TECHNOLOGY RADAR *VOL.17*

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) | Bharani Subramaniam | Camilla Crispim | Erik Doernenburg
Evan Bottcher | Fausto de la Torre | Hao Xu | Ian Cartwright | James Lewis
Jonny LeRoy | Ketan Padegaonkar | Lakshminarasimhan Sudarshan | Marco Valtas | Mike Mason
Neal Ford | Rachel Laycock | Scott Shaw | Shangqi Liu | Zhamak Dehghani

WHAT'S NEW?

Highlighted themes in this edition:

OPEN SOURCE ON THE RISE IN CHINA

The tide is rising. Because of changes in both attitude and policy, large Chinese companies such as Alibaba and Baidu are rapidly releasing open source frameworks, tools and platforms. The growth of their software ecosystems is accelerating fast as they expand economically.

Given the number of software projects in the booming and massive Chinese markets, the number and quality of open source projects appearing on GitHub and other open source sites is expected to rise. Why would Chinese companies open source so many assets? As is the case with hot software markets such as Silicon Valley, competition for developers is tight and offering higher compensation only goes so far.

The prospect of working on cutting-edge open source projects with other smart developers is a universal incentive. We expect major open source innovations to continue the trend of README files written in Chinese first and English second.

KUBERNETES, THE CHOICE FOR CONTAINER ORCHESTRATOR

A large number of Radar entries revolved around Kubernetes and its increasingly dominant presence on many projects. It seems that the software development ecosystem is settling on Kubernetes and related tools to solve the common problems related to deployment, scaling and operating containers.

Radar entries such as GKE, Kops, and Sonobuoy introduce managed platform services and tooling that improve the overall experience of adopting and running Kubernetes. Indeed, its capability to simply run multiple containers as one unit of scheduling enables service mesh and sidecar for endpoint security.

Kubernetes has become the default operating system for containers: many cloud providers have taken advantage of its open and modular architecture to adopt and run Kubernetes, while tools leverage its open APIs to access abstractions such as workloads, clusters, configuration, and storage.

We see more products utilizing Kubernetes as an ecosystem, making it the next level of abstraction after microservices and containers. This is further evidence that developers can successfully leverage modern architectural styles despite the inherent complexities of distributed systems.

CLOUD AS THE NEW NORMAL

The other pervasive topic of conversation, while pulling together this edition, had a distinctly "cloudy" nature. As cloud providers become more capable and reach parity of features, the public cloud model is becoming the new default for many organizations.

Instead of asking, "Why in the cloud?", many companies now ask, "Why *not* in the cloud?" when embarking on new projects. Certainly, some types of software still demand on-premises systems, but as prices drop and capabilities expand, cloud-native development becomes increasingly viable.

Even though basic feature parity is a given among the major cloud solution providers, they each also provide unique offerings to differentiate themselves for specific types of solutions. Thus, we see companies taking advantage of several different providers via Polycloud, choosing the specialized capabilities of those platforms that best suit their customers' needs.

TRUST IN BLOCKCHAIN MORE EVENLY DISTRIBUTED

Despite the chaos surrounding cryptocurrencies in the markets, many of our clients are finding ways to leverage blockchain solutions for distributed ledgers and smart contracts. Several Radar entries show maturity in the use of blockchain-related technologies, providing increasingly interesting ways to implement smart contracts, with a variety of techniques and programming languages.

Blockchains solve the age-old problem of distributed trust and shared, indelible ledgers. Today, companies are increasing their users' confidence in the underlying mechanics of blockchain implementations. Many industries have distinct distributed trust problems; we expect blockchain solutions to continue to find ways to solve them.

© ThoughtWorks, Inc. All Rights Reserved.

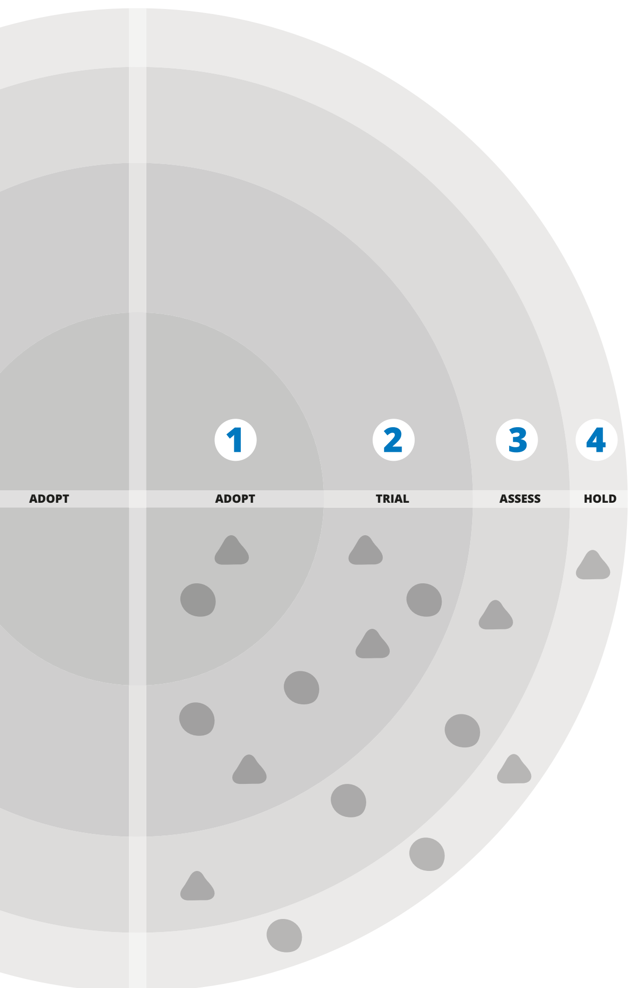
ABOUT THE RADAR

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

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

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

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



RADAR AT A GLANCE

1 ADOPT

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

2 TRIAL

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

3 ASSESS

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

4 HOLD

Proceed with caution.

NEW OR CHANGED

NO CHANGE

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

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

© ThoughtWorks, Inc. All Rights Reserved.

THE RADAR

TECHNIQUES

ADOPT
1. Lightweight Architecture Decision Records

TRIAL
2. Applying product management to internal platforms *NEW*
3. Architectural fitness function *NEW*
4. Autonomous bubble pattern *NEW*
5. Chaos Engineering *NEW*
6. Decoupling secret management from source code
7. DesignOps *NEW*
8. Legacy in a box
9. Micro frontends
10. Pipelines for infrastructure as code *NEW*
11. Serverless architecture
12. TDD'ing containers *NEW*

ASSESS
13. Algorithmic IT operations *NEW*
14. Ethereum for decentralized applications *NEW*
15. Event streaming as the source of truth *NEW*
16. Platform engineering product teams
17. Polycloud *NEW*
18. Service mesh *NEW*
19. Sidecars for endpoint security *NEW*
20. The three Rs of security *NEW*

HOLD
21. A single CI instance for all teams
22. CI theatre
23. Enterprise-wide integration test environments
24. Recreating ESB antipatterns with Kafka *NEW*
25. Spec-based codegen

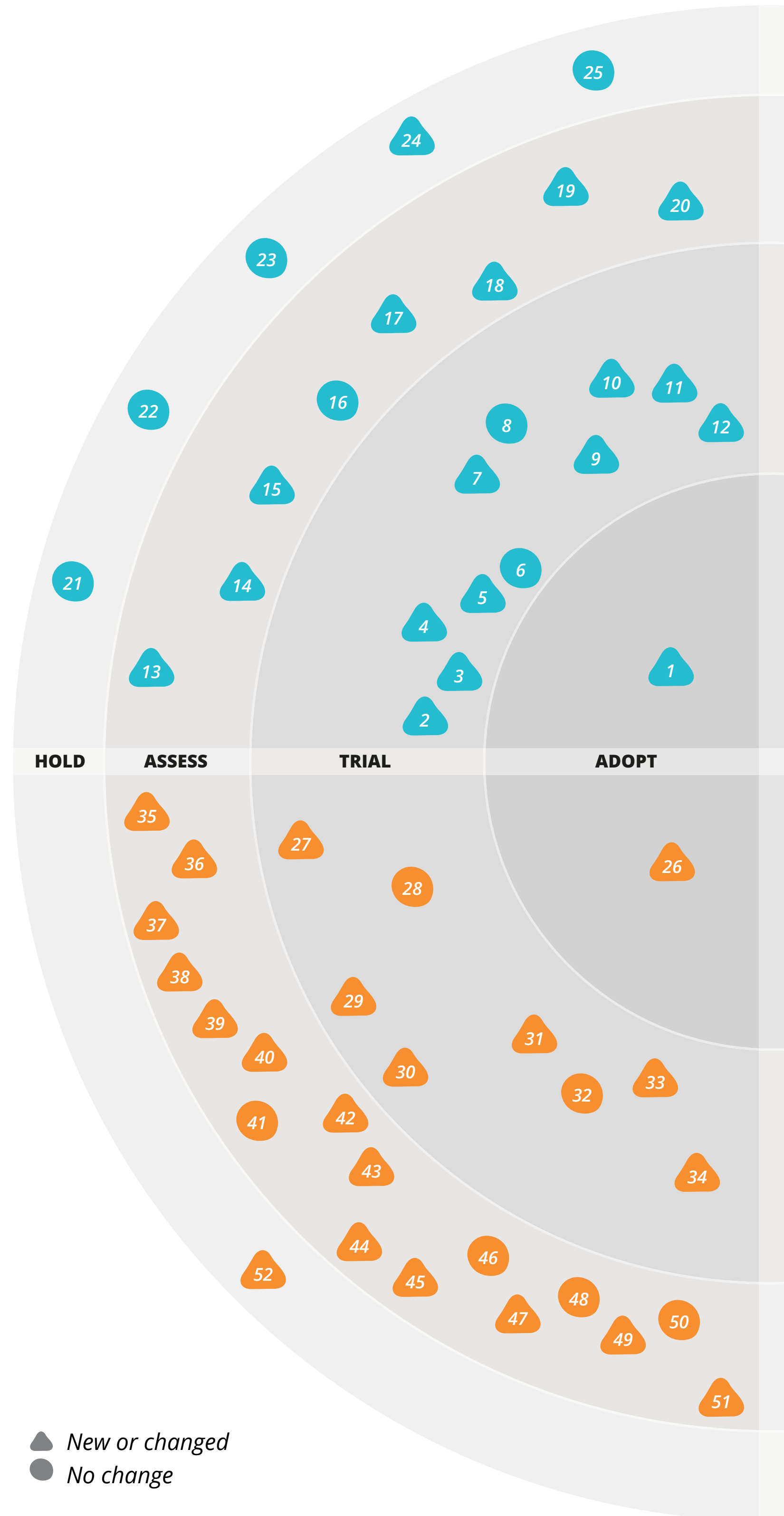
PLATFORMS

ADOPT
26. Kubernetes

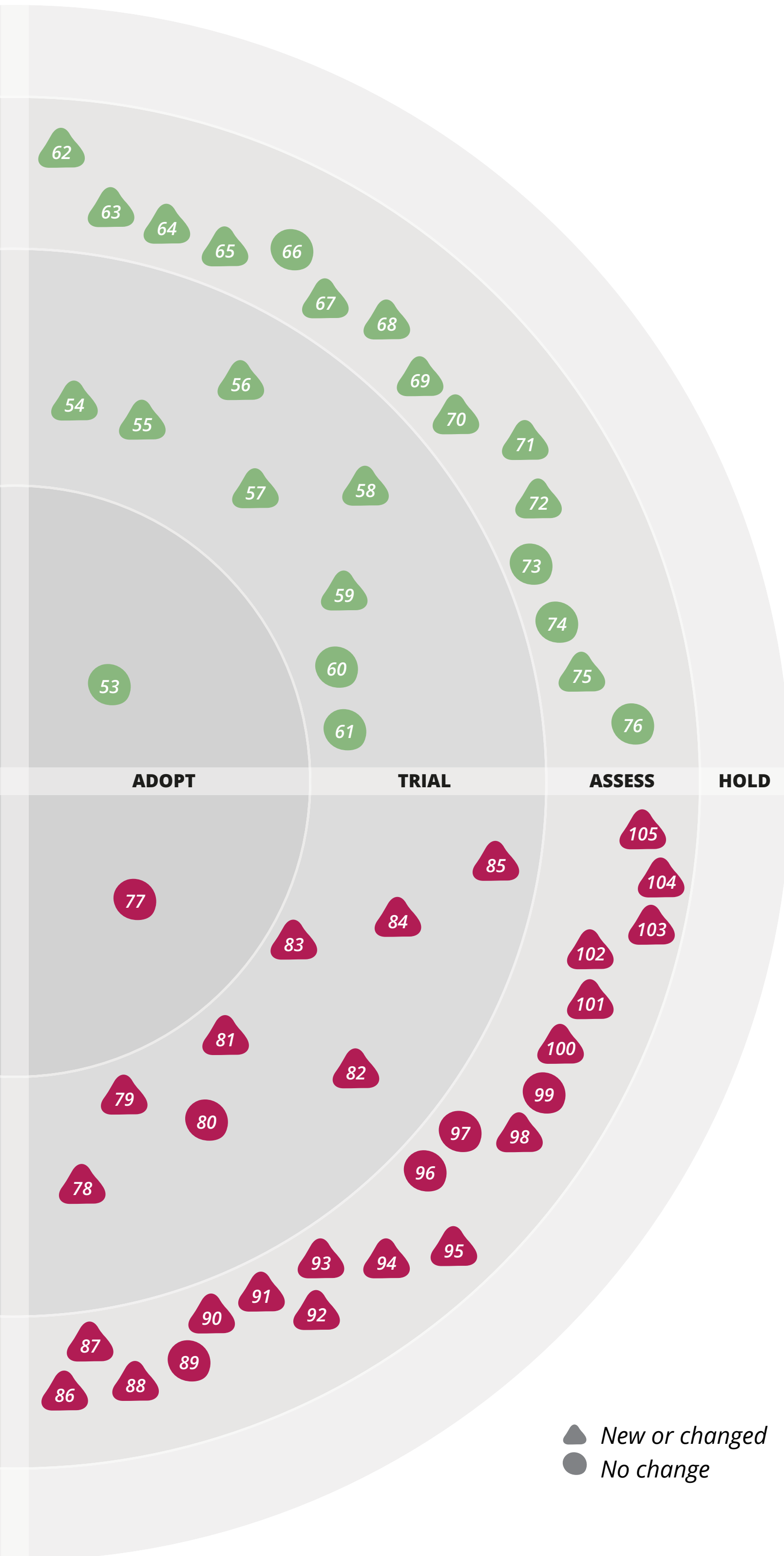
TRIAL
27. .NET Core
28. AWS Device Farm
29. Flood IO *NEW*
30. Google Cloud Platform *NEW*
31. Keycloak
32. OpenTracing
33. Unity beyond gaming
34. WeChat *NEW*

ASSESS
35. Azure Service Fabric *NEW*
36. Cloud Spanner *NEW*
37. Corda *NEW*
38. Cosmos DB *NEW*
39. DialogFlow
40. GKE *NEW*
41. Hyperledger
42. Kafka Streams
43. Language Server Protocol *NEW*
44. LoRaWAN *NEW*
45. MapD *NEW*
46. Mosquitto
47. Netlify *NEW*
48. PlatformIO
49. TensorFlow Serving *NEW*
50. Voice platforms
51. Windows Containers *NEW*

HOLD
52. Overambitious API gateways



© ThoughtWorks, Inc. All Rights Reserved.



THE RADAR

TOOLS

ADOPT
53. fastlane

TRIAL
54. Buildkite *NEW*
55. CircleCI *NEW*
56. gopass *NEW*
57. Headless Chrome for front-end test *NEW*
58. jsoniter *NEW*
59. Prometheus
60. Scikit-learn
61. Serverless Framework

ASSESS
62. Apex *NEW*
63. assertj-swagger *NEW*
64. Cypress *NEW*
65. Flow *NEW*
66. InSpec
67. Jupyter *NEW*
68. Kong API Gateway *NEW*
69. kops *NEW*
70. Lighthouse *NEW*
71. Rendertron *NEW*
72. Sonobuoy *NEW*
73. spaCy
74. Spinnaker
75. Spring Cloud Contract *NEW*
76. Yarn

HOLD

LANGUAGES & FRAMEWORKS

ADOPT
77. Python 3

TRIAL
78. Angular
79. AssertJ *NEW*
80. Avro
81. CSS Grid Layout *NEW*
82. CSS Modules *NEW*
83. Jest *NEW*
84. Kotlin
85. Spring Cloud

ASSESS
86. Android Architecture Components *NEW*
87. ARKit/ARCore *NEW*
88. Atlas and BeeHive *NEW*
89. Caffe
90. Clara rules *NEW*
91. CSS-in-JS *NEW*
92. Digdag *NEW*
93. Druid *NEW*
94. ECharts *NEW*
95. Gobot *NEW*
96. Instana
97. Keras
98. LeakCanary *NEW*
99. PostCSS
100. PyTorch *NEW*
101. single-spa *NEW*
102. Solidity *NEW*
103. TensorFlow Mobile *NEW*
104. Truffle *NEW*
105. Weex *NEW*

HOLD

© ThoughtWorks, Inc. All Rights Reserved.

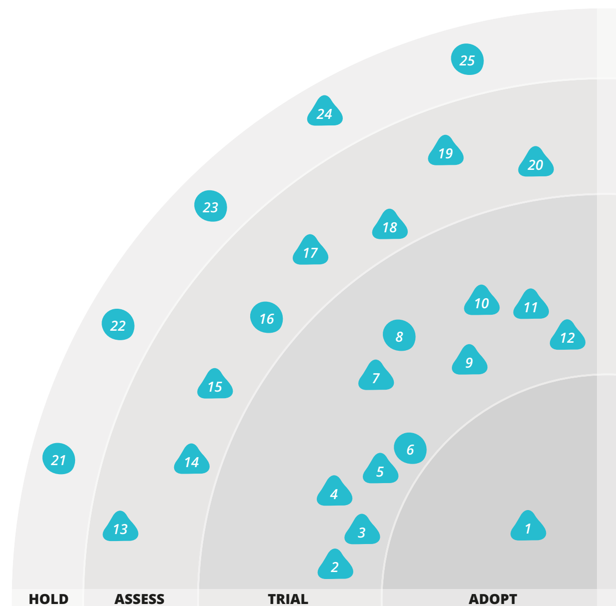
TECHNIQUES

ADOPT
1. Lightweight Architecture Decision Records

TRIAL
2. Applying product management to internal platforms *NEW*
3. Architectural fitness function *NEW*
4. Autonomous bubble pattern *NEW*
5. Chaos Engineering *NEW*
6. Decoupling secret management from source code
7. DesignOps *NEW*
8. Legacy in a box
9. Micro frontends
10. Pipelines for infrastructure as code *NEW*
11. Serverless architecture
12. TDD'ing containers *NEW*

ASSESS
13. Algorithmic IT operations *NEW*
14. Ethereum for decentralized applications *NEW*
15. Event streaming as the source of truth *NEW*
16. Platform engineering product teams
17. Polycloud *NEW*
18. Service mesh *NEW*
19. Sidecars for endpoint security *NEW*
20. The three Rs of security *NEW*

HOLD
21. A single CI instance for all teams
22. CI theatre
23. Enterprise-wide integration test environments
24. Recreating ESB antipatterns with Kafka *NEW*
25. Spec-based codegen



Much documentation can be replaced with highly readable code and tests. In a world of evolutionary architecture, however, it's important to record certain design decisions for the benefit of future team members as well as for external oversight. **LIGHTWEIGHT ARCHITECTURE DECISION RECORDS** is a technique for capturing important architectural decisions along with their context and consequences. We recommend storing these details in source control, instead of a wiki or website, as then they can provide a record that remains in sync with the code itself. For most projects, we see no reason why you wouldn't want to use this technique.

We've seen a steep increase in interest in the topic of digital platforms over the past 12 months. Companies looking to roll out new digital solutions quickly and efficiently are building internal platforms, which offer teams self-service access to the business APIs, tools, knowledge and support necessary to build and operate their own solutions. We find that these platforms are most effective when they're given the same respect as an external product offering. **APPLYING PRODUCT MANAGEMENT TO INTERNAL PLATFORMS** means establishing empathy with internal consumers (read: developers) and collaborating with them on the design. Platform product managers establish roadmaps and ensure the platform delivers value to the business and enhances the developer experience. Some owners even create a brand identity for the internal platform and use that to market the benefits to their colleagues. Platform product managers look after the quality of the platform, gather usage metrics, and continuously improve it over time. Treating the platform as a product helps to create a thriving ecosystem and avoids the pitfall of building yet another stagnant, underutilized service-oriented architecture.

Borrowed from evolutionary computing, a fitness function is used to summarize how close a given design solution is to achieving the set aims.

(Architectural fitness function)

© ThoughtWorks, Inc. All Rights Reserved.

Borrowed from evolutionary computing, a fitness function is used to summarize how close a given design solution is to achieving the set aims. When defining an evolutionary algorithm, the designer seeks a 'better' algorithm; the fitness function defines what 'better' means in this context. An **ARCHITECTURAL FITNESS FUNCTION**, as defined in Building Evolutionary Architectures, provides an objective integrity assessment of some architectural characteristics, which may encompass existing verification criteria, such as unit testing, metrics, monitors, and so on. We believe architects can communicate, validate and preserve architectural characteristics in an automated, continual manner, which is the key to building evolutionary architectures.

CI and CD tools can be used to test server configuration, server image building, environment provisioning, and integration of environments.

(Pipelines for infrastructure as code)

Many organizations we work with are trying hard to use modern engineering approaches to build new capabilities and features, while also having to coexist with a long tail of legacy systems. An old strategy that, based on our experience, has turned out to be increasingly helpful in these scenarios is Eric Evans's **AUTONOMOUS BUBBLE PATTERN**. This approach involves creating a fresh context for new application development that is shielded from the entanglements of the legacy world. This is a step beyond just using an anticorruption layer. It gives the new bubble context full control over its backing data, which is then asynchronously kept up-to-date with the legacy systems. It requires some work to protect the boundaries of the bubble and keep both worlds consistent, but the resulting autonomy and reduction in development friction is a first bold step toward a modernized future architecture.

In previous editions of the Radar, we've talked about using Chaos Monkey from Netflix to test how a running system is able to cope with outages in production by randomly disabling instances and measuring the results. **CHAOS ENGINEERING** is the nascent term for the wider application of this technique. By running experiments on distributed systems in production, we're able to build confidence that those systems work as expected under turbulent conditions. A good place to start understanding this technique is the Principles of Chaos Engineering website.

Inspired by the DevOps movement, **DESIGNOPS** is a cultural shift and a set of practices that allows people across an organization to continuously redesign products without compromising quality, service coherency or team autonomy. DesignOps advocates for the creation and evolution of a design infrastructure that minimizes the effort necessary to create new UI concepts and variations, and to establish a rapid and reliable feedback loop with end users. With tools such as Storybook promoting close collaboration, the need for upfront analysis and specification handoffs is reduced to the absolute minimum. With DesignOps, design is shifting from being a specific practice to being a part of everyone's job.

We've seen significant benefits from introducing microservices architectures, which have allowed teams to scale the delivery of independently deployed and maintained services. Unfortunately, we've also seen many teams create front-end monoliths — a single, large and sprawling browser application — on top of their back-end services. Our preferred (and proven) approach is to split the browser-based code into **MICRO FRONTENDS**. In this approach, the web application is broken down into its features, and each feature is owned, frontend to backend, by a different team. This ensures that every feature is developed, tested and deployed independently from other features. Multiple techniques exist to recombine the features — sometimes as pages, sometimes as components — into a cohesive user experience.

The use of continuous delivery pipelines to orchestrate the release process for software has become a mainstream concept. However, automatically testing changes to infrastructure code isn't as widely understood. Continuous integration (CI) and continuous delivery (CD) tools can be used to test server configuration (e.g., Chef cookbooks, Puppet modules, Ansible playbooks), server image building (e.g., Packer), environment provisioning (e.g., Terraform, CloudFormation) and integration of environments. The use of **PIPELINES FOR INFRASTRUCTURE AS CODE** enables errors to be found before changes are applied to operational environments — including environments used for development and testing. They also offer a way to ensure that infrastructure tooling is run consistently,

© ThoughtWorks, Inc. All Rights Reserved.

from CI/CD agents, as opposed to being run from individual workstations. Some challenges remain, however, such as the longer feedback loops associated with standing up containers and virtual machines. Still, we've found this to be a valuable technique.

The use of **SERVERLESS ARCHITECTURE** has very quickly become an accepted approach for organizations deploying cloud applications, with a plethora of choices available for deployment. Even traditionally conservative organizations are making partial use of some serverless technologies. Most of the discussion goes to Functions as a Service (e.g., AWS Lambda, Google Cloud Functions, Azure Functions) while the appropriate patterns for use are still emerging. Deploying serverless functions undeniably removes the nontrivial effort that traditionally goes into server and OS configuration and orchestration. Serverless functions, however, are not a fit for every requirement. At this stage, you must be prepared to fall back to deploying containers or even server instances for specific requirements. Meanwhile, the other components of a serverless architecture, such as Backend as a Service, have become almost a default choice.

Many development teams have adopted test-driven development practices for writing application code because of their benefits. Others have turned to containers to package and deploy their software, and it's accepted practice to use automated scripts to build the containers. What we've seen few teams do so far is combine the two trends and drive the writing of the container scripts using tests. With frameworks such as Serverspec and Goss, you can express the intended functionality for either isolated or orchestrated containers, with short feedback loops. This means that it's possible to use the same principles we've championed for code by **TDD'ING CONTAINERS**. Our initial experience doing so has been very positive.

The amount of data collected by IT operations has been increasing for years. For example, the trend toward microservices means that more applications are generating their own operational data, and tools such as Splunk, Prometheus, or the ELK stack make it easier to store and process data later on, to gain operational insights. When combined with increasingly democratized machine learning tools, it's inevitable that operators will start to incorporate statistical models and trained classification algorithms into their toolsets. Although these algorithms have been available for years, and various attempts have been made to automate service management, we're only just starting to understand how machines and humans can collaborate to identify outages earlier or pinpoint the source of failures. Although there is a risk of overhyping **ALGORITHMIC IT OPERATIONS**, steady improvement in machine learning algorithms will inevitably change the role of humans in operating tomorrow's data centers.

Blockchains have been widely hyped as the panacea for all things fintech, from banking to digital currency to supply chain transparency. We've previously featured Ethereum because of its feature set, which includes smart contracts. Now, we're seeing more development using **ETHEREUM FOR DECENTRALIZED APPLICATIONS** in other areas. Although this is still a very young technology, we're encouraged to see it being used to build decentralized applications beyond cryptocurrency and banking.

Blockchains have been widely hyped as the panacea for everything from banking, to digital currency, to supply chain transparency.
(Ethereum for decentralized applications)

As event streaming platforms, such as Apache Kafka, rise in popularity, many consider them as an advanced form of message queuing, used solely to transmit events. Even when used in this way, event streaming has its benefits over traditional message queuing. However, we're more interested in how people use **EVENT STREAMING AS THE SOURCE OF TRUTH** with platforms (Kafka in particular) as the primary store for data as immutable events. A service with an Event Sourcing design, for example, can use Kafka as its event store; those events are then available for other services to consume. This technique has the potential to reduce duplicating efforts between local persistence and integration.

The major cloud providers (Amazon, Microsoft and Google) are locked in an aggressive race to maintain parity on core capabilities while their products are differentiated only marginally. This is causing a few organizations to adopt a **POLYCLOUD** strategy — rather than going 'all-in' with one provider, they are passing different types of workloads to different providers in a best-of-breed approach. This may

© ThoughtWorks, Inc. All Rights Reserved.

involve, for example, putting standard services on AWS, but using Google for machine learning, Azure for .NET applications that use SQLServer, or potentially using the Ethereum Consortium Blockchain solution. This is different than a cloud-agnostic strategy of aiming for portability across providers, which is costly and forces lowest-common-denominator thinking. Polycloud instead focuses on using the best that each cloud offers.

A service mesh offers consistent discovery, security, tracing, monitoring and failure handling without the need for a shared asset such as an API gateway or ESB.

(Service mesh)

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

Microservices architecture, with a large number of services exposing their assets and capabilities through APIs and an increased attack surface, demand a zero trust security architecture — 'never trust, always verify'. However, enforcing security controls for communication between services is often neglected, due to increased service code complexity and lack of libraries and language support in a polyglot environment. To get around this complexity, some teams delegate security to an out-of-process sidecar — a process or a container that is deployed and scheduled with each service sharing the same execution context, host and identity. Sidecars implement security capabilities, such as transparent encryption of the communication and TLS (Transport Layer Security) termination, as well as authentication and authorization of the calling service or the end user. We recommend you look into using Istio, linkerd or Envoy before implementing your own **SIDECARS FOR ENDPOINT SECURITY**.

Traditional approaches to enterprise security often emphasize locking things down and slowing the pace of change. However, we know that the more time an attacker has to compromise a system, the greater the potential damage. The three Rs of enterprise security — rotate, repair and repave — take advantage of infrastructure automation and continuous delivery to eliminate opportunities for attack. Rotating credentials, applying patches as soon as they're available and rebuilding systems from a known, secure state — all within a matter of minutes or hours — makes it harder for attackers to succeed. **THE THREE Rs OF SECURITY** technique is made feasible with the advent of modern cloud-native architectures. When applications are deployed as containers, and built and tested via a completely automated pipeline, a security patch is just another small release that can be sent through the pipeline with one click. Of course, in keeping with best distributed systems practices, developers need to design their applications to be resilient to unexpected server outages. This is similar to the impact of implementing Chaos Monkey within your environment.

Kafka is becoming very popular as a messaging solution, and along with it, Kafka Streams is at the forefront of the wave of interest in streaming architectures. Unfortunately, as they start to embed Kafka at the heart of their data and application platforms, we're seeing some organizations **RECREATING ESB ANTIPATTERNS WITH KAFKA** by centralizing the Kafka ecosystem components — such as connectors and stream processors — instead of allowing these components to live with product or service teams. This reminds us of seriously problematic ESB antipatterns, where more and more logic, orchestration and transformation were thrust into a centrally managed ESB, creating a significant dependency on a centralized team. We're calling this out to dissuade further implementations of this flawed pattern.

© ThoughtWorks, Inc. All Rights Reserved.

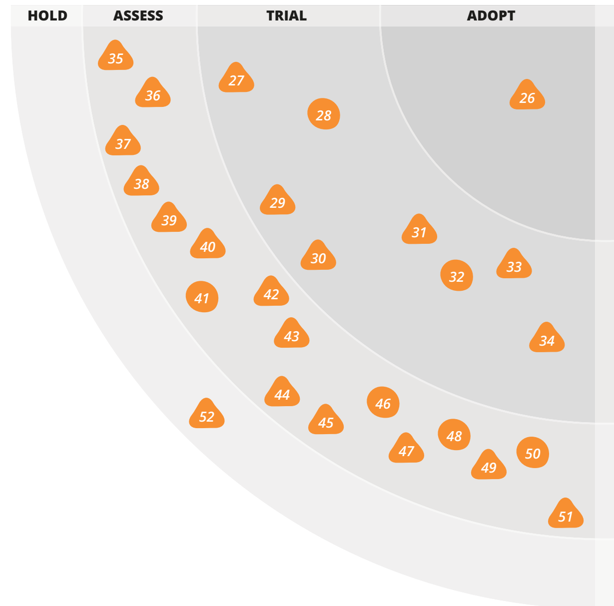
PLATFORMS

ADOPT
26. Kubernetes

TRIAL
27. .NET Core
28. AWS Device Farm
29. Flood IO *NEW*
30. Google Cloud Platform *NEW*
31. Keycloak
32. OpenTracing
33. Unity beyond gaming
34. WeChat *NEW*

ASSESS
35. Azure Service Fabric *NEW*
36. Cloud Spanner *NEW*
37. Corda *NEW*
38. Cosmos DB *NEW*
39. DialogFlow
40. GKE *NEW*
41. Hyperledger
42. Kafka Streams
43. Language Server Protocol *NEW*
44. LoRaWAN *NEW*
45. MapD *NEW*
46. Mosquitto
47. Netlify *NEW*
48. PlatformIO
49. TensorFlow Serving *NEW*
50. Voice platforms
51. Windows Containers *NEW*

HOLD
52. Overambitious API gateways



Since we last mentioned **KUBERNETES** in the Radar, it has become the default solution for most of our clients when deploying containers into a cluster of machines. The alternatives didn't capture as much mindshare, and in some cases our clients are even changing their 'engine' to Kubernetes. Kubernetes has become the container orchestration platform of choice for major public cloud platforms, including Microsoft's Azure Container Service and Google Cloud (see the GKE blip). And there are many useful products enriching the fast-growing Kubernetes ecosystem. Platforms that try to hide Kubernetes under an abstraction layer, however, have yet to prove themselves.

We're seeing increased adoption of **.NET CORE**, the open source cross-platform software framework. .NET Core enables the development and deployment of .NET applications on Windows, macOS and Linux. With the release of .NET Standard 2.0 increasing the number of standard APIs across .NET platforms, the migration path to .NET Core has become clearer. Issues related to library support on .NET Core are becoming less problematic, and first-class cross-platform tooling is now available, allowing for productive development on non-Windows platforms. Blessed Docker images are provided to make it easy to integrate .NET Core services into a containerized environment. Positive directions in the community and feedback from our projects indicate that .NET Core is ready for widespread use.

Load testing became easier with the maturity of tools such as Gatling and Locust. At the same time, elastic cloud infrastructures make it possible to simulate a large number of client instances. We're delighted to see Flood and other cloud platforms go further by leveraging these technologies. **FLOOD IO** is an SaaS load-testing service that helps to distribute and execute testing scripts across hundreds of servers in the cloud. Our teams find it simple to migrate performance testing to Flood by reusing existing Gatling scripts.

© ThoughtWorks, Inc. All Rights Reserved.

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

Google Cloud Platform (GCP) has expanded in terms of available geographic regions and maturity of services and it's now a serious consideration in cloud strategy globally.

(Google Cloud Platform)

In a microservice, 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. It supports single sign-on, social login and standard protocols such as OpenID Connect, OAuth 2.0 and SAML out of the box. Our teams have been using this tool and plan to keep using it for the foreseeable future. But it requires a little work to set up. Because configuration happens both at initialization and at runtime through APIs, it's necessary to write scripts to ensure deployments are repeatable.

In previous Radars, we mentioned that Unity has become the platform of choice for VR and AR application development because it provides the abstractions and tooling of a mature platform, while being more accessible than its main alternative, the Unreal Engine. With the recent introductions of ARKit for iOS and ARCore for Android, the two main mobile platforms now have powerful native SDKs for building augmented reality applications. Yet, we feel that many teams, especially those without deep experience in building games, will benefit from using an abstraction such as Unity, which is why we're calling out **UNITY BEYOND GAMING**. This allows developers unfamiliar with the technology to focus on one SDK. It also offers a solution for the huge number of devices, especially on the Android side, that are not supported by the native SDKs.

WECHAT, often seen as a WhatsApp equivalent, is becoming the de facto business platform in China. Many people may not know but WeChat is also one of the most popular online payment platforms. With the app's built-in CMS and membership management, small businesses are now conducting their commerce entirely on WeChat. Through the Service Account feature, large organizations can interface their internal system to their employees. Given that more than 70 percent of Chinese people are using WeChat, it's an important consideration for businesses that want to expand into the China market.

AZURE SERVICE FABRIC is a distributed systems platform built for microservices and containers. It's comparable to container orchestrators such as Kubernetes, but also works with plain old services. It can be used in a bewildering array of ways, starting from simple services in your language of choice to Docker containers or services built using an SDK. Since its release a couple of years ago, it has steadily added more features, including Linux container support. Kubernetes has been the poster child of container orchestration tools, but Service Fabric is the default choice for .NET applications. We're using it in a few projects at ThoughtWorks and we like what we've seen so far.

CLOUD SPANNER is a fully managed relational database service offering high availability and strong consistency without compromising latency. Google has been working on a globally distributed database called Spanner for quite some time. It has recently released the service to the outside world as Cloud Spanner. You can scale your database instance from one to thousands of nodes across the globe without worrying about data consistency. By levering TrueTime, a highly available and distributed clock, Cloud Spanner provides strong consistency for reads and snapshots. You can use standard SQL to read data from Cloud Spanner, but for write operations you have to use their RPC API. Although not all services would require a global-scale distributed database, the general availability of Cloud Spanner is a big shift in the way we think about databases. And its design is influencing open source products such as CockroachDB.

© ThoughtWorks, Inc. All Rights Reserved.

After thorough exploration, R3, an important player in the blockchain space, realized that blockchain doesn't fit their purpose well, so they created **CORDA**. Corda is a distributed ledger technology (DLT) platform focused on the financial field. R3 have a very clear value proposition and know that their problem requires a pragmatic technology approach. This matches our own experience; current blockchain solutions may not be the reasonable choice for some business cases, due to mining costs and operational inefficiency. Although the development experience we have on Corda thus far has not been the smoothest, APIs are still unstable after v1.0 release, we expect to see the DLT space mature further.

COSMOS DB is Microsoft's globally distributed, multimodel database service, which became generally available earlier this year. While most modern NoSQL databases offer tunable consistency, Cosmos DB makes it a first-class citizen and offers five different consistency models. It's worth highlighting that it also supports multiple models — key value, document, column family and graph — all of which map to its internal data model, called atom-record-sequence (ARS). One interesting aspect of Cosmos DB is that it offers service level agreements (SLAs) on its latency, throughput, consistency and availability. With its wide range of applicability, it has set a high standard for other cloud vendors to match.

In parallel with the recent surge of chatbots and voice platforms, we've seen a proliferation of tools and platforms that provide a service to extract intent from text and management of conversational flows that you can hook into. **DIALOGFLOW** (formerly API.ai), which was acquired by Google, is one such 'natural-language-understanding as a service' offering that competes with wit.ai and Amazon Lex among other players in this space.

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

Language servers pull features such as autocomplete, finding callers and refactoring into an API that allows any text editor to work with the language's abstract syntax tree.

(Language Server Protocol)

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). New developments include 'exactly once' stream processing in a Kafka cluster. This was achieved by introducing idempotency in Kafka producers and allowing atomic writes across multiple partitions using the new Transactions API.

Much of the power of sophisticated IDEs comes from their ability to parse a program into an abstract syntax tree (AST) and then use that AST for program analysis and manipulation. This supports features such as autocomplete, finding callers and refactoring. Language servers pull this capability into a process that allows any text editor to access an API to work with the AST. Microsoft has led the creation of the **LANGUAGE SERVER PROTOCOL** (LSP), harvested from their OmniSharp and TypeScript Server projects. Any editor that uses this protocol can work with any language that has an LSP-compliant server. This means we can keep using our favorite editors without forgoing the rich text editing modes of many languages — much to the delight of our Emacs addicts.

© ThoughtWorks, Inc. All Rights Reserved.

LORAWAN is a low-power wide-area network, designed for low-power consumption and communication over long distances using low bitrates. It provides for communication between devices and gateways, which can then forward the data to, for example, applications or servers. A typical usage is for a distributed set of sensors, or for Internet of Things (IoT) devices, for which long battery life and long-range communication is a must. LoRaWAN addresses two of the key problems with attempting to use normal Wi-Fi for such applications: range and power consumption. There are several implementations, a notable one being The Things Network, a free, open source implementation.

As we move from experimental use to production, we need a reliable way to host and deploy machine learning models that can be accessed remotely and scale with the number of consumers.

(TensorFlow Serving)

MAPD is an in-memory columnar analytic database with SQL support that's built to run on GPU. We debated whether the database workload is actually I/O or computationally bound but there are instances where the parallelism of the GPU, combined with the large bandwidth of VRAM, can be quite useful. MapD transparently manages the most frequently used data in VRAM (such as columns involved in group-by, filters, calculations and join conditions) and stores the rest of the data in the main memory. With this memory management setup, MapD achieves significant query performance without the need of indexes. Although there are other GPU database vendors, MapD is leading this segment with the recent open source release of its core database and through the GPU Open Analytics Initiative. If your analytical workload is computationally heavy, can exploit GPU parallelism and can fit in the main memory, we recommend assessing MapD.

We like simple tools that solve one problem really well, and **NETLIFY** fits this description nicely. You can create static website content, check it into GitHub and then quickly and easily get your site live and available. There is a CLI available to control the process; content delivery networks (CDNs) are supported; it can work alongside tools such as Grunt; and, most importantly, Netlify supports HTTPS.

Machine-learning models are starting to creep into everyday business applications. When enough training data is available, these algorithms can address problems that might have previously required complex statistical models or heuristics. As we move from experimental use to production, we need a reliable way to host and deploy the models that can be accessed remotely and scale with the number of consumers. **TENSORFLOW SERVING** addresses part of that problem by exposing a remote gRPC interface to an exported model; this allows a trained model to be deployed in a variety of ways. TensorFlow Serving also accepts a stream of models to incorporate continuous training updates, and its authors maintain a Dockerfile to ease the deployment process. Presumably, the choice of gRPC is to be consistent with the TensorFlow execution model; however, we're generally wary of protocols that require code generation and native bindings.

Microsoft is catching up in the container space with **WINDOWS CONTAINERS**. At the time of writing, Microsoft provides two Windows OS images as Docker containers, Windows Server 2016 Server Core and Windows Server 2016 Nano Server. Although there is room for improvement for Windows Containers, for instance, decreasing the large image sizes, and enriching ecosystem support and documentation, our teams have started using them in scenarios where other containers have been working successfully, such as build agents.

We remain concerned about business logic and process orchestration implemented in middleware, especially where it requires expert skills and tooling while creating single points of scaling and control. Vendors in the highly competitive API gateway market are continuing this trend by adding features through which they attempt to differentiate their products. This results in **OVERAMBITIOUS API GATEWAY** products whose functionality — on top of what is essentially a reverse proxy — encourages designs that continue to be difficult to test and deploy. API gateways do provide utility in dealing with some specific concerns — such as authentication and rate limiting — but any domain smarts should live in applications or services.

© ThoughtWorks, Inc. All Rights Reserved.

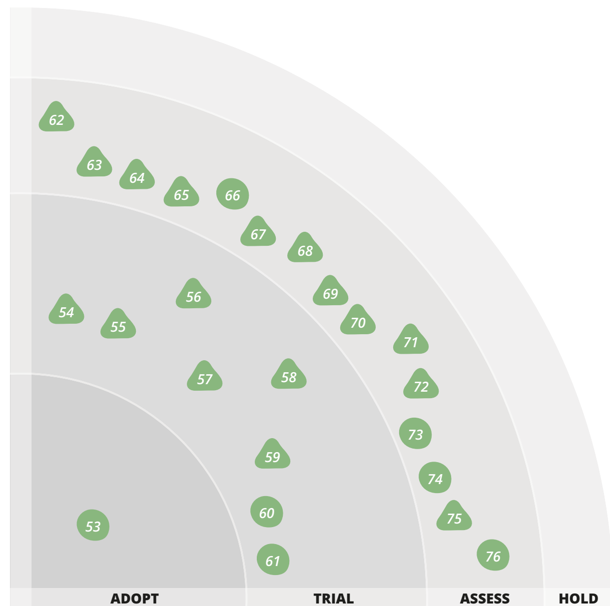
TOOLS

ADOPT
53. fastlane

TRIAL
54. Buildkite *NEW*
55. CircleCI *NEW*
56. gopass *NEW*
57. Headless Chrome for front-end test *NEW*
58. jsoniter *NEW*
59. Prometheus
60. Scikit-learn
61. Serverless Framework

ASSESS
62. Apex *NEW*
63. assertj-swagger *NEW*
64. Cypress *NEW*
65. Flow *NEW*
66. InSpec
67. Jupyter *NEW*
68. Kong API Gateway *NEW*
69. kops *NEW*
70. Lighthouse *NEW*
71. Rendertron *NEW*
72. Sonobuoy *NEW*
73. spaCy
74. Spinnaker
75. Spring Cloud Contract *NEW*
76. Yarn

HOLD



Our teams very much like the hosted CI/CD tool **BUILDKITE** for its simplicity and quick setup. With Buildkite, you provide your own machines to execute builds — on premise or in the cloud — and install a lightweight agent application to connect the build agent to the hosted service. In many cases, having this level of control over the configuration of your build agents is a plus when compared to using hosted agents.

CIRCLECI is a continuous integration engine offered as SaaS and on premise. CircleCI has been the go-to SaaS CI tool for many of our development teams, who needed a low-friction and easy-to-setup build and deployment pipeline. CircleCI version 2.0 supports workflows of build jobs, with fan-in and fan-out flows and manual gates, as well as mobile development. It allows developers to run the pipelines locally and easily integrates with Slack and other notification and alerting systems. We recommend you take a closer look at the security practices of CircleCI, just as you would with any other SaaS product that hosts your company's assets.

A descendant of pass, gopass adds features such as support for recipient management and multiple password stores in a single tree; an interactive search functionality; time-based one-time password (TOTP) support; and storage of binary data.

(gopass)

GOPASS is a password management solution for teams, built on GPG and Git. It's a descendant of pass and adds features such as: support for recipient management and multiple password stores in a single tree; an interactive search functionality; time-based one-time password (TOTP) support; and storage of binary data. Migration of your pass store is fairly straightforward, because gopass is largely compatible with the format pass uses. This also means integration into provisioning workflows can be achieved with a single call to a stored secret.

© ThoughtWorks, Inc. All Rights Reserved.

Since mid-2017, Chrome users have had the option of running the browser in headless mode. This feature is ideally suited to running front-end browser tests without the overhead of displaying actions on a screen. Previously, this was largely the province of PhantomJS but Headless Chrome is rapidly replacing the JavaScript-driven WebKit approach. Tests in Headless Chrome should run much faster, and behave more like a real browser, but our teams have found that it does use more memory than PhantomJS. With all these advantages, **HEADLESS CHROME FOR FRONT-END TEST** is likely to become the de facto standard.

If you're looking for a JSON encoder/decoder with high performance in Go and Java, check out the open source **JSONITER** library. The library is compatible with the standard JSON encoding package in Go.

We've seen continuing improvements in and an uptick in adoption of Prometheus, the monitoring and time series database tool originally developed by Soundcloud.
(Prometheus)

We've seen both continuing improvements in and an uptick in adoption of **PROMETHEUS**, the monitoring and time series database tool originally developed by Soundcloud. Prometheus primarily supports a pull-based HTTP model but it also supports alerts, making it an active part of your operational toolset. As of this writing, Prometheus 2.0 is in prerelease, and continues to evolve. Prometheus developers have focused their efforts on core time series databases and the variety of metrics available. Grafana has become the dashboard visualization tool of choice for Prometheus users and support for Grafana ships with the tool. Our teams also find that Prometheus monitoring nicely complements the indexing and search capabilities of an Elastic Stack.

APEX is a tool to build, deploy and manage AWS Lambda functions with ease. With Apex, you can write functions in languages that are not yet natively supported in AWS, including Golang, Rust and others. This is made possible by a Node.js shim, which creates a child process and processes events through stdin and stdout. Apex has a lot of nice features that improve the developer experience, and we particularly like the ability to test functions locally and perform a dry run of the changes before they're applied to AWS resources.

An AssertJ library, **ASSERTJ-SWAGGER** enables you to validate an API implementation's compliance with its contract specification. Our teams use assertj-swagger to catch problems when the API endpoint implementation changes without updating its Swagger specification, or fails to publish the updated documentation.

Fixing end-to-end test failures in CI can be a painful experience, especially in headless mode. **CYPRESS** is a useful tool that helps developers build end-to-end tests easily and records all test steps as a video in an MP4 file. Instead of reproducing the issue in headless mode, developers can watch the testing video in order to fix it. Cypress is a powerful platform, not only a testing framework. Currently, we've integrated its CLI with headless CI in our projects.

FLOW is a static type checker for JavaScript that allows you to add type checking across the codebase incrementally. Unlike Typescript, which is a different language, Flow can be added incrementally to an existing JavaScript codebase supporting the 5th, 6th and 7th editions of ECMAScript. We suggest adding Flow to your continuous integration pipeline, starting with the code that concerns you most. Flow adds to the clarity of the code, increases the reliability of refactoring and catches type-related bugs early during the build.

Over the last couple of years, we've noticed a steady rise in the popularity of analytics notebooks. These are Mathematica-inspired applications that combine text, visualization and code in a living, computational document. In a previous edition, we mentioned GorillaREPL, a Clojure variant of these. But increased interest in machine learning — along with the emergence of Python as the programming language of choice for practitioners in this field — has focused particular attention on Python notebooks, of which **JUPYTER** seems to be gaining the most traction among ThoughtWorks teams.

Kong is an open source API gateway built and sponsored by Mashape, who also provide an enterprise offering integrating Kong with their proprietary API analytics and developer portal tools. They can be deployed in a variety of configurations, as an edge API gateway or an internal API proxy. OpenResty, through its Nginx modules, provides a strong and performant foundation, with Lua plugins for extensions. Kong can either use PostgreSQL for single region deployments or Cassandra for multiregion configurations. Our

© ThoughtWorks, Inc. All Rights Reserved.

developers have enjoyed Kong's high performance, its API-first approach (which enables automation of its configuration) and its ease of deployment as a container. **KONG API GATEWAY**, unlike overambitious API gateways, has a smaller set of features but it implements the essential set of API gateway capabilities such as traffic control, security, logging, monitoring and authentication. We're excited to assess Kong in a sidecar configuration in the near future.

KOPS is a command line tool for creating and managing high-availability production Kubernetes clusters. Initially targeting AWS, it now has experimental support for other providers. It can get you up and running fast, and even though a few features (such as rolling upgrades) have yet to be fully developed, we've been impressed by the community.

LIGHTHOUSE is a tool written by Google to assess web applications for adherence to Progressive Web App standards. This year's Lighthouse 2.0 release adds performance metrics and accessibility checks to the basic toolset. This added functionality has now been incorporated into the standard Chrome developer tools under the audit tab. Lighthouse 2.0 is yet another beneficiary of Chrome's headless mode. This provides an alternative to Pa11y and similar tools for running accessibility checks in a continuous integration pipeline, since the tool can be run from the command line or standalone as a Node.js application.

A perennial problem for JavaScript-heavy web applications is how to make the dynamic portion of those pages available to search engines. Historically, developers have resorted to a variety of tricks, including server-side rendering with React, external services or prerendering content. Now Google Chrome's new headless mode adds a new 'trick' to the toolbox — **RENDERTRON**, a headless Chrome rendering solution. Rendertron wraps an instance of headless Chrome in a Docker container, ready to deploy as a standalone HTTP server. Bots that don't render JavaScript can be routed to this server to do the rendering for them. Although developers can always deploy their own headless Chrome proxy and associated routing machinery, Rendertron simplifies the configuration and deployment process, and provides example middleware code for detecting and routing bots.

A perennial problem for JavaScript-heavy web applications is how to make the dynamic portion of those pages available to search engines. Bots that don't render JavaScript can be routed to a Rendertron server to do the rendering for them.

(Rendertron)

SONOBUOY is a diagnostic tool for running end-to-end conformance tests on any Kubernetes cluster in a nondestructive way. The team at Heptio, which was founded by two creators of the Kubernetes projects, built this tool to ensure that the wide array of Kubernetes distributions and configurations conform to the best practices, while following the open source standardization for interoperability of clusters. We're experimenting with Sonobuoy to run as part of our infrastructure as code build pipeline, as well as continuous monitoring of our Kubernetes installations, to validate the behavior and health of the whole cluster.

If you're implementing Java services using the Spring framework, you may want to consider **SPRING CLOUD CONTRACT** for consumer-driven contract testing. The current ecosystem of this tool supports verification of the client calls and the server implementation against the contract. In comparison to Pact, an open source consumer-driven contract testing tool set, it lacks the brokering of the contracts and the support for other programming languages. However, it integrates well with the Spring ecosystem, for instance message routing with Spring Integration.

© ThoughtWorks, Inc. All Rights Reserved.

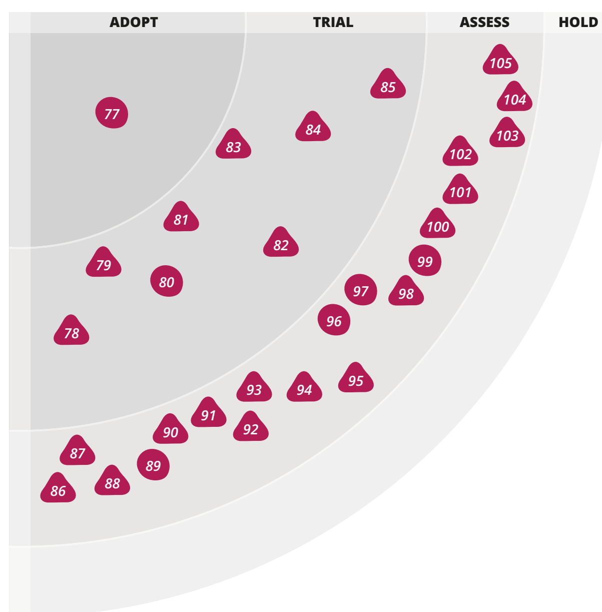
LANGUAGES & FRAMEWORKS

ADOPT
77. Python 3

TRIAL
78. Angular
79. AssertJ *NEW*
80. Avro
81. CSS Grid Layout *NEW*
82. CSS Modules *NEW*
83. Jest *NEW*
84. Kotlin
85. Spring Cloud

ASSESS
86. Android Architecture Components *NEW*
87. ARKit/ARCore *NEW*
88. Atlas and BeeHive *NEW*
89. Caffe
90. Clara rules *NEW*
91. CSS-in-JS *NEW*
92. Digdag *NEW*
93. Druid *NEW*
94. ECharts *NEW*
95. Gobot *NEW*
96. Instana
97. Keras
98. LeakCanary *NEW*
99. PostCSS
100. PyTorch *NEW*
101. single-spa *NEW*
102. Solidity *NEW*
103. TensorFlow Mobile *NEW*
104. Truffle *NEW*
105. Weex *NEW*

HOLD



In previous Radar editions, we've been hesitant to give **ANGULAR** a strong recommendation because it was essentially a new, and on the whole unexciting, framework, sharing only its name with AngularJS, an older framework we loved in days past. In the meantime, Angular, now in version 5, has improved steadily while providing backward compatibility along the way. Several of our teams have Angular applications in production and reportedly, they like what they see. For this reason, we're moving Angular into the Trial ring in this Radar, to signify that some of our teams now consider it a solid choice. Most of our teams, however, still prefer React, Vue or Ember over Angular.

ASSERTJ is a Java library that provides a fluent interface for assertions, which makes it easy to convey intent within test code. AssertJ gives readable error messages, soft assertions, and improved collections and exception support. We're seeing some teams default to its use instead of JUnit combined with Hamcrest.

CSS Grid Layout is a two-dimensional grid-based layout system that provides a mechanism to divide available space for layout into columns and rows using a set of predictable sizing behaviors.

(CSS Grid Layout)

CSS is the preferred choice for laying out web pages, even when it did not provide much explicit support for creating layouts. Flexbox helped with simpler, one-dimensional layouts, but developers usually reached for

© ThoughtWorks, Inc. All Rights Reserved.

libraries and toolkits for more complex layouts. **CSS GRID LAYOUT** is a two-dimensional grid-based layout system that provides a mechanism to divide available space for layout into columns and rows using a set of predictable sizing behaviors. Grid does not require any libraries and plays well with Flexbox and other CSS display elements. However, since IE11 is only partially supported, it ignores users who still depend on a Microsoft browser on Windows 7.

Most large CSS codebases require complex naming schemes to help avoid naming conflicts in the global namespace. **CSS MODULES** address these problems by creating a local scope for all class names in a single CSS file. This file is imported to a JavaScript module, where CSS classes are referenced as strings. Then, in the build pipeline (Webpack, Browserify, etc.), the class names are replaced with generated unique strings. This is a significant change in responsibilities. Previously, a human had to manage the global namespace, to avoid class naming conflicts; now that responsibility rests with the build tooling. A small downside we've encountered with CSS Modules: functional tests are usually out of the local scope and can therefore not reference classes by the name defined in the CSS file. We recommend using IDs or data attributes instead.

Jest is a 'zero configuration' front-end testing tool with out-of-the-box features such as mocking and code coverage, targeted at React and other JavaScript frameworks.
(Jest)

Our teams are delighted with the results of using **JEST** for front-end testing. It provides a 'zero-configuration' experience and has out-of-the-box features such as mocking and code coverage. You can apply this testing framework not only to React applications, but also to other JavaScript frameworks. One of Jest's often hyped features is UI snapshot testing. Snapshot testing would be a good addition to the upper layer of the test pyramid, but remember, unit testing is still the solid foundation.

The announcement of first-class Android support has given an extra boost to the rapidly progressing **KOTLIN** language, and we're closely following the progress of Kotlin/Native — the LLVM-backed ability to compile to native executables. Null safety, data classes and the ease of creating DSLs are some of the benefits we've enjoyed, along with the Anko library for Android development. Despite the downsides of slow initial compilation and reliance on IntelliJ for first-class IDE support, we recommend giving this fresh and concise modern language a try.

SPRING CLOUD continues to evolve and add interesting new features. Support for binding to Kafka Streams, for example, in the spring-cloud-streams project makes it relatively easy to build message driven applications with connectors for Kafka and RabbitMQ. The teams we have using it appreciate the simplicity it brings to using sometimes complex infrastructure, such as ZooKeeper, and support for common problems that we need to address when building distributed systems, tracing with the spring-cloud-sleuth for example. The usual caveats apply but we're successfully using it on multiple projects.

Historically, Google's Android documentation examples lacked architecture and structure. This changes with the release of **ANDROID ARCHITECTURE COMPONENTS**, a set of opinionated libraries that help developers create Android applications with better architecture. They address longstanding pain points of Android development: handling lifecycles; pagination; SQLite databases; and data persistence over configuration changes. The libraries don't need to be used together — you can pick the ones you need most and integrate them into your existing project.

We've seen a flurry of activity in mobile augmented reality much of it fueled by **ARKIT AND ARCORE**, the native AR libraries used by Apple and Google, respectively. These libraries are bringing mobile AR technologies to the mainstream. However, the challenge will be for companies to find use cases that go beyond gimmicky and provide genuine solutions that actually enhance the user experience.

A multi-app strategy is really controversial, particularly at a time when fewer and fewer users are downloading new apps. Instead of introducing a new app and struggling with the download numbers, multiteams have to deliver functionality via a single app that is already widely installed, which creates an architectural challenge. **ATLAS AND BEEHIVE** are modularization solutions for Android and iOS apps, respectively. Atlas and BeeHive enable multiteams working on physically isolated modules to reassemble or dynamically load

© ThoughtWorks, Inc. All Rights Reserved.

these modules from a facade app. Both are Alibaba open source projects, since Alibaba encountered the same problem of dwindling downloads and single-app architectural challenges.

Our first rule of thumb in selecting a rules engine is normally: you don't need a rules engine. We've seen too many people tying themselves to a hard-to-test black-box rules engine for spurious reasons, when custom code would have been a better solution. That said, we've had success using **CLARA RULES** for scenarios where a rules engine does make sense. We like that it uses simple Clojure code to express and evaluate the rules, which means they are amenable to refactoring, testing and source control. Rather than chasing the illusion that business people should directly manipulate the rules, it drives collaboration between the business experts and developers.

CSS IN JS is a technique of writing CSS styling in the JavaScript programming language. This encourages a common pattern of writing the styling with the JavaScript component it applies to, co-locating presentational and logical concerns. The new players — including JSS, emotion and styled-components — rely on the tooling to translate the CSS-in-JS code to separate CSS stylesheets, to make them suitable for browser consumption. This is the second-generation approach to writing CSS in JavaScript and unlike the previous approaches doesn't rely on in-line styles. That means it provides the benefit of supporting all CSS features, sharing of CSS using the npm ecosystem and utilization of components across multiple platforms. Our teams have found styled-components working well with component-based frameworks, such as React, and unit testing of CSS with jest-styled-components. This space is new and rapidly changing; the approach requires some effort for manual debugging of the generated class names in the browser, and it may not apply to some projects where the front-end architecture does not support reusing components and requires global styling.

DIGDAG is a tool for building, running, scheduling and monitoring complex data pipelines in the cloud. You can define these pipelines in YAML, using either the rich set of out-of-the-box operators or building your own through the API. Digdag has most of the common features in a data pipeline solution such as dependency management, modular workflow to promote reuse, secured secret management and multilingual support.

The feature we're most excited about is polycloud support, which lets you move and join data across AWS RedShift, S3, and Google BigQuery. As more and more cloud providers offer competing data-processing solutions, we think Digdag (and similar tools) will be useful in leveraging the best option for the task.

DRUID is a JDBC connection pool with rich monitoring features. It has a built-in SQL parser, which provides semantic monitoring of the SQL statements executing in the database. Injections or suspicious SQL statements will be blocked and logged directly from the JDBC layer. What's more, queries can be merged based on their semantics. This is an Alibaba open source project, and reflects the lessons Alibaba learnt from operating their own database systems.

Android Architecture Components are a set of opinionated libraries that help developers create Android applications with better architecture.

(Android Architecture Components)

ECHARTS is a lightweight charting library with rich support for different types of charts and interactions. Since ECharts is entirely based on the Canvas API, it has incredible performance even when dealing with over 100k data points, and it's also been optimized for mobile usage. Together with its sibling project, ECharts-X, it can support 3D plotting. ECharts is a Baidu open source project.

The ability to compile the Go programming language to bare metal targets has raised interest among developers in using the language for embedded systems. **GOBOT** is a framework for robotics, physical computing, and the Internet of Things, written in the Go programming language and supporting a variety of platforms. We've used the framework for experimental robotic projects where real-time response hasn't been a requirement, and we've created open source software drivers with Gobot. Gobot HTTP APIs enable simple hardware integration with mobile devices to create richer applications.

Our mobile teams have been excited about **LEAKCANARY**, a tool for detecting annoying memory leaks in Android and Java. It's simple to hook up and provides notifications with a clear trace-back to the

© ThoughtWorks, Inc. All Rights Reserved.

cause of the leak. Adding this to your toolkit can save tedious hours troubleshooting out-of-memory errors on multiple devices.

PYTORCH is a complete rewrite of the Torch machine learning framework from Lua to Python. Although quite new and immature compared to Tensorflow, programmers find PyTorch much easier to work with. Because of its object-orientation and native Python implementation, models can be expressed more clearly and succinctly and debugged during execution. Although many of these frameworks have emerged recently, PyTorch has the backing of Facebook and broad range of partner organisations, including NVIDIA, which should ensure continuing support for CUDA architectures. ThoughtWorks teams find PyTorch useful for experimenting and developing models but still rely on TensorFlow's performance for production-scale training and classification.

SINGLE-SPA is a JavaScript metaframework that allows us to build micro frontends using different frameworks that can coexist in a single application. In general, we don't recommend using more than one framework for an application, but there are times when we can't avoid doing so. For instance, single-spa can be quite useful when you're working with a legacy application and you want to experiment by developing a new feature, with either a new version of the existing framework or a completely different one. Given the short life span of many JavaScript frameworks, we see a need for a solution that would allow for future framework changes and localized experimentation, without affecting the entire application. single-spa seems to be a good start in that direction.

Programming for smart contracts requires a more expressive language than a scripting system for transactions. **SOLIDITY** is the most popular among the new programming languages designed for smart contracts. Solidity is a contract-oriented, statically typed language whose syntax is similar to JavaScript. It provides abstractions for writing self-enforcing business logic in smart contracts. The toolchain around Solidity is growing fast. Nowadays, Solidity is the primary choice on the Ethereum platform. Given the immutable nature of deployed smart contracts, it should go without saying that rigorous testing and audit of dependencies is vital.

TENSORFLOW MOBILE makes it possible for developers to incorporate a wide range of comprehension and classification techniques into their iOS or Android applications. This is particularly useful given the range of sensor data available on mobile phones. Pretrained TensorFlow models can be loaded into a mobile application and applied to inputs such as live video frames, text or speech. Mobile phones present a surprisingly opportune platform for implementing these computational models. TensorFlow models are exported and loaded as protobuf files, which can present some problems for implementers. Protobuf's binary format can make it hard to examine models and requires that you link the correct protobuf library version to your mobile app. But local model execution offers an attractive alternative to TensorFlow Serving without the communication overhead of remote execution.

We've had success using Clara rules for scenarios where a rules engine makes sense. We like that it uses simple Clojure code to express and evaluate the rules, which means they are amenable to refactoring, testing and source control.

(Clara rules)

TRUFFLE is a development framework that brings a modern web development experience to the Ethereum platform. It takes over the job of smart contract compiling, library linking and deployment, as well as dealing with artifacts in different blockchain networks. One of the reasons we love Truffle is that it encourages people to write tests for their smart contracts. You need to take tests really seriously as smart contract programming is often related to money. With its built-in testing framework and integration with TestRPC, Truffle makes it possible to write the contract in a TDD way. We expect to see more technologies similar to Truffle to promote continuous integration in the blockchain area.

WEEX is a framework for building cross-platform mobile apps by using the Vue.js component syntax. For those who prefer the simplicity of Vue.js, Weex is a viable option for native mobile apps, but it also works very well for more complicated apps. We see many successes for fairly complicated mobile apps built on this framework, including TMall and Taobao, two of the most popular mobile apps in China. Weex was developed by Alibaba, and is now an Apache incubator project.

© ThoughtWorks, Inc. All Rights Reserved.

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

SUBSCRIBE NOW

thght.works/Sub-EN

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

Founded over 20 years ago, ThoughtWorks has grown to a company of over 4500 people, including a products division which makes pioneering tools for software teams. ThoughtWorks has 42 offices across 15 countries: Australia, Brazil, Canada, Chile, China, Ecuador, Germany, India, Italy, Singapore, South Africa, Spain, Turkey, the United Kingdom and the United States.

thoughtworks.com