, social login, and standard protocols such as OpenID Connect, OAuth2 and SAML.

MESOSPHERE DCOS is a platform built on top of Mesos that abstracts away your underlying infrastructure for containerized applications as well as for applications you don't want to run inside Docker. This may be overkill for more modest deployments, but we're beginning to see successes with both the commercial and open source versions. We particularly like that it facilitates portability between different cloud providers as well as dedicated hardware, and that for containerized workloads you're not tied into one container orchestration framework. Although upgrades can be a little more complex than we would like, the overall stack is stabilizing nicely.

In our experience—for Internet of Things (IoT) solutions where a lot of devices communicate with each other and/or a central data hub—the MQTT connectivity protocol has proven itself. We've also come to like the **MOSQUITTO** MQTT broker. It might not satisfy all demands, particularly with regard to scalability, but its compact nature and easy setup makes it ideal for development and testing purposes.

PLATFORMIO provides a rich ecosystem for IoT development by providing cross-platform builds, library management and good integration with existing IDEs. The intelligent code completion and Smart Code Linter with built-in terminal and serial port monitor greatly enhances the developer experience. It also organizes and maintains thousands of libraries and provides a clean dependency manager with semantic versioning to ease IoT development. We've started using PlatformIO in a few IoT projects and we really like it for its simplicity and wide support of platforms and boards.

Alongside virtual reality (VR), which has a relatively high bar to entry due to hardware requirements and the effort to create virtual worlds, alternate reality (AR) and mixed reality (MR) also entered into the mainstream last year. Pokémon Go provided evidence that regular smartphones are sufficient to create compelling AR/MR experiences. **TANGO** is a new hardware sensor technology for mobile phones that further enhances the possibilities for AR/MR on phones. It allows apps to acquire detailed 3-D measurements of the user's surroundings so that virtual objects can be placed and rendered more convincingly on the camera feed. The first phones with Tango technology are now available.

VOICE PLATFORMS such as Amazon Alexa and Google Home are riding high on the hype cycle; some even herald the ubiquity of the conversational voice interface. We're already integrating conversational UIs into products and seeing the impact of this new interaction in how we design interfaces. Alexa specifically was built from the ground up without a screen and treats the conversational UI as first-class. But it's still too early to believe the hype, and we expect more big players to get in the game.

We're already integrating conversational UIs into products and seeing the impact of this new interaction in how we design interfaces.

— Voice platforms

WEBVR is an experimental JavaScript API that enables you to access VR devices through your browser. It has garnered support from the community and is available through nightly builds as well as in some release versions. If you are looking to build VR experiences in your browser, then this is a great place to start. This technology alongside complementary tools such Three.js, A-Frame, ReactVR, Argon.js and Awe.js brings AR experiences to the browser. The flurry of tools in this space, alongside Internet commission standards, could help promote stronger adoption of AR and VR.

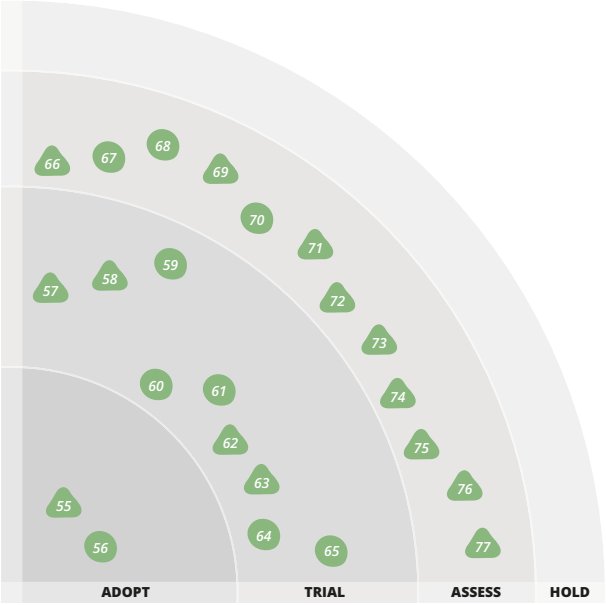
TOOLS

ADOPT
55. fastlane
56. Grafana

TRIAL
57. Airflow *NEW*
58. Cake and Fake *NEW*
59. Galen
60. HashiCorp Vault
61. Pa11y
62. Scikit-learn
63. Serverless Framework *NEW*
64. Talisman
65. Terraform

ASSESS
66. Amazon Rekognition *NEW*
67. Android-x86
68. Bottled Water
69. Claudia *NEW*
70. Clojure.spec
71. InSpec *NEW*
72. Molecule *NEW*
73. Spacemacs *NEW*
74. spaCy *NEW*
75. Spinnaker *NEW*
76. Testinfra *NEW*
77. Yarn *NEW*

HOLD



Web application developers have it easy when it comes to simplifying and automating diverse application workflows; they can choose from a variety of solutions to help automate release processes. When developing for mobile, however, we're dealing with two operating systems with two different ways of building, testing, distribution, generating screenshots, signing and distributing applications. To help ease the pain, our teams have adopted **FASTLANE** as the go-to tool to automate the release process for iOS and Android applications. Using simple configurations and multiple pipelines, they can achieve continuous delivery for mobile development.

AIRFLOW is a tool to programmatically create, schedule and monitor data pipelines. By treating Directed Acyclic Graphs (DAGs) as code, it encourages maintainable, versionable and testable data pipelines. We've leveraged this configuration in our projects to create dynamic pipelines that resulted in lean and explicit data workflows. Airflow makes it easy to define your operators and executors and to extend the library so that it fits the level of abstraction that suits your environment.

Our teams have adopted fastlane as the go-to tool to automate the release process for iOS and Android applications.

— fastlane

MSBuild has been the primary build system in the .NET ecosystem since its introduction in 2005; however, it suffers from many of the same weaknesses we've previously called out in Maven. The .NET community has started to develop alternatives to MSBuild which are easier to maintain and more flexible, and evolve more fluidly as a project grows. Two of these alternatives are **CAKE AND FAKE**. Cake uses a DSL built in C#, while

Fake uses F#. Each has seen significant growth over the last year and has proven to be a viable alternative to MSBuild for orchestrating common build tasks in .NET projects.

The .NET community has started to develop alternatives to MSBuild which are easier to maintain and more flexible, and evolve more fluidly as a project grows.

— Cake and Fake

SCIKIT-LEARN is not a new tool (it is approaching its tenth birthday); what is new is the rate of adoption of machine-learning tools and techniques outside of academia and major tech companies. Providing a robust set of models and a rich set of functionality, Scikit-learn plays an important role in making machine-learning concepts and capabilities more accessible to a broader (and often non-expert) audience.

The popular **SERVERLESS FRAMEWORK** provides tooling for scaffolding and deployment of serverless applications, primarily using AWS Lambda and other AWS offerings. Serverless Framework provides template support for JavaScript, Python, Java and C#, and has an active community that contributes plugins that extend the framework. The framework also supports the Apache incubator project OpenWhisk as an alternative to AWS Lambda.

AMAZON REKOGNITION is one of the cloud-based image comprehension tools we've mentioned elsewhere in this Radar. What we like about it is that Amazon has taken a somewhat novel approach to making faces anonymous (using GUIDs) from AWS to accommodate some of the privacy concerns that come with facial recognition.

The combination of AWS Lambda with Amazon API Gateway has had a big impact on how we deploy services and APIs. However, even in this serverless configuration, the amount of configuration required to wire things together is not trivial. **CLAUDIA** is a tool which automates deployment of AWS Lambda functions written in JavaScript and associated API Gateway configurations. It provides reasonable defaults, and our teams have found it allows them to get started quickly with Lambda-based microservices.

How does a business hand autonomy to delivery teams while still making sure their deployed solutions are safe and compliant? How do you ensure that servers, once deployed, remain secure and compliant over their operational lifetime? These are the problems that InSpec tries to address. **INSPEC** is an infrastructure testing tool inspired by Serverspec, but with modifications that make the tool more useful for security professionals who need to ensure compliance across thousands of servers. Individual tests can be combined into complete security profiles and run remotely from a command line. InSpec is useful for developers but extends to testing deployed production infrastructure continuously, moving toward QA in production.

MOLECULE is designed to aid in the development and testing of Ansible roles. By building the scaffolding for running Ansible role tests on a virtual machine or container of choice, we don't have to setup our testing environment manually. Molecule leverages Vagrant, Docker, and OpenStack to manage virtual machines or containers, and supports Serverspec, Testinfra, or Goss to run the tests. The default steps in the sequence facility model include: virtual machine management, Ansible linting, idempotence testing and convergence testing. Although it is a fairly young project, we see a great potential for its usage.

As any Emacs fan will tell you, Emacs is more than a text editor; it is a platform for character-mapped applications. Over the past few years, there has been an explosion of new developments on this platform, but we think **SPACEMACS** deserves particular attention. Spacemacs provides an introduction to the Emacs platform, with a new keyboard user-interface, simplified customization layers, and a curated distribution of Emacs packages. One of the project's aims is to be the best of worlds by combining the Vim UI with the internal reprogrammability of Emacs. We consider developer productivity tools to be a vital part of effective software

development, and if you haven't considered Emacs for a while, we suggest you take a look at how Spacemacs rethinks this classic development platform.

SPACY is a Natural Language Processing (NLP) library written in Python. It is a high-performance library, intended for use by developers in production, and applies NLP models suited for processing text that often mixes in emoticons and inconsistent punctuation marks. Unlike other NLP frameworks, spaCy is a pluggable library and not a platform; it is aimed at production applications rather than model training for research. It plays well with TensorFlow and the rest of the Python AI ecosystem. We've used spaCy in the enterprise context to build a search engine that takes human language queries and helps users make business decisions.

Netflix has open sourced **SPINNAKER**, its microservices continuous delivery (CD) platform. Compared to other CI/CD platforms, Spinnaker implements cluster management and deployment of baked images to the cloud as first-class features. It supports out-of-the-box deployment and cluster management for multiple cloud providers such as Google Cloud Platform, AWS and Pivotal Cloud Foundry. You can integrate Spinnaker with Jenkins to run a Jenkins job build. We like Spinnaker's opinionated approach for deploying microservices to the cloud—with the exception that Spinnaker's pipelines are created via a user interface (UI) and cannot be configured as code.

Given the wide use of infrastructure tools today, it should come as no surprise that infrastructure as code has increased in current projects. With this tendency comes the need for testing this code. With **TESTINFRA** you can test the actual state of your servers configured manually or by tools such as Ansible, Puppet and Docker. Testinfra aims to be a Serverspec equivalent in Python and is written as a plugin to the Pytest test engine.

Given the wide use of infrastructure tools today, it should come as no surprise that infrastructure as code has increased in current projects.

— Testinfra

YARN is a new package manager that replaces the existing workflow for the npm client while remaining compatible with the npm registry. With the npm client, we may end up with a different tree structure under node_modules based on the order that dependencies are installed. This nondeterministic nature can cause "works on my machine" problems. By breaking the installation steps into resolution, fetching and linking, Yarn avoids these issues using deterministic algorithms and lockfiles and thus guarantees repeatable installations. We've also seen significantly faster builds in our continuous integration (CI) environment because of Yarn caching all the packages it downloads.

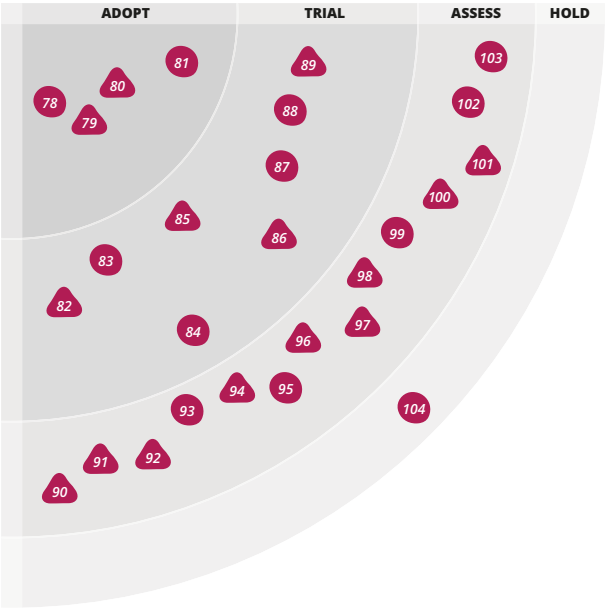
LANGUAGES & FRAMEWORKS

ADOPT
78. Ember.js
79. Python 3
80. ReactiveX
81. Redux

TRIAL
82. Avro *NEW*
83. Elixir
84. Enzyme
85. Hangfire *NEW*
86. Nightwatch *NEW*
87. Phoenix
88. Quick and Nimble
89. Vue.js

ASSESS
90. Angular 2 *NEW*
91. Caffe *NEW*
92. DeepLearning.scala *NEW*
93. ECMAScript 2017
94. Instana *NEW*
95. JuMP
96. Keras *NEW*
97. Knet.jl *NEW*
98. Kotlin *NEW*
99. Physical Web
100. PostCSS *NEW*
101. Spring Cloud *NEW*
102. Three.js
103. WebRTC

HOLD
104. AngularJS



PYTHON 3 introduced many useful features that are not backward compatible with Python 2.x. It also removed numerous Python 2.x features that were maintained for backward compatibility, making Python 3 easier to learn and use and more consistent with the rest of the language. Our experience using Python 3 in domains such as machine learning and web application development shows that both the language and most of its supporting libraries have matured for adoption. We were able to fork and patch minor issues of existing libraries or avoided using incompatible Python 2.x libraries that had been abandoned. If you are developing in Python we strongly encourage you to use Python 3.

Distributed systems often utilize multithreading, event-based communication and nonblocking I/O to improve the overall system efficiency. These programming techniques impose challenges such as low-level threading, synchronization, thread safety, concurrent data structures, and non-blocking I/O. The open source **REACTIVEX** library beautifully abstracts away these concerns, provides the required application plumbing, and extends the observable pattern on streams of asynchronous events. ReactiveX also has an active developer community and supports a growing list of languages, the most recent addition being RxSwift. It also implements binding to mobile and desktop platforms.

The open source REACTIVEX library beautifully abstracts away these concerns—low-level threading, synchronization, thread safety, concurrent data structures, and non-blocking I/O—to provide the required application plumbing, and extends the observable pattern on streams of asynchronous events.

— ReactiveX

AVRO is a framework for data serialization. By storing schema along with the message content, it encourages schema evolution. Producers can edit field names, add new fields or delete existing fields and Avro guarantees

that the clients continue to consume the messages. Having a schema allows data to be written without overhead which results in compact data encoding and faster data processing. Although the exchange of structure-less messages between producer and consumer is flexible, we've seen teams facing issues with incompatible unprocessed messages in the queue during deployments. We've used Avro in a number of projects and would recommend using it over just sending unstructured messages.

Hangfire is both easy to use and flexible, and it embraces a functional style. Particularly interesting is its ability to save a task's state so it can resume when an application restarts after a crash or shutdown.

— Hangfire

One common problem in application development is how to schedule tasks that run outside the main process periodically or when certain conditions are met. The problem gets more complicated when unexpected events, such as application shutdowns, occur. The **HANGFIRE** framework, as our teams discovered, can do this and much more in the .NET environment. Hangfire is both easy to use and flexible, and it embraces a functional style. Particularly interesting is its ability to save a task's state so it can resume when an application restarts after a crash or shutdown.

NIGHTWATCH is a framework that allows automated acceptance tests for browser-based apps to be created in JavaScript and run in Node.js. Nightwatch allows tests to be defined using a fluent API which can then be executed against a Selenium/WebDriver server. In the case of single page apps or other JavaScript-heavy pages, this allows the automated tests to be created and run within the same language and environment as the bulk of the code.

In the ever-changing world of front-end JavaScript frameworks, one of the emerging favorites appears to be **VUE.JS**. Vue.js is 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 such as modularity, components and reactive data flow. It has a low learning curve, which makes it interesting for less experienced developers and beginners. Note, though, that Vue.js 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.

In the previous Radar, we moved AngularJS into the Hold ring (where it remains in this edition). When it comes to **ANGULAR 2**, we're seeing mixed messages. Over the past year some teams at ThoughtWorks have used Angular 2 successfully and consider it a solid choice. However, Angular 2 is a rewrite, not an evolution, of AngularJS, and switching from AngularJS to Angular 2 is not much different than switching from AngularJS to another framework. Given the, in our experience, superior contenders such as React.js, Ember.js and Vue.js, we're still hesitant to give Angular 2 a strong recommendation. We do want to highlight, though, that it is not a bad choice, especially if you bought into TypeScript.

CAFFE is an open source library for deep learning created by the Berkeley Vision and Learning Center. It mostly focusses on convolutional networks for computer vision applications. Caffe is a solid and popular choice for computer vision-related tasks and you can download many successful models made by Caffe users from the Caffe Model Zoo for out-of-the-box use. Like Keras, Caffe is a Python-based API. In Keras, however, models and components are objects created directly in Python code, whereas Caffe models are described by Protobuf configuration files. Either approach has its pros and cons, and converting between the two is also possible.

DEEPLEARNING.SCALA is an open source deep-learning toolkit in Scala created by our colleagues at ThoughtWorks. We're excited about this project because it uses differentiable functional programming to create and compose neural networks; a developer simply writes code in Scala with static typing. DeepLearning.scala currently supports basic types such as float, double, GPU-accelerated N-dimensional arrays as well as algebraic data types. We're looking forward to future releases of the toolkit which are said to support higher order functions and distributed training on Spark.

INSTANA is yet another entrant into the crowded application performance management space. The fact that it's built from the ground up for cloud native architectures differentiates Instana from many of its

competitors. Features include dynamic discovery, distributed tracing and service health plus the ability to "time shift" your view of your infrastructure to the moment an incident occurred. It remains to be seen whether this product can gain traction over the combination of open source projects—such as Consul, Prometheus and the implementations of OpenTracing—that do the same thing; however it's worth taking a look if you need an out-of-the-box solution.

KERAS is a high-level interface in Python for building neural networks. Created by a Google engineer, Keras is open source and runs on top of either TensorFlow or Theano. It provides an amazingly simple interface for creating powerful deep-learning algorithms to train on CPUs or GPUs. Keras is well designed with modularity, simplicity, and extensibility in mind. Unlike a library such as Caffe, Keras supports more general network architectures such as recurrent nets, making it overall more useful for text analysis, NLP and general machine learning. If computer vision, or any other specialized branch of machine learning, is your primary concern, Caffe may be a more appropriate choice. However, if you're looking to learn a simple yet powerful framework, Keras should be your first choice.

KNET.JL is the Koç University deep-learning framework implemented in Julia by Deniz Yuret and collaborators. Unlike gradient-generating compilers such as Theano and TensorFlow which force users into a restricted mini-language, Knet allows the definition and training of machine-learning models using the full power and expressiveness of Julia. Knet uses dynamic computational graphs generated at runtime for the automatic differentiation of almost any Julia code. We really like the support of GPU operations through the KnetArray type, and in case you don't have access to a GPU machine, the team behind Knet also maintains a preconfigured Amazon Machine Image (AMI) so you can evaluate it in the cloud.

The **KOTLIN** programming language is on many of our developers' bucket lists to assess this year, and some have already used it successfully in production. It is an open source JVM language from JetBrains. Our Swift mobile developers like it as it is syntactically closer to Swift and equally concise. Our Java developers have enjoyed its seamless interoperability with the Java language and tools and found it easier to learn than Scala. Kotlin supports functional programming concepts but with less features than Scala. Developers on our teams who like static typing with the compiler catching null pointer defects found themselves writing fewer boilerplate tests.

Teams building systems composed of microservices need to think about coordination techniques such as service discovery, load balancing, circuit breaking and health checking.

— Spring Cloud

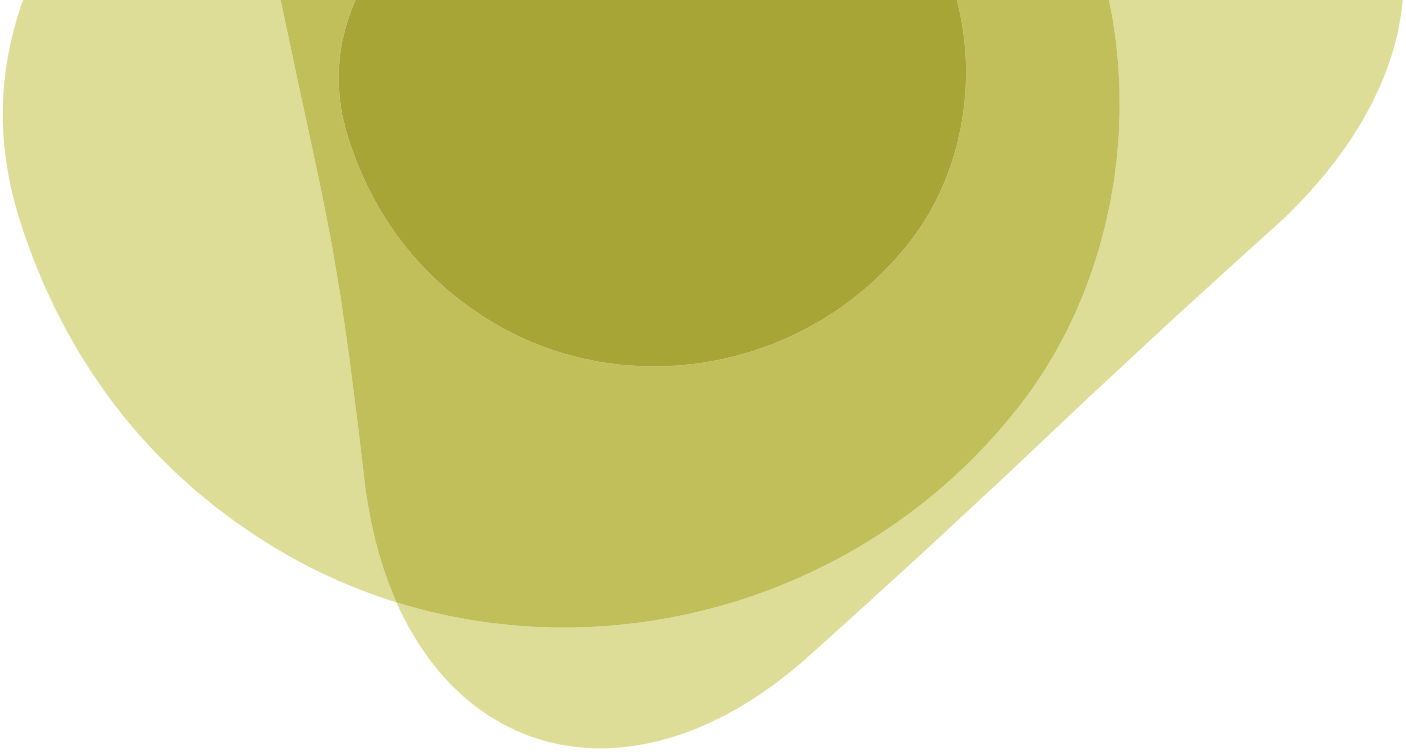
POSTCSS is a Node.js-based JavaScript framework for operating on an abstract syntax tree-based representation of CSS documents with a rich ecosystem of plugins. Often incorrectly thought of as a preprocessor (such as SaaS or Less), we find that the real power of PostCSS comes from the number of things that can be done with the rich set of plugins which includes linting (the stylelint plugin), cross-compilation (the sugarss plugin), name-mangling to avoid selector collision (the modules plugin), boilerplate CSS code generation (the autoprefixer plugin), minification and many others. The different maturity levels of the plugins notwithstanding, PostCSS itself remains a simple and powerful framework for treating CSS like a full-fledged language for front-end development.

Teams building systems composed of microservices need to think about coordination techniques such as service discovery, load balancing, circuit breaking and health checking. Many of these techniques require teams to set up tooling, which is not always trivial. The **SPRING CLOUD** project provides tools for developers so they can use these coordination techniques in the familiar Spring environment. These tools support Consul, ZooKeeper and the Netflix OSS full stack, all tools that we like. Simply put, it makes it easy to do the right thing with these tool sets. Although our usual concerns with Spring still stand, namely that it hides too much of the complexity, you should consider Spring Cloud if you are in the ecosystem and need to solve these problems.

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

SUBSCRIBE NOW

thght.works/Sub-EN



ThoughtWorks®

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

Founded over 20 years ago, ThoughtWorks has grown to a company of over 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.com