ThoughtWorks®

TECHNOLOGY RADAR

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

JANUARY 2015

thoughtworks.com/radar

WHAT'S NEW?

Here are the trends highlighted in this edition:

EXPLOSIVE GROWTH IN THE DEVOPS ARENA

Much of the work on this radar involved evaluating a sea of technologies related to DevOps, which continues to grow and innovate at a break-neck pace. With the advent of containerization, cloud offerings, and various permutations and combinations, it's no wonder that innovation is rampant in this space. The growth and popularity of architectural styles like microservices emphasizes the intersection of architecture and DevOps, so we expect the pace of innovation to continue.

NEXT GENERATION DATA PLATFORMS GAIN TRACTION

Big Data isn't new—in fact we've been advising against buying into the hype for some time—but we're beginning to see related technologies take hold and find uses in the Enterprise. Techniques such as the Data Lake and Lambda Architectures are new ways of looking at enterprise data platforms, and can be useful regardless of whether you really have "big" data to deal with.

DEVELOPERS FOCUS ON SECURITY-MINDED TOOLING

Each week brings new stories of data leaks or misuse and public demand for secure, privacy-respecting systems is on the rise. Blackbox, TOTP two-factor authentication and OpenID Connect—all tools featured on this edition of the Radar—can help developers create secure systems and infrastructure.

CONTRIBUTORS

The ThoughtWorks Technology Advisory Board is comprised of:

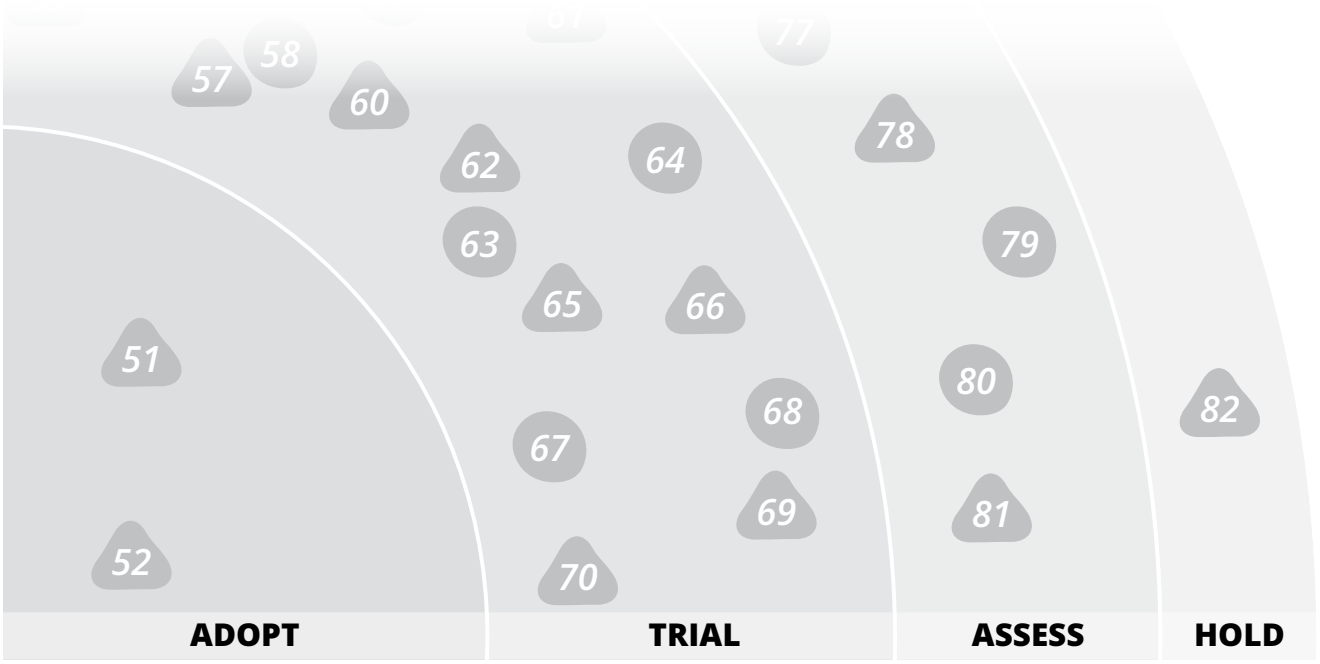
Rebecca Parsons (CTO)	Erik Doernenburg	Jeff Norris	Sam Newman
Martin Fowler(Chief Scientist)	Evan Bottcher	Jonny LeRoy	Scott Shaw
Badri Janakiraman	Hao Xu	Mike Mason	Srihari Srinivasan
Brain Leke	Ian Cartwright	Neal Ford	Thiyagu Palanisamy
Claudia Melo	James Lewis	Rachel Laycock	

© January 2015, ThoughtWorks, Inc. All Rights Reserved.

ABOUT THE TECHNOLOGY RADAR

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

The radar captures the output of the Technology Advisory Board's discussions in a format that provides value to a wide range of stakeholders, from CIOs to developers. The content is intended as a concise summary. We encourage you to explore these technologies for more detail. The radar is graphical in nature, grouping items into techniques, tools, platforms, and languages & frameworks. When radar items could appear in multiple quadrants, we chose the one that seemed most appropriate. We further group these items in four rings to reflect our current position on them. The rings are:



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

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

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

Proceed with caution.

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

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

© January 2015, ThoughtWorks, Inc. All Rights Reserved.

THE RADAR

TECHNIQUES

ADOPT
1. Focus on mean time to recovery
2. Forward Secrecy
3. Structured logging

TRIAL
4. Canary builds
5. Datensparsamkeit
6. Front end instrumentation
7. Hipster batch
8. Humane registry
9. Inverse Conway Maneuver
10. Living CSS Style Guides
11. Local storage sync
12. NoPSD
13. Partition infrastructure along team bounds
14. REST without PUT
15. Static site generators
16. Tailored Service Template

ASSESS
17. Append-only data store
18. Blockchain beyond bitcoin
19. Enterprise Data Lake
20. Machine image pipelines
21. Pace-layered Application Strategy

HOLD
22. Cloud lift and shift
23. Long lived branches with Gitflow
24. Microservice envy
25. Programming in your CI/CD tool
26. SAFe™
27. Security sandwich
28. Separate DevOps team
29. Testing as a separate organization
30. Velocity as productivity

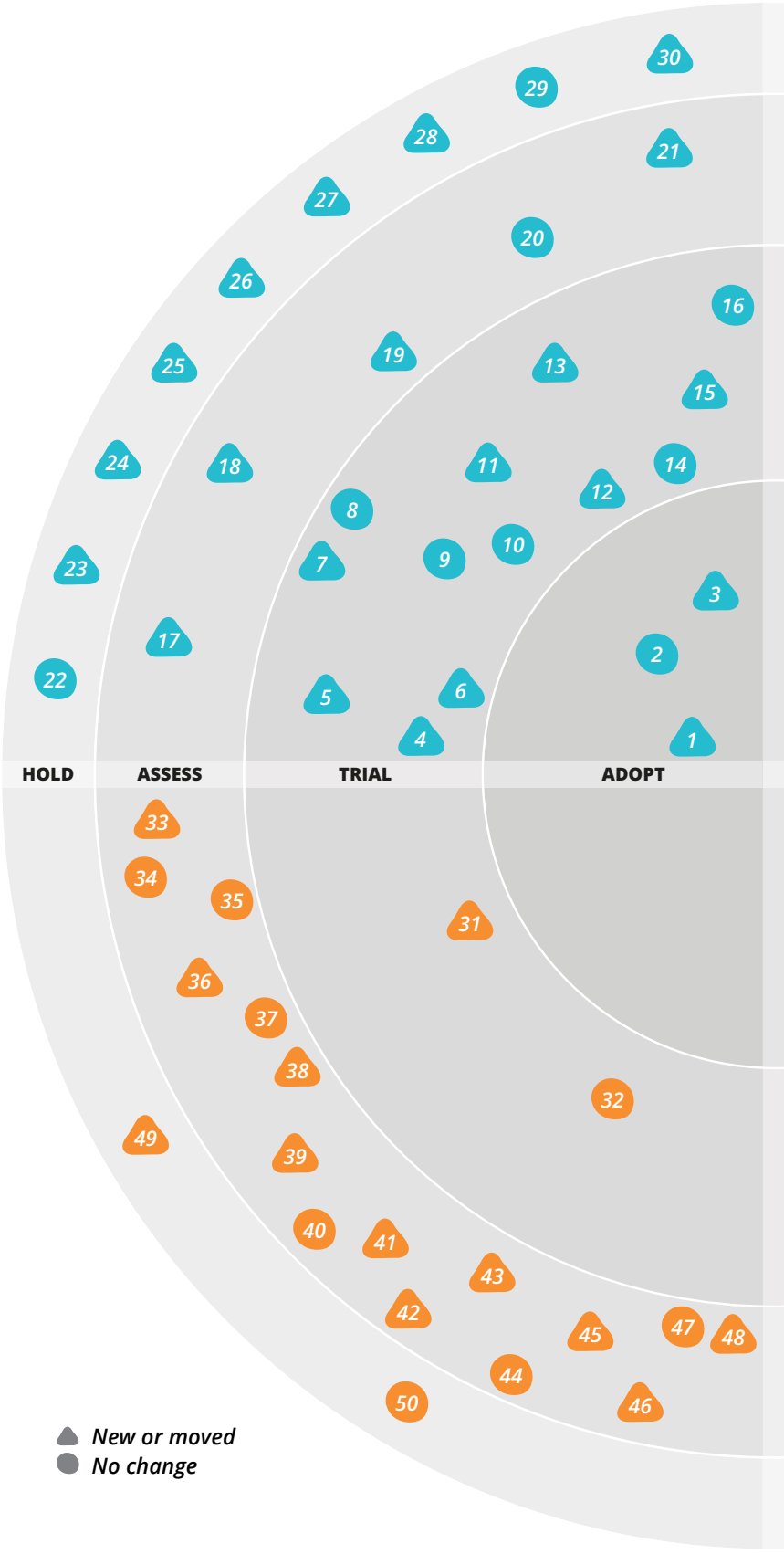
PLATFORMS

ADOPT

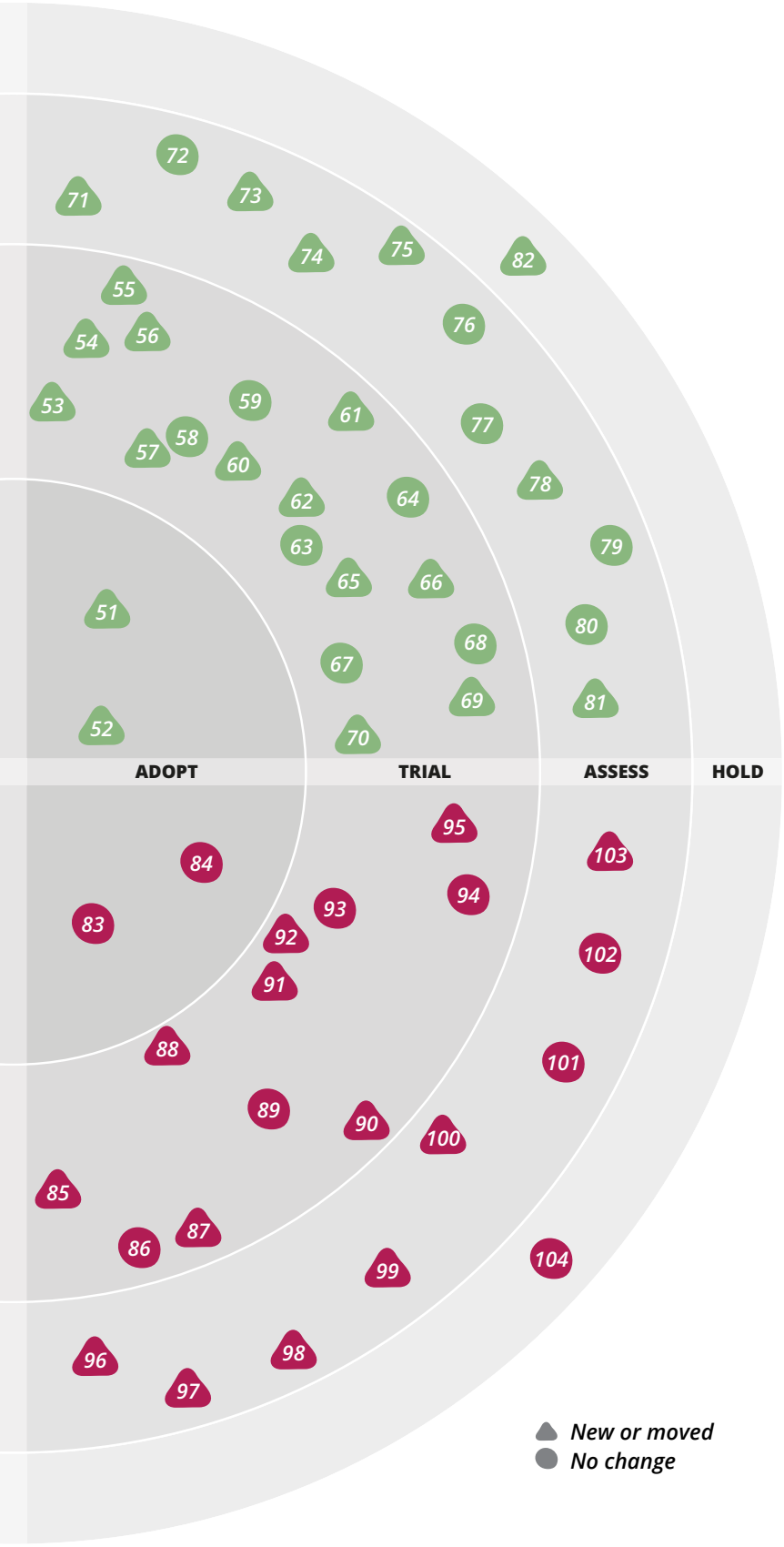
TRIAL
31. DigitalOcean
32. iBeacon

ASSESS
33. Apache Mesos
34. ARM Server SoC
35. CoAP
36. CoreOS
37. EventStore
38. Jackrabbit Oak
39. Linux security modules
40. Mapbox
41. MariaDB
42. Netflix OSS Full stack
43. OpenAM
44. OpenID Connect
45. SDN
46. Text it as a service / Rapidpro.io
47. TOTP Two-Factor Authentication
48. U2F

HOLD
49. CMS as a platform
50. OSGi



© January 2015, ThoughtWorks, Inc. All Rights Reserved.



THE RADAR

TOOLS

ADOPT
51. Flyway
52. Go CD

TRIAL
53. Appium
54. Boot2docker
55. Composer
56. Cursive
57. Docker
58. Foreman
59. GenyMotion
60. Gitlab
61. Grunt.js
62. IndexedDB
63. Packer
64. Pact & Pacto
65. Papertrail
66. Postman
67. SnapCI
68. Snowplow Analytics & Piwik
69. Swagger
70. Xamarin

ASSESS
71. Blackbox
72. Consul
73. Dc.js
74. Gorilla REPL
75. Gulp
76. leaflet.js
77. Mountebank
78. Packetbeat
79. Roslyn
80. Spark
81. Terraform

HOLD
82. Citrix for development

LANGUAGES & FRAMEWORKS

ADOPT
83. Go language
84. Java 8

TRIAL
85. AngularJS
86. Core Async
87. Dashing
88. Django Rest
89. HAL
90. Ionic Framework
91. Nashorn
92. Om
93. Q & Bluebird
94. R as Compute Platform
95. Retrofit

ASSESS
96. Flight.js
97. Haskell Hadoop library
98. Lotus
99. React.js
100. Reagent
101. Rust
102. Spring Boot
103. Swift

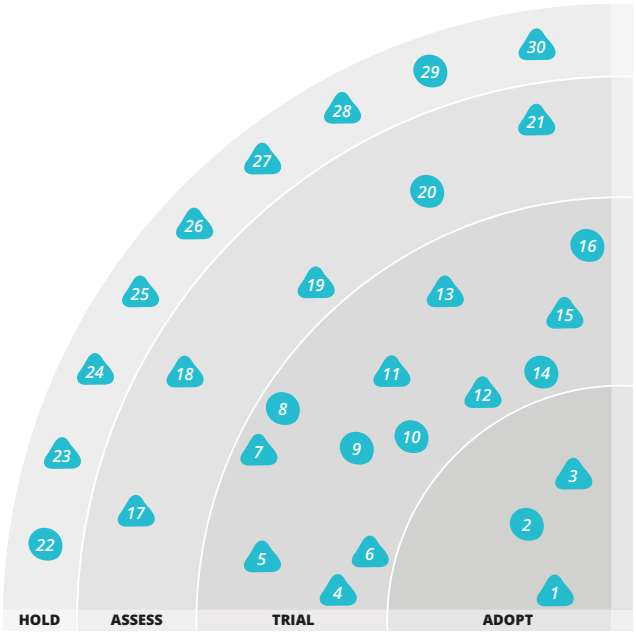
HOLD
104. JSF

© January 2015, ThoughtWorks, Inc. All Rights Reserved.

TECHNIQUES

Many projects have external code dependencies, a large amount of which is provided by open source projects. In order to ensure our builds are reproducible, we integrate against known versions of them, but that can mean that it takes a while for us to integrate against newer versions of these libraries leading to a larger merge effort down the line. One approach we have seen to avoid this is to have a nightly **Canary Build** which tries to pull in the latest version of all dependencies. If the build is green, we know we can change which versions we depend on.

The term **Datensparsamkeit** (martinfowler.com/bliki/Datensparsamkeit.html) is taken from German privacy legislation and describes the idea to only store as much personal information as is absolutely required for the business or applicable laws. Customer privacy continues to be a hot topic. Companies such as Uber are apparently collecting highly personal customer data, as well as being quite lax with security (washingtonpost.com/blogs/the-switch/wp/2014/12/01/is-ubers-rider-database-a-sitting-duck-for-hackers). This is a disaster waiting to happen. Following datensparsamkeit or using de-identification (en.wikipedia.org/wiki/De-identification) techniques even in jurisdictions where it is not legally mandated, can allow you to reduce the information you store. If you never store the information, you do not need to worry about someone stealing it.



There has been a lot of recent attention to the use of ATOM-style event feeds over HTTP as a method of integration. Instead of maintaining a live service to expose those feeds, it is often acceptable to use old-style scheduled batch processing to create and publish feed files. When combined with cloud technology like Amazon's S3 file storage and hypermedia linking, this can create a highly available, yet simple and testable solution. Our teams have started to call this old-meets-new approach '**Hipster batch**'.

When implementing single-page applications, sooner or later the question of offline use will come up. Given how hard it is to get this right when retrofitting an offline mode into an existing application, there is a trend towards implementing single-page applications with an "offline-first" mindset. An important implementation technique that we have used successfully is **local storage sync**. With this technique, the user facing code never makes requests to the backend. It retrieves data solely from local storage. A background worker synchronises the data in local storage with the backend systems, usually employing calls to some form of REST API.

ADOPT
1. Focus on mean time to recovery
2. Forward Secrecy
3. Structured logging

TRIAL
4. Canary builds
5. Datensparsamkeit
6. Front end instrumentation
7. Hipster batch
8. Humane registry
9. Inverse Conway Maneuver
10. Living CSS Style Guides
11. Local storage sync
12. NoPSD
13. Partition infrastructure along team bounds
14. REST without PUT
15. Static site generators
16. Tailored Service Templatee

ASSESS
17. Append-only data store
18. Blockchain beyond bitcoin
19. Enterprise Data Lake
20. Machine image pipelines
21. Pace-layered Application Strategy

HOLD
22. Cloud lift and shift
23. Long lived branches with Gitflow
24. Microservice envy
25. Programming in your CI/CD tool
26. SAFe™
27. Security sandwich
28. Separate DevOps team
29. Testing as a separate organization
30. Velocity as productivity

© January 2015, ThoughtWorks, Inc. All Rights Reserved.

TECHNIQUES *continued*

NoPSD (thoughtworks.github.io/p2/issue02/continuous-design) is a movement to integrate design activities into the iterative feedback cycles required to build great software. The name aims to dislodge the PSD as the final canonical design artifact rather than taking a dig at the Adobe software. Instead of signing off on a pixel-perfect design specification at the start of a project, teams are urged to embrace Continuous Design: embedding designers into delivery teams, using lo-fi techniques for prototyping, and collaborating to refine the design in the target UI technology (normally HTML and CSS). This approach speeds responding to real user feedback, allows testing designs across multiple devices and form-factors, and embraces the dynamic nature of both digital products and the product creation process.

Many of our customers have made DevOps a reality in their organization with delivery teams that build, deploy, and support their own applications and services. Unfortunately, a regular roadblock on that journey is allowing teams to have superuser privileges in production environments. In most organizations, the production environment is shared, and therefore risky to provide access widely. It is effective when we can **partition infrastructure along team bounds**, so that those teams can have safe isolated access to do their work, without risking impact to other systems. Where cloud environments are used, this is much easier to implement, aligning account structures to team boundaries.

Static site generators like Middleman or Jekyll have become popular for creating simple websites or blogs, but we are increasingly seeing their use as part of more complex application stacks. The default assumption that all content delivered over HTTP has to be dynamically created on request is shifting, with more teams looking to use static pre-generated content.

Immutable data structures are becoming more popular with functional languages such as Clojure providing immutability by default. Immutability allows code to be more easily written, read, and reasoned about. Using an **append-only data store** can confer some of these benefits in the database layer, as well as making audit and historical querying simple. Implementation options vary, from specific append-only data stores such as Datomic (datomic.com) to simply using an "append-don't-update" approach with a traditional database.

While the currency aspect of Bitcoin and other cryptocurrencies gets most of the news, we are equally excited about possibilities for using the **Blockchain beyond bitcoin** and financial transactions. The Blockchain is a mechanism for verifying the contents of a shared ledger without relying on a centralized service. We already see the Blockchain (either the underlying technology or the public Bitcoin Blockchain) being used at the heart of systems as varied as identity, ownership, record-keeping, voting, cloud storage and even managing networks of smart devices. If you are building systems that require trust over decentralized networks, then the Blockchain is a technology worth assessing.

An **Enterprise Data Lake** is an immutable data store of largely un-processed "raw" data, acting as a source for other processing streams but also made directly available to a significant number of internal, technical consumers using some efficient processing engine. Examples include HDFS or HBase within a Hadoop, Spark or Storm processing framework. We can contrast this with a typical system that collects raw data into some highly restricted space that is only made available to these consumers as the end result of a highly controlled ETL process.

Embracing the concept of the data lake is about eliminating bottlenecks due to lack of ETL developer staffing or excessive up front data model design. It is about empowering developers to create their own data processing pipelines in an agile fashion when they need it and how they need it—within reasonable limits—and so has much in common with another model that we think highly of, the DevOps model.

Gitflow is a strict branching pattern for releases using Git. Although not an inherently bad pattern, we often see it misused. If the feature and develop branches are short lived and merged often, you are really using the power of git, which makes these activities easy. However, a problem we often see is that these become **long lived branches**, which results in the dreaded merge conflicts many people began using Git to escape. A merge is a merge. Regardless of the source control tool or pattern you use. If you wait more than a day or two to merge, you could hit a big merge conflict. This becomes a real issue if you have a larger team. If you have more than a few people waiting to merge, you can have a serious a bottleneck. Introducing patterns like Gitflow require the discipline the merge often to be successful. So by all means use the pattern, but only if you have the discipline to prevent long lived branches.

© January 2015, ThoughtWorks, Inc. All Rights Reserved.

We remain convinced that microservices can offer significant advantages to organizations, in terms of improving team autonomy and faster frequency of change. The additional complexity that comes from distributed systems requires an additional level of maturity and investment. We are concerned that some teams are rushing in to adopting microservices without understanding the changes to development, test, and operations that are required to do them well. Our general advice remains simple. **Avoid microservice envy** and start with one or two services before rushing headlong into developing more, to allow your teams time to adjust and understand the right level of granularity.

We still see teams configure their CI and CD tools by directly embedding complex multi-line commands directly into the configuration of the tool. Often these embedded commands also contains steps that would only ever take effect in the build environment including things such as CI specific environment variables, steps that would create/modify files and templates only in the CI environment etc. This makes the build environment a special beast - whose results cannot be duplicated locally on a developer's machine.

This is extremely problematic because the CI/CD tool, which is supposed to expose problems in your code, itself becomes a complex beast whose behavior is hard to debug and whose results are hard to replicate.

The way to avoid **programming in your CI/CD tool** is to extract the complexities of the build process from the guts of the tool and into a simple script which can be invoked by a single command. This script can then be executed on any developer workstation and therefore eliminates the privileged/singular status of the build environment.

Scaling agile across enterprises is a continuing challenge. Several approaches have been proposed, with **SAFe™** being one gaining significant mindshare. While SAFe™ provides a useful checklist for areas of concern, in general these approaches are easy to misuse, by introducing the same kind of large release tendencies like the release train and gated control processes that agile removes. Enterprises in particular look for a degree of commonality across endeavors that SAFe™ seems to provide, promoting aggressive standardization when some degree of customization provides significant value. Other lean approaches that include experimentation and incorporate continuous improvement practices like the Improvement Katas offer organizations a better model for scaling agile.

Scaled Agile Framework® and SAFe™ are trademarks of Scaled Agile, Inc.

Traditional approaches to security have relied on up-front specification followed by validation at the end. This "**Security Sandwich**" approach is hard to integrate into Agile teams, since much of the design happens throughout the process, and it does not leverage the automation opportunities provided by continuous delivery. Organizations should look at how they can inject security practices throughout the agile development cycle. This includes: evaluating the right level of Threat Modeling to do up-front; when to classify security concerns as their own stories, acceptance criteria, or cross-cutting non-functional requirements; including automatic static and dynamic security testing into your build pipeline; and how to include deeper testing, such as penetration testing, into releases in a continuous delivery model. In much the same way that DevOps has recast how historically adversarial groups can work together, the same is happening for security and development professionals.

© January 2015, ThoughtWorks, Inc. All Rights Reserved.

PLATFORMS

Mesos (mesos.apache.org) is a platform that abstracts out underlying computing resources to make it easier to build massively scalable distributed systems. It can be used to provide a scheduling layer for Docker, or to act as an abstraction layer to things like AWS. Twitter has used it to great effect to help them scale their infrastructure. Tools build on top of Mesos are starting to appear such as Chronos (nerds.airbnb.com/introducing-chronos), which is a distributed, fault tolerant cron replacement.

CoreOS is a Linux distribution designed to run large, scalable systems. All applications deployed on a CoreOS instance are run in separate Docker containers, and CoreOS provides a suite of tools to help manage them, including etcd their own distributed configuration store. Newer services, such as fleet, help cluster management by ensuring that a specific number of service instances are always kept running. FastPatch allows atomic CoreOS upgrades using an active-passive root partition scheme and helps with quick rollback in case of problems. These new developments make CoreOS well worth looking into if you are already comfortable with Docker.

Jackrabbit Oak (jackrabbit.apache.org/oak), formerly named Jackrabbit 3, is a scalable and performant implementation of hierarchical content repository for use as the foundation of content management system. In addition to file based storage solution, MongoDB and RDBMS storage are also supported, and preferred in large volume use scenarios. Although implemented in Java, it can be easily accessed from various platforms via standards like JCR.

While server hardening is an old technique that is considered fairly commonplace by sysadmins who have had to manage production systems, it has not become commonplace among the developer community. However, the rise in the DevOps culture has resulted in renewed focus on tools like SELinux, AppArmor and Grsecurity that aim to make this simpler, at least on the Linux ecosystem. Each of these tools comes with their own strengths and weaknesses and it is a currently hard to pick one as being the only one you will need. That said, we highly recommend that all teams could at least assess which tool would be the right one for them and make security and server hardening a part of their development workflow.

ADOPT

TRIAL
31. DigitalOcean
32. iBeacon

ASSESS
33. Apache Mesos
34. ARM Server SoC
35. CoAP
36. CoreOS
37. EventStore
38. Jackrabbit Oak
39. Linux security modules
40. Mapbox
41. MariaDB
42. Netflix OSS Full stack
43. OpenAM
44. OpenID Connect
45. SDN
46. Text it as a service / Rapidpro.io
47. TOTP Two-Factor Authentication
48. U2F

HOLD
49. CMS as a platform
50. OSGii

© January 2015, ThoughtWorks, Inc. All Rights Reserved.

PLATFORMS *continued*

After Oracle's acquisition of MySQL, more and more close sourced modules are bundled into its enterprise edition. There are concerns over the future of MySQL. **MariaDB** (mariadb.org) is a community-developed GPL-only fork of MySQL intended to remain truly open source, yet fully compatible and competitive with MySQL. High-profile adopters include large-scale internet organizations Google and Wikipedia, as well as key Linux distributors RedHat and SUSE.

While we are reluctant to recommend wholesale adoption of the **Netflix OSS Full Stack** unless you happen to be entering the globally distributed video streaming business, the stack is chock full of interesting ideas, complete with open source implementations. Some of the tools, Asgard for example, are highly coupled into a virtually turnkey architecture, making them challenging to use individually. Other tools like Ice and Hystrix, which we featured on the radar previously, can be used stand-alone. We think teams should understand the ideas and approaches encapsulated within the tools even when they choose not to leverage the full stack.

When Oracle ceased development on Sun's OpenSSO—an open source access management platform—It was picked up by ForgeRock and integrated into their Open Identity Suite. Now named **OpenAM** (forgerock.com/en-us/platform/access-management), it fills the niche for a scalable, open-source platform that supports a variety of federated identity standards, including OpenID Connect and SAML 2.0. These standards are a necessary enabler for secure microservice implementations.

Software Defined Networking (**SDN**) is a broad topic, but is becoming ever more important. The ability to configure our networking devices using software is blurring the lines of where our application deployments end. It encompasses everything from virtual networking appliances like AWS's Load Balancers or CoreOS's Flannel (github.com/coreos/flannel), to networking equipment that supports standards like OpenFlow (opennetworking.org/Openflow). Where cloud providers have previously focused on compute and storage, we expect the growing array of SDN tools to deliver further efficiencies to how we handle our systems both off and on premise.

Text-it-as-a-service / Rapidpro (rapidpro.io) offers ability to easily set up or modify complex short message service application for business without extensive need of a developer. With the lower costs of text messages compared to USSD sessions, this provides a more affordable way to build scalable applications targeting feature phones and we have seen success in our projects. Flows are very simple to build and actions can be triggered at any point such as sending an sms, email or even calling an external api.

Securing online accounts is at the same time extremely important and notoriously difficult. Two-factor authentication does greatly increase security and we have recommended TOTP as a good solution. A new entrant in this field is Universal 2nd Factor (**U2F**), a solution based on public key cryptography and inexpensive USB hardware tokens. While developed at Google, it has now become a standard managed by the FIDO Alliance. We do like the promise of better protection against phishing and man-in-the-middle attacks, but are concerned because the standard currently references a specific elliptic curve digital signature algorithm that is considered to be flawed.

© January 2015, ThoughtWorks, Inc. All Rights Reserved.

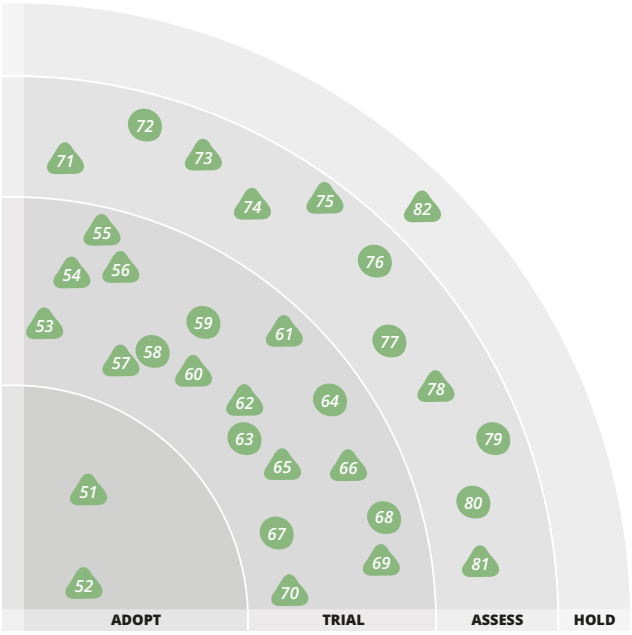
TOOLS

With techniques such as continuous delivery becoming more mainstream, automated database migrations are a baseline capability for many software teams. While there are many tools in this space, we continue to recommend **Flyway** for its low-friction approach. Flyway has a vibrant open-source community behind it, and support for both traditional and cloud-based databases such as Amazon Redshift and Google Cloud SQL.

Continuously delivering high quality software to production in a rapid and reliable manner requires coordinating many automated steps. **Go CD** (go.cd) is an open-source tool built by ThoughtWorks to handle exactly this scenario, with the concept of deployment pipelines at its core, it handles complex workflows over many nodes and enables transparent, traceable promotion of trusted artifacts across environments. While it is possible to craft deployment pipelines on top of continuous integration tools, our teams see the benefit derived from a tool purpose built for this job.

Boot2docker is a lightweight linux distribution running Docker, packaged as a VM for OSX and Windows. This is a great way to get started experimenting with Docker. For teams using microservices, it can also be an effective way to run multiple services on a local machine for dev and test purposes, where the overhead of multiple vagrant VMs may be too much.

Although the idea of dependency management is not new and considered to be a fundamental development practice, it is not widely adopted by the PHP community. **Composer** (getcomposer.org) is a tool for dependency management in PHP. It is strongly influenced by tools from other technology stacks like Node's npm and Ruby's Bundler.



Cursive (cursiveclojure.com) is a Clojure IDE that works as a plugin for IntelliJ. While still in early access, we have found it very useful when working with larger Clojure codebases. Cursive provides strong renaming and navigation support, has shown itself to be stable and reliable, and is great for environments with mixed JVM languages. For organizations adopting Clojure, Cursive has helped lower the barrier to entry for existing developers.

ADOPT
51. Flyway
52. Go CD

TRIAL
53. Appium
54. Boot2docker
55. Composer
56. Cursive
57. Docker
58. Foreman
59. GenyMotion
60. Gitlab
61. Grunt.js
62. IndexedDB
63. Packer
64. Pact & Pacto
65. Papertrail
66. Postman
67. SnapCI
68. Snowplow Analytics & Piwik
69. Swagger
70. Xamarin

ASSESS
71. Blackbox
72. Consul
73. Dc.js
74. Gorilla REPL
75. Gulp
76. leaflet.js
77. Mountebank
78. Packet beat
79. Roslyn
80. Spark
81. Terraform

HOLD
82. Citrix for development

© January 2015, ThoughtWorks, Inc. All Rights Reserved.

TOOLS *continued*

Gitlab is an on-premise Git repository hosting platform that gives proprietary software development teams the familiar and ubiquitous workflow that hosted version control services like Github and BitBucket provide OSS developers. While it is available as free community edition software, the commercial enterprise option provides support and deep integration with LDAP servers.

As single page applications and offline-first become more viable and widespread there is a growing need to persist data in the web browser. Local Storage is very easy to use and well supported by the web browsers. For more complex use cases, there is **IndexedDB**. While it can be a good solution we recommend to only use it when absolutely necessary, due to the increase in complexity and a somewhat clumsy API. We have also had positive experience with the LocalForage (github.com/mozilla/localForage) framework that provides an abstraction layer over the various persistence solutions.

Postman (www.getpostman.com/features) is a Chrome extension that acts as a REST client in your browser, allowing you to create requests and inspect responses. It is a useful tool when developing an API or implementing a client to call an existing API. It offers a suite of extensions that allow you to use it as a full-blown test runner too, although we discourage the record and replay style of testing it promotes.

Blackbox (github.com/StackExchange/blackbox) is a simple tool for encrypting specific files while at rest in your source repository. This is particularly useful if you need to store passwords or private keys. Blackbox works with Git, Mercurial and Subversion and uses GPG for the encryption. Each user has their own key, which makes it easy to revoke access on a granular level.

We have recommended D3.js before and in this radar we want to extend our recommendation to **Dc.js** (dc-js.github.io/dc.js), a charting library based on D3 for exploring large multi-dimensional datasets. With D3, it shares the ease with which beautiful interactive graphs can be created. It is different in that it trades the flexibility to create almost any kind of data visualization for a simpler programming model to create common chart types.

GorillaREPL (gorilla-repl.org) is a tool for creating nicely-rendered documents consisting of text, live Clojure code, and plots. In some ways similar to iPython notebooks, GorillaREPL should be particularly useful for data analysts or code tutorials. But beyond that, GorillaREPL is fun!. It is a creative way to demonstrate the power of Clojure's simple abstractions over immutable values.

As distributed systems become more complex, it can be useful to have tools that help you understand how your system is behaving in production. **Packetbeat** (packetbeat.com) is an open source tool which uses agents to sniff traffic between nodes, allowing you to see traffic patterns, error rates and other useful information. It requires Elasticsearch (elasticsearch.org/overview/elasticsearch) and Kibana (elasticsearch.org/overview/kibana) to work, but if you are already using these tools as part of log aggregation, it could be an easy drop-in to give you more insight into your production system.

With **Terraform**, cloud infrastructure can be managed by writing declarative definitions. The configuration of the servers instantiated by Terraform is usually left to tools like Puppet, Chef, or Ansible. We like Terraform because the syntax of its files is quite readable and because it supports multiple cloud providers while making no attempt to provide an artificial abstraction across these providers. At this stage, Terraform is new and not everything is implemented yet. We have also found its state management to be fragile, often needing awkward manual work to untangle.

For security and compliance reasons, offshore teams are sometimes asked to use **Citrix** to connect to an onshore virtual desktop, where they do **development**. While a good tool for some use cases, Citrix provides an extremely poor remote development experience and often cripples an offshore team. There are many better technical solutions, such as the NoMachine remote desktop or Cloud9 IDE, which can provide a more workable experience. An even better solution is to tackle the underlying security and compliance concerns. Since you are trusting the remote team to work on your source code and check in to your code repository, you should try to get to a point where you also trust them to have source code on their machines. They will be much more productive!

© January 2015, ThoughtWorks, Inc. All Rights Reserved.

LANGUAGES & FRAMEWORKS

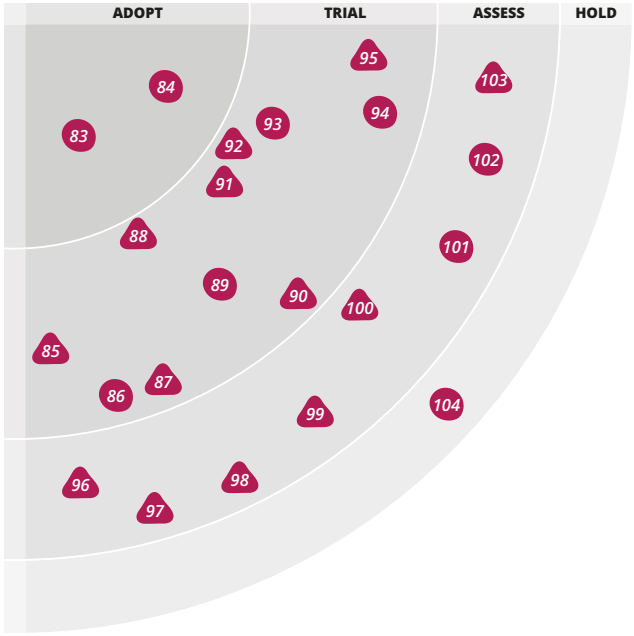
The importance of big, visible displays in team areas has been written about many times before, and we certainly value the approach of helping everyone see and understand key pieces of information about how our software or our teams are doing. **Dashing** (dashing.io) is a ruby-based dashboard system we have been using for many years to create clear, visible displays optimised for large monitors. It is very hackable, allowing you to pull in information from a variety of sources from build systems, ticket or story tracking tools, or production monitoring systems.

We have used the **Django Rest Framework** (django-rest-framework.org), which is a flexible and customizable framework that makes it easy to build web APIs, in several of our projects. It allows you to build restful APIs in python with django, exposing API endpoints which are accessible from a consumer front-end. Django rest gives a browsable web API that allows developers to visualize data being transferred through the API and returns response examples, which the consumer application will receive. It provides a number of authentication schemes out of the box, and allows implementation of custom schemes.

Ionic Framework (ionicframework.com) is an open-source front-end framework that offers a library of mobile-optimized html, css and javascript components and tools for building highly interactive applications. It is built with SASS and optimized for AngularJS. We have seen success in several of our projects employing this framework, with its ease to install and test. We recommend investigating this framework when you are performance obsessed and looking for a seamlessly integrated front-end framework.

Nashorn is a new JavaScript engine for Java that has been released with Java 8. When the exact same code should be run in the web browser and on the server, which is often the case for validation and data migration logic, it is the tool of choice in the Java world, and that is the case despite some rough edges. We are not convinced that using Nashorn to host entire applications, via Node support or the Avatar project, is a good idea.

Retrofit (square.github.io/retrofit) offers a reliable way to build http clients on android projects by converting a rest API into a Java interface. Retrofit intergrates with okhttp and allows developers to provide custom error handling for requests. It does JSON parsing automatically using GSON and has a very well supported community.



ADOPT
83. Go language
84. Java 8

TRIAL
85. AngularJS
86. Core Async
87. Dashing
88. Django Rest
89. HAL
90. Ionic Framework
91. Nashorn
92. Om
93. Q & Bluebird
94. R as Compute Platform
95. Retrofit

ASSESS
96. Flight.js
97. Haskell Hadoop library
98. Lotus
99. React.js
100. Reagent
101. Rust
102. Spring Boot
103. Swift

HOLD
104. JSF

© January 2015, ThoughtWorks, Inc. All Rights Reserved.

LANGUAGES & FRAMEWORKS *continued*

In the crowded space of JavaScript frameworks, we want to highlight **Flight.js** (flightjs.github.io) as an alternative to consider. Flight is extremely lightweight and gets by without much magic when adding behavior to DOM nodes. Its event-driven and component-based nature promotes writing decoupled code. This makes testing individual components comparatively easy. Care must be taken, however, when components need to interact with each other. There is little support for testing and a real danger to get into event hell. We do like that it uses functional mixins for behaviour, like composition instead of inheritance.

While there are lots of fans of **Haskell** among ThoughtWork's language devotees, we rarely see it on the kinds of projects we work on—until recently. Several open source projects now marry **Hadoop**'s map/reduce jobs to Haskell's syntax, which some developers and/or data scientists find appealing.

We don't know who named **Lotus** (lotusrb.org), but we can only assume they are too young to have worked with a certain office collaboration product. Lotus is a new Rack-based MVC framework written in Ruby that can be deployed modularly so that you are free to use only the portions of the framework you need. It is a modern alternative to the monolithic Ruby-on-Rails framework (that turned 10 this year). Lotus has the potential to make full-stack Ruby MVC development as easy as 1-2-3.

One benefit to the ongoing avalanche of front-end JavaScript frameworks is that occasionally, a new idea crops up that makes us think. **React.js** is a UI/View framework in which JavaScript functions generate HTML in a reactive data flow. Although we are wary of mixing code and markup, it results in UI components that are nicely encapsulated and composable. React.js is getting a lot of developer attention and will benefit from more tools and examples becoming available.

Reagent (holmsand.github.io/reagent) has emerged as a lightweight minimalist alternative to Om for wrapping React.js in ClojureScript. Whereas Om provides a comprehensive Clojure-idiomatic front-end programming framework, Reagent takes advantage of Clojure's expressiveness to focus on simple components and a readable DSL for writing HTML. By representing HTML in Clojure data, Reagent retains the performance and understandability of React.js without embedding foreign markup in the code.

Swift (apple.com/swift), Apple's new development language, contains many improvements over the perennial Objective-C, including emphasis on functional programming and modern syntax. In most ways, this is an upgrade if you are coding on the Apple platform.

ThoughtWorks is a software company and community of passionate, purpose-led individuals that specialize in software consulting, delivery and products. We think disruptively to deliver technology to address our clients' toughest challenges, all while seeking to revolutionize the IT industry and create positive social change. We make pioneering tools for software teams who aspire to be great. Our products help organizations continuously improve and deliver quality software for their most critical needs. Founded over 20 years ago, ThoughtWorks has grown from a small group in Chicago to a company of over 3000 people spread across 30 offices in 12 countries: Australia, Brazil, Canada, China, Ecuador, Germany, India, Singapore, South Africa, Uganda, the United Kingdom, and the United States.

ThoughtWorks®

© January 2015, ThoughtWorks, Inc. All Rights Reserved.