redhat

Finding Flexibility with Microservices, Containers, and Runtimes

FEATURING RESEARCH FROM FORRESTER

How To Capture The Benefits Of Microservice Design

Finding Flexibility with Microservices, Containers, and Runtimes

IN THIS DOCUMENT

FINDING FLEXIBILITY WITH MICROSERVICES, CONTAINERS, AND RUNTIMES

An introduction to the Forrester analyst report: How to Capture the Benefits of Microservice Design

There has always been a relationship between an application and the platform and services which run it. For a long time, that relationship was very tight and encompassed a lot of areas related to the application design, from the language the application could be written in to administrative behaviors like monitoring and logging, even things like transaction management, frontend UI development, or integration methods.

In a sense, the *application platform* was just another design consideration for the application. A few years ago, as significant changes in platform technology like cloud, containers, and virtual machines started rolling out, the primary emphasis for CTOs and application architects was on how to move to those platforms. The platform choice was a major part of the application. The focus was — one way or another — getting into "the cloud."

There's been a shift in customers' priorities over the last few years. A 2016 survey of IT professionals by Red Hat found that fewer than 25% of respondents were actually focusing on moving applications to the cloud — the majority were focused on writing new applications or maintaining existing ones. The focus is moving from platform to the application itself.

That's even changing how they look at applications themselves. While 64% are running traditional Java EE applications today, about a third expect to be running a mix of traditional Java EE and cloud native apps within the next year and another 38% expect to be running applications exclusively in the cloud. The change in the platform is not the goal; it is the result of writing new cloud-native applications.

FOCUS ON THE APPLICATION MORE THAN THE INFRASTRUCTURE

Applications have to be developed, tested, and released more quickly. Methodologies like agile and DevOps break down the size of the code delivered to help speed up how quickly teams iterate through features and new services. This has helped developers and architects reimagine their application design — and that has led to a natural, incremental introduction of microservices.

Traditional monolithic applications haven't truly been monolithic in decades. Certain critical functionality has long been abstracted into separate applications, such as SQL databases or JavaScript UIs serving a Java application.

It's been a natural move to start breaking out new services rather than trying to add features to an existing monolith, using whatever language and framework makes the most sense for the specific service. Forrester calls this a pragmatic approach — using the precise tool for a precise job. New platform and deployment options have made that easier, because cloud instances and containers are designed for small, lightweight, easy to deploy applications. Deploying, scaling, and retiring services is easy without wasting computing resources.

APPLICATION DESIGN INFLUENCES PLATFORM REQUIREMENTS — NOT THE OTHER WAY AROUND

Entire IT departments used to be defined by the application platform they ran, such as Java shops, C/C++, or COBOL. Putting the application first has given new autonomy — and creativity — to developers. This means IT organizations need to shift how they evaluate and what's important in an application platform. Rather than trying to find developers or application architectures that conform to their existing infrastructure, IT managers should look at providing tools that allow for developer independence.

There are three primary characteristics for a strategic, adaptive application platform:

- **Hybrid infrastructure.** As services change, the environments they run in change. An application should function the same in physical hardware, virtual machines, in public clouds like Amazon and Azure, or in private clouds like OpenStack. That means that the development environment should create applications that are portable.
- **Multiple runtimes and languages.** While Java remains the #1 programming language for enterprise applications, it is no longer likely to be the sole language in an organization. An application environment needs to support multiple languages simultaneously, both to create new, independent services and to help backport and extend existing applications. Supporting multiple languages allows developers to choose the best technology for a service.
- **Container image catalogs.** One of the chief problems with distributed applications is the increase in complexity. Maintaining consistency, even orchestrating related services, can be incredibly difficult. Containers provide immutable images, which allow consistency and quality across development and production environments. Container administration tools like image catalogs, orchestration services, and monitoring make it much easier for developers and operations teams to manage applications and control the inherent complexity of distributed architectures.

Red Hat provides a full stack application platform with Red Hat OpenShift Application Runtimes, which has the container platform and management of Red Hat OpenShift Container Platform and different runtimes, including Red Hat JBoss Enterprise Application Platform, Wildfly Swarm, Vert.x, Node,js, and even Spring. JBoss EAP is a full Java EE certified platform; Wildfly Swarm is an implementation of Eclipse Microprofile, which is a microservices-oriented spec with a subset of the Java EE libraries. These offer an entire spectrum of Java development environments, from more traditional Java applications to microservices. In addition, there is a full set of developer tools and application services such as integration and business logic. That full stack of developer and application tools gives more freedom to developers and helps organizations make existing applications cloud-ready and innovate with cloud-native applications.

FORRESTER®

FOR APPLICATIONS DEVELOPMENT & DELIVERY PROFESSIONALS

How To Capture The Benefits Of Microservice Design

Three Phases Of Adoption Begin With Today's Programming Models And Platforms

by Jeffrey S. Hammond and John R. Rymer

May 26, 2016

Why Read This Report

Developing software using microservice design is the hot new approach. Why? Microservices promise implementation flexibility, graceful scaling, faster delivery, and higher resiliency. But comprehensive microservice architectures are too complex for most enterprise AD&D teams, so developers are finding pragmatic and incremental ways to gain the benefits of microservice design via new programming models and platform frameworks. Read this report to learn about these emerging methods and their impact.

Key Takeaways

Developers Find Pragmatic Microservices
Pure, comprehensive development approaches based on microservice architecture are too difficult for most enterprises to quickly adopt. Developers are instead adopting aspects of the new approach through new, microservices-inspired programming frameworks and cloud platform services.

Microservices Are Remaking App Platforms
Platform vendors are adding container services that support the deployment of microservices. At the same time, a new generation of application platforms with native support for microservice designs and deployment is arriving in the form of new platform-as-a-service options or as components from which teams can build their own platforms.

How To Capture The Benefits Of Microservice Design

Three Phases Of Adoption Begin With Today's Programming Models And Platforms

by Jeffrey S. Hammond and John R. Rymer
with Christopher Mines, Randy Heffner, Dave Bartoletti, Claudia Tajima, and Rachel Birrell

May 26, 2016

Table Of Contents

Notes & Resources

Forrester interviewed 10 AD&D pros working with microservices and spoke with Cloudsoft, Google, Mesosphere, Microsoft, Oracle, Pivotal Labs, Red Hat, Salesforce, and Weaveworks.

Related Research Documents

Boost Application Delivery Speed And Quality With Agile DevOps Practices

The Dawn Of Enterprise JavaScript

Microservices Have An Important Role In The Future Of Solution Architecture

Vendor Landscape: Public Cloud Platforms Consolidate, But New Disruptions On The Way

FORRESTER®

Forrester Research, Inc., 60 Acorn Park Drive, Cambridge, MA 02140 USA
+1 617-613-6000 | Fax: +1 617-613-5000 | forrester.com

© 2016 Forrester Research, Inc. Opinions reflect judgment at the time and are subject to change. Forrester®, Technographics®, Forrester Wave, RoleView, TechRadar, and Total Economic Impact are trademarks of Forrester Research, Inc. All other trademarks are the property of their respective companies. Unauthorized copying or distributing is a violation of copyright law. Citations@forrester.com or +1 866-367-7378

Microservices Hold Great Promise, But Complexity Hinders Adoption

As developers rise to the challenges of building applications in the age of the customer, microservice designs are prominent in their thinking.[1] A *microservice* is a software component that does one thing (and only one thing) well, is self-contained, and communicates with other microservices through APIs or messages independent of the specific programming language.[2]

THE POTENTIAL BENEFITS OF MICROSERVICE DESIGNS

Developers use the term *microservices* to signify practices and technologies aimed at rapid-fire application delivery for business innovation and responsiveness — specifically (see Figure 1):

› **Speeding app delivery.** A key principle of modern application delivery is to deliver software faster and reduce the size of code modules to contain the risk of bugs and/or failures. Microservice designs arose from this development practice. A knock-on effect of this strategy is loose coupling between code modules, which interact via APIs. Thus, reusing APIs to speed development is now another way to achieve this benefit.

› **Enabling quick, incremental app changes.** The loose coupling of small software components also improves the odds that developers will be able to change one service without breaking other services that depend on it. At scale, this allows organizations to release small changes many times a day.[3] Amazon used microservice-centric delivery pipelines to release over 50 million changes in 2014 alone, an average of two per second.[4]

› **Ensuring resiliency at scale.** Independent microservices not only enable incremental, isolated changes but also allow other parts of a system to keep going when a microservice goes down. Netflix uses a collection of simian-themed testing tools that test microservice resiliency in production by randomly killing infrastructure components in order to quickly isolate and eliminate hard-coded dependencies.[5]

› **Optimizing production costs.** Microservice designs raise the density of software deployments by packing more into either virtual machines or containers. Further, loosely coupled, independent components are easier to install and shut down as needed, incurring costs only when the software runs.

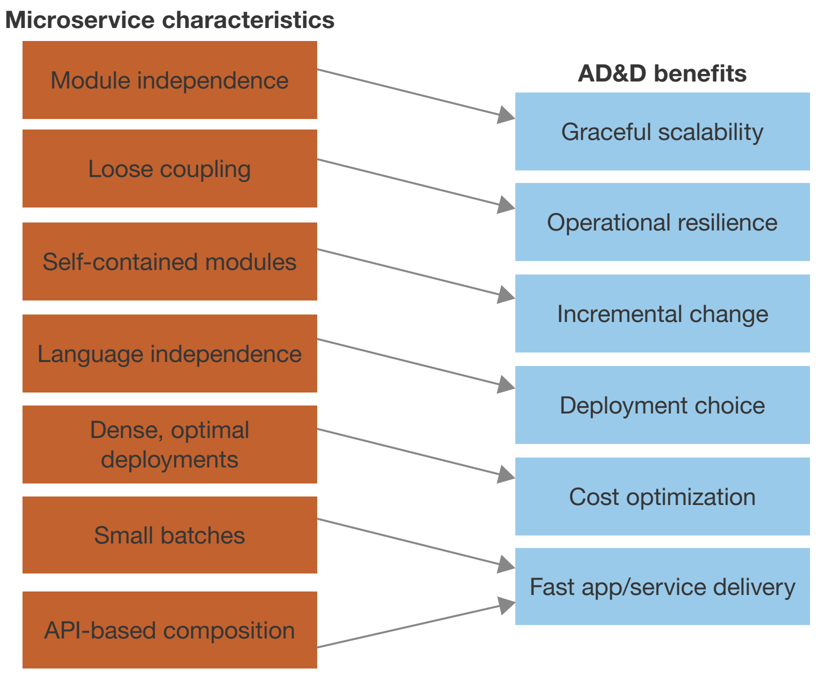
› **Supporting graceful scaling.** Apps structured as independent services — particularly those using stateless designs — allow scaling with consistent performance on commodity hardware simply by adding new instances. Developers love the potential to scale up easily from small to massive workloads without having to either rewrite their apps or employ expensive infrastructure.

› **Freeing developers to use the best tool for the job.** Independent microservices that interact via APIs also allow developers to select the best language and framework for any particular service.

FORRESTER®

© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

FIGURE 1 A Developer's View Of The Characteristics And Benefits Of Microservices



FOUR MAJOR CONSTRAINTS ADD COMPLEXITY AND DETER ADOPTION OF MICROSERVICE DESIGNS

To realize these potential benefits of microservices, developers must overcome four major constraints. Each of these adds complexity to current application development environments and practices, making for a challenging transition. Overcoming all four constraints simultaneously is a Herculean task.

› **Microservices create a dependency-management challenge.** Most developers agree on what microservices are, at least in theory. But there's little agreement about the architectures and practices that manage design and runtime dependencies between microservices and change management in the widely distributed applications that microservice designs enable.

"We're trying to break some habits. You've got to treat every dependency as third-party integration — even if you own it. Also, expect to deal with backwards and forwards compatibility." (Director of cloud engineering, large financial services firm)

› **Continuous integration and delivery is difficult to scale.** Simply speeding up existing release management processes is a recipe for disaster and destabilization. The ability to change apps incrementally with minimal risk to dependents rests on good overall design, architecture, and delivery processes that allow independent releases. Teams that succeed employ ruthless automation, simplify staging, control traffic flow, use feature and version flags, and organize parallel delivery pipelines around individual microservices. Don't expect these capabilities to mature overnight.

FORRESTER®
© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

› **Microservice composition requires new API and messaging technologies.** In order for their singular focus to work in larger applications, microservices must be capable of asking other microservices for help as well as servicing others' requests in turn. The most common ways developers accomplish this goal? Using RESTful APIs and JSON payloads or pulling messages off a queue or event bus. For example, Netflix developers use the Ribbon library to service and load-balance interprocess requests among microservices.[6]

› **Internet-scale applications on public cloud platforms require new platforms.** Microservice leaders like AWS, Google, and Netflix operate at massive global scale and need software and infrastructure architectures that do, too. Public cloud platforms may offer massive pools of compute and storage, but taking advantage of that capacity while maintaining reliable operations and optimal costs requires new application structures, service runtimes, and platform services capable of managing massively distributed applications. Adopting stateless application architectures alone is not enough.

Three Ways To Overcome Microservices' Complexity

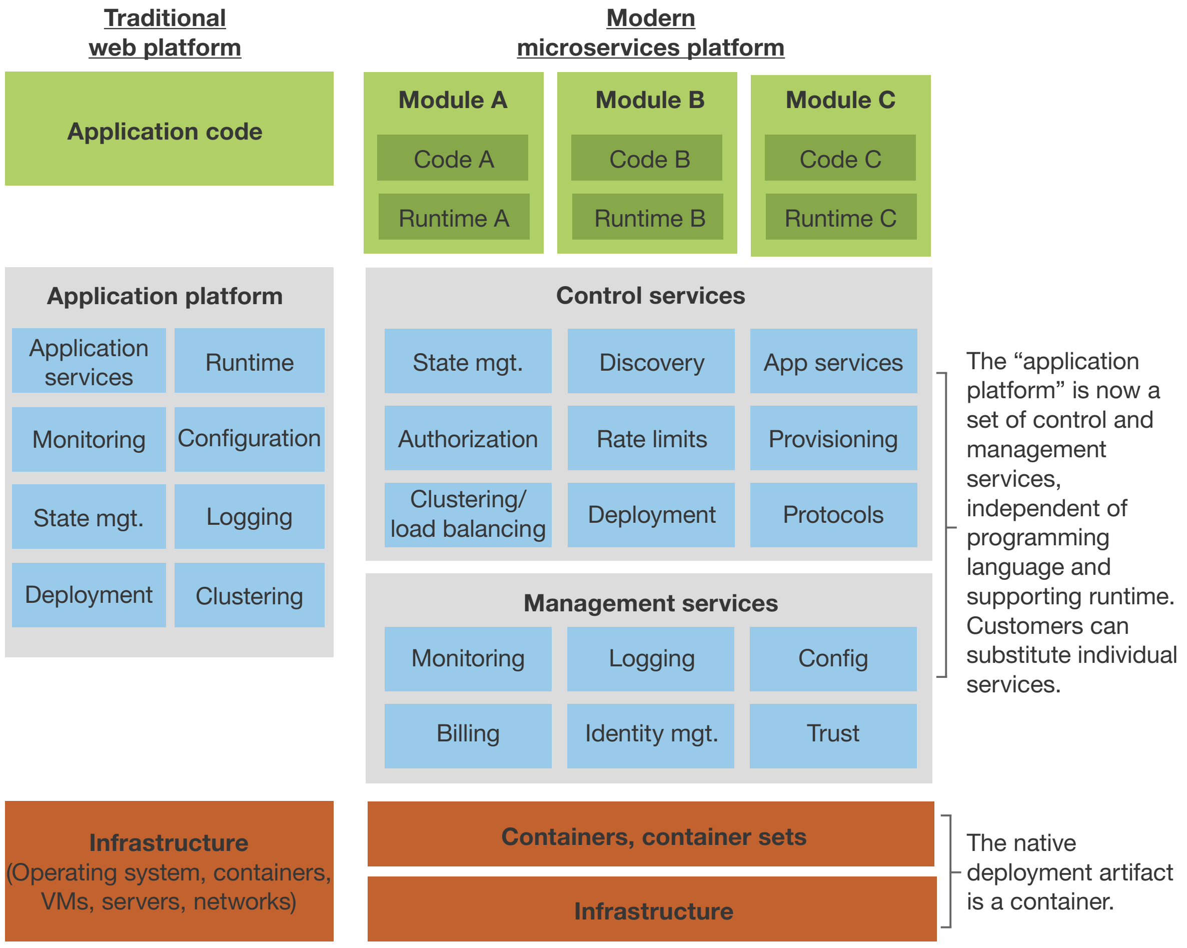
Motivated developers will always find a way to move forward. Pressed to deliver their immediate products rather than invent entirely new enterprise architectures from scratch, developers are finding pragmatic, incremental ways to capture the benefits associated with microservice designs by:

1. Adapting today's programming models and platforms to microservice designs.
2. Adopting new programming models on today's platforms.
3. Adopting new programming models and new microservice platforms.

This is the same progression that microservice innovators like Google and Netflix followed. The first movers started out using conventional programming models and platforms. As they deepened their commitments to microservice designs, they embraced containers — principally Docker — rather than virtual machines (VMs) for deployment.[7] They also either invented or adopted new platform services to host, connect, compose, scale, and manage their microservices (see Figure 2). Enterprise AD&D teams have the benefit of following the pioneers' trail.

© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

FIGURE 2 Web Platforms Versus Microservice Platforms



Adapting Today's Programming And Platforms To Microservices

Today's typical application platforms, including the web programming models they support, accommodate microservice development. For example, developers can choose to constrain the designs of Java modules to do only one thing well, but they'll still have to deploy that Java code and its configuration as an archive file into a conventional Java app server.

> "You naturally start to break your code base down into smaller services." (Director of cloud engineering, large financial services firm)

Developers can also choose to deploy their applications (either conventional C#, Java, etc., or new microservices) on virtual machines without using containers (as many do when using cloud infrastructure services). The major cloud platform vendors now also provide independent container

FORRESTER®
© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

services that developers can use instead of VMs. While pragmatic, these accommodations of microservice designs and deployments on existing platforms limit the benefits for developers to only fast delivery and incremental change. Why?

› **Language-independent runtimes are scarce.** Application servers tend to at least favor a limited set of programming languages.[8] Java application servers support many languages but are too bloated for a microservice approach.

› **Bloated runtimes raise costs.** Unless they're virtualized, web app servers assume they've got their own "server" (a VM, core, or actual server) to play on — and ample compute and storage resources.[9] If an application requires many thousands of microservice instances, this runtime configuration — one service; one app server instance; one VM, core, or server — will generate expensive bills. Better to run instances only when actually needed, something conventional application servers can't do. For one Azure public-sector customer we spoke with, a microservice approach reduced its peak demand from 8,000 VMs to about 1,000.

› **Runtimes host interdependent services, reducing resilience.** The typical web application comprises many interdependent services — designs that require all services, including the application server runtimes, to be running to support resilience. This is the opposite of applications made of loosely coupled independent services that can route around a service failure to keep running. Microservices favor much more compact runtimes that can be quickly and reliably started, stopped, relocated, and changed.

› **There's a bias toward stateless services.** Stateless services are ideally suited to the elastic scaling of cloud platforms, and so application platforms like the various Cloud Foundry implementations are almost always used for stateless services and apps. Developers store state in external services, which is a big change from the shared-state designs common in MVC applications; this adds complexity to their application when scaling.[10]

Adopting New Programming Models On Today's Platforms

As developers' adoption of microservices expands, so does their experimentation with new microservice platforms, frameworks, and approaches to design and delivery. Early favorites are already emerging. These new programming models are the vanguard of the microservice revolution.

FUNCTIONAL PIPELINES EASE THE PROCESSING OF DATA AT SCALE

In 2013, we observed that modern (mobile) applications were powered by programming patterns that were quite different from the *model-view-controller* (MVC) implementation patterns used in traditional .NET and Java applications.[11] Fast forward to today, and we find public cloud providers like Amazon, Google, IBM, and Microsoft supporting developers by automating the *pipes-and-filters* pattern with functional snippets of code triggered on demand by services, HTTP endpoints, or in-app activity.

© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

Developers use these functional bits of code to filter the data that comes in and then trigger other filters (see Figure 3). In the process, they build complex pipelines of events that handle unpredictable data flows of variable scale and timing, without any thought to the underlying server infrastructure. Development teams love this *functional pipeline* programming style as the code is:

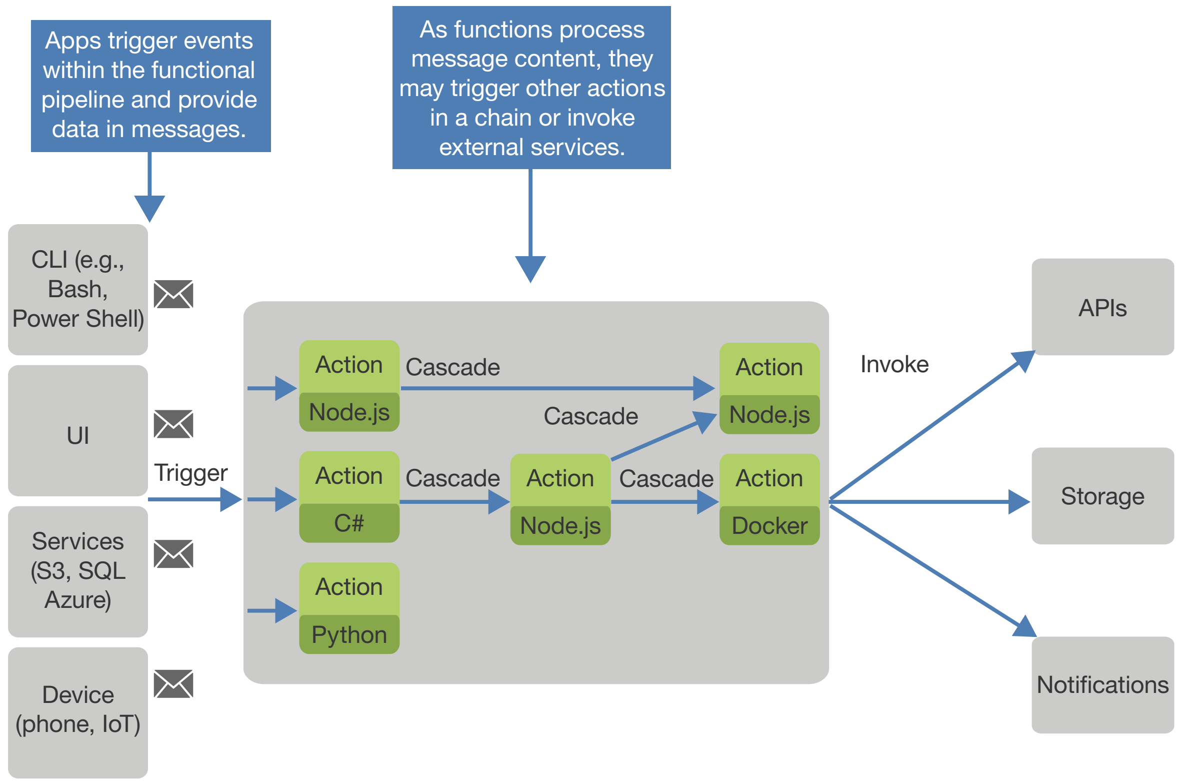
› **Simple to create.** When Amazon introduced AWS Lambda in 2013, its CTO Werner Vogels used the concept of a Microsoft Excel macro as a metaphor to communicate the simplicity of its programming model.[12] Writing the equivalent of a macro or regular expression is a lot less work than setting up an app server, rules engine, or business process server just so a developer can execute a few dozen lines of code.

› **Quick to deploy.** Developers deploy code snippets without regard to VMs, containers, or service configurations. Multiply the effort to create one filter by hundreds or thousands of times, and it's easy to understand why the early adopters we've spoken with have jumped in with both feet. Developers increasingly refer to this abstraction from underlying infrastructure as "serverless" — meaning they don't have to care about all the infrastructure and code under their filters, but those filters are properly executed when environmental events occur.

› **Cheap to use.** Developers at digital agencies and startups also love functional pipelines because they get billed only when their filters actually run, not for infrastructure whether it's used or not. As a result, the execution cost of their application infrastructure is an order of magnitude less than deploying and managing complex event processing runtimes or business rule servers inside a VM-based infrastructure. Functional pipelines also allow developers to experiment with applications where events are only occasionally triggered because they only pay for execution time, not time spent listening for messages.

FORRESTER®
© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

FIGURE 3 A Functional Pipeline Event Flow


Apps trigger events within the functional pipeline and provide data in messages.
As functions process message content, they may trigger other actions in a chain or invoke external services.
CLI (e.g., Bash, Power Shell)
UI
Services (S3, SQL Azure)
Device (phone, IoT)
Trigger
Action
Node.js
Cascade
Action
Node.js
Cascade
Action
C#
Cascade
Action
Node.js
Cascade
Action
Docker
Action
Python
Invoke
APIs
Storage
Notifications

THE ACTOR MODEL SEES RENEWED INTEREST AS CLOUD CONCURRENCY CHALLENGES GROW

The *Actor* message-passing model traces its origins to a 1973 paper.[13] It envisioned "the prospect of highly parallel computing machines consisting of dozens, hundreds, or even thousands of independent microprocessors, each with its own local memory and communications processor, communicating via a high-performance communications network."[14] In some ways, mainstream development and scale-out public clouds have caught up with yesteryear's systems design theory. As scale-out complexity grows ever more painful, small sets of knowledgeable developers leverage the Actor model for help. Microsoft's Halo 4 cloud services team did so when it turned to the "Orleans" project from Microsoft Research in 2011 (see Figure 4).[15] Why? The Actor model:

› **Makes complex parallel processing tasks simpler.** Traditional shared-state object models that employ multithreading for concurrency are almost impossible for mere mortal developers to manage at scale. Actors don't share state, and they implement concurrency via asynchronous message passing. Since there's no shared state, there's no need to lock actors to update them or synchronize processing threads. The bottom line: A much larger proportion of the developer population finds the model approachable compared to writing multithreaded code within object-oriented frameworks.[16]

© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

- **Provides building blocks for resiliency.** While an Actor implementation deals with concurrency well, it can still prove unreliable if individual actors in a system go down or the network impedes message flow across VMs. New platforms help by adding redundancy and failover for developers. Azure Service Fabric creates “virtual actors” that abstract multiple actual actors on different VMs and manage consistency between them; if one goes down, the others in the collection pick up the slack. Lightbend’s Apache-licensed Akka framework uses a reliable proxy pattern to ensure that messages sent across VMs have the same quality of service as those sent between actors inside a VM.
- **Eases fast updates of individual actors.** When implemented in the default five-VM configuration, Azure Service Fabric allows developers to apply changes to one VM at a time without interrupting the current service delivery. Actor instances on each VM use an eventual consistency model to coordinate changes, while completely abstracting developers from its complexity.

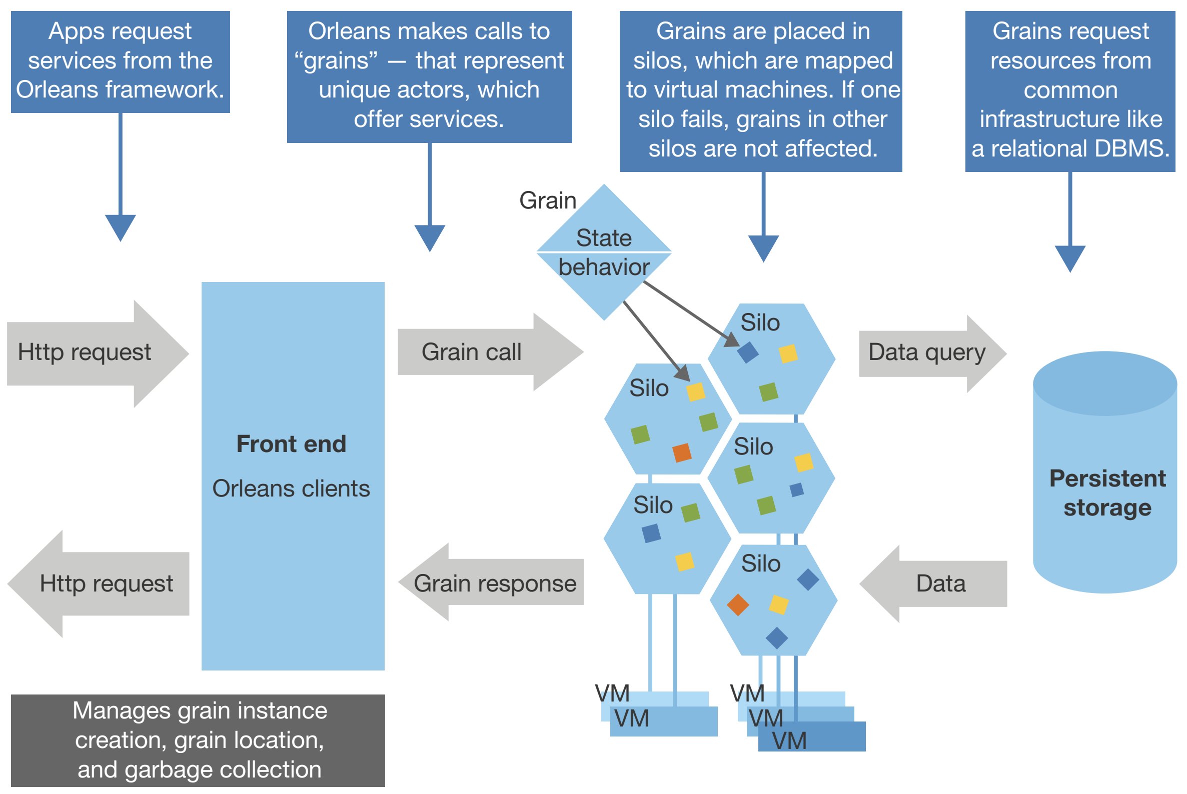
“We see several benefits from Service Fabric, including rolling updates and the ability to scale out individual services as needed.” (TJ Butler, chief software architect, Mesh Systems)

- **Presents an easier migration path for many current applications.** Actors differ from classical objects in that they don’t support inheritance or allow communication via functions; they use messages instead. That said, it’s possible to map an entity model to actors in a relatively straightforward way, then replace function calls with messages. As an example, Microsoft implemented its Reliable Collections in Service Fabric to map closely to the traditional Collections class that many.NET developers use today.

FORRESTER®

© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

FIGURE 4 The Orleans Project Maps Actors To Multiple VMs And Manages Concurrency For Developers


Apps request services from the Orleans framework.
Orleans makes calls to "grains" — that represent unique actors, which offer services.
Grains are placed in silos, which are mapped to virtual machines. If one silo fails, grains in other silos are not affected.
Grains request resources from common infrastructure like a relational DBMS.
Grain
State behavior
Http request
Front end
Orleans clients
Grain call
Silo
Data query
Persistent storage
Http request
Grain response
Data
VM
Manages grain instance creation, grain location, and garbage collection

Adopting New Programming On New Microservice Platforms

The future of application platforms will combine the new programming models with platforms built to support microservices. That future is unfolding now — and not just at unique digital enterprises like Amazon, Google, Microsoft, and Netflix. AD&D teams are already putting native microservice platforms to the test either by building their own platforms using technologies from Netflix (through its open source tools), Kubernetes and Mesos (the microservice orchestration tools that clients mention most), and other services or by adopting microservice platforms-as-a-service from CoreOS, Docker, Engine Yard, Google, Jelastic, Microsoft (Service Fabric), and Red Hat (OpenShift v3) (see Figure 5).[17]

In essence, the new platforms employ containers as their native service-deployment unit and remove control and management functions from application servers so they may apply to all microservices — whichever language or runtime they use. These platforms:

› **Support self-contained, independent, language-agnostic modules.** To a native platform, microservices are independent, self-contained units capable of using the platform's control and management services. Thus, the platforms are language-agnostic. The container images are not language-independent; each incorporates the runtime services it requires.

© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

- **Support internet-scale apps scaling with resilience and optimal costs.** Control services for microservices contribute to language independence. These services provide the dynamic deployment and provisioning services vital to internet scaling and pay-when-you-run economics, including clusters and multicopy deployments for resilience. The control services provide the communications protocols for microservice interactions within the same node or cluster as well as interactions with external services using APIs that enable composition, managing access and usage limits. Lastly, control services provide state management and other app services and also collect signals and data used by the platform's management plane.

- **Support resilient operations and trust boundaries.** In microservice platforms, monitoring, logging, and configurations are independent services applying to all services, helping ensure resiliency and good performance as well as assisting with problem resolution. Management services include identity resolution and establishment of trust boundaries through network configurations for microservices and tenants. Usage tracking and billing (as applicable) is part of the management plane as well.

FIGURE 5 The New Generation Of Microservice Platforms

Platform	Programming model	Release status	Programming language support	Billing model
Akka	Actor model	GA	Java, Scala	Open source: Commercial support is available from Lightbend on a tiered per-server model.
Amazon AWS Lambda	Functional pipeline	GA	Node.is (JavaScript), Java, and Python	Pay per execution (100 ms increments)
Google Cloud Functions	Functional pipeline	Alpha	Node.is (JavaScript)	To be announced
IBM Bluemix OpenWhisk	Functional pipeline	Experimental	Node.is (JavaScript) and Swift	Pay per execution
IronWorker Iron.io	Functional pipeline	GA	Ruby, Python, PHP, .NET, Node.is (JavaScript)	Customized pricing per customer, based on resources consumed
Microsoft Azure Service Fabric	Actor model	GA	C++, C#, F#	Per VM compute instance + storage + networking (1 min increments)
Microsoft Azure Functions	Functional pipeline	Preview	JavaScript, C#, Python, and PHP	Pay per execution (100 ms increments) after freemium tier
Pivotal Cloud Foundry Spring Cloud Services	Functional pipeline + annotations	GA	Java, Groovy, Kotlin	Open source: Commercial support is available from Pivotal on a per-container basis.

© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

Recognize The Destination Of Your Microservice Journey

Your developers have likely already begun their journey to microservice designs and, ultimately, to new application architectures. These beginnings may take the form of continuous integration and delivery practices and new tool chains. They may be found in new designs enabled by public cloud platforms and services or in increased software composition using APIs. All these roads lead to microservices.

AD&D leaders' most important move now is to recognize the destination and help channel developers' efforts toward a strategy that is safe and effective. The short-term to-dos include:

› **Downsize deployment units.** Any progress toward smaller deployment units is progress toward realizing the benefits of microservices.

› **Survey your developers to understand how current applications might adapt.** Expect the move toward microservices to initiate a new wave of application modernization decisions. Applications with well-structured entity models will suit modernization using Actor model frameworks, while tightly coupled, monolithic "balls of mud" will be candidates for replacement, retirement, or encapsulation behind new microservices.

› **Find a new architectural balance.** The choices developers make today will influence enterprise architectures over the long term, so it's important for AD&D leaders to understand how microservices are already creeping into their application portfolios. To progress toward a comprehensive architecture for microservices, harvest good design practices as guidelines and conventions and identify the most productive tools and methods to deal with the risks of microservice designs.

› **Fit new workloads to the best programming patterns.** Use functional pipelines for real-time stream processing; ETL; file processing; experimental apps with unpredictable, variable, or low infrastructure demands; and as an infrastructure services layer for mobile apps. Use Actor model frameworks for high-scaling transactional systems or as the server-side representation of connected IoT devices.

› **Decide explicitly on build versus buy for platform services.** AD&D teams that build their own platforms for microservices and/or container management bring middleware engineering skills to the task. Most enterprises can't sustain such platform engineering commitments and will do best adopting an integrated platform.

FORRESTER®

© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

What It Means

Prepare For The Serverless, Appless Future

Serverless computing is the aggregation of services requiring no knowledge of the infrastructure or platform services used to support them. Microservices lead to a serverless, appless future. Why?

› **Today's PaaS investments lead to serverless computing.** Programming models and platforms supporting microservices deliver this vision as a natural evolution of first-generation platforms-as-a-service. The irony is that native microservice platforms themselves provide the services required for microservices to interact and for microservice environments to operate, but these services are mostly hidden from developers behind programming models, manifests, and policy definitions.

› **Microservices lead to collaborative control over apps.** In a world of interacting services doing one thing (and only one thing) well, what is an application? In a microservice world, an app is defined by its a) business purpose and b) interactions with other microservices to accomplish that purpose at any given moment — not by a deployment archive containing related artifacts. Thus, microservice designs shift control over the application artifacts and services from monolithic application servers with high implementation consistency to networks of services with low implementation consistency. Even within a single enterprise, the owners of multiple services will share responsibility for application performance, scaling, reliability, security, and change management.

Engage With An Analyst

Gain greater confidence in your decisions by working with Forrester thought leaders to apply our research to your specific business and technology initiatives.

Analyst Inquiry

Ask a question related to our research; a Forrester analyst will help you put it into practice and take the next step. Schedule a 30-minute phone session with the analyst or opt for a response via email.

Learn more about inquiry, including tips for getting the most out of your discussion.

Analyst Advisory

Put research into practice with in-depth analysis of your specific business and technology challenges. Engagements include custom advisory calls, strategy days, workshops, speeches, and webinars.

Learn about interactive advisory sessions and how we can support your initiatives.

Supplemental Material

COMPANIES INTERVIEWED FOR THIS REPORT

Cloudsoft

Google

Mesosphere

Microsoft

Oracle

Pivotal Labs

Red Hat

Salesforce

Weaveworks

Endnotes

[1] For a discussion of microservices and application architectures, see the "Microservices Have An Important Role In The Future Of Solution Architecture" Forrester report.

[2] The "one thing" done well is from the perspective of the service consumer, and it could be a small service or large and powerful service. A large service's implementation would be further broken down into additional microservices. Source: Chris Richardson, "Introduction to Microservices," NGINX, May 19, 2015 (https://www.nginx.com/blog/introduction-to-microservices/).

[3] Driven by customer demands and competitive pressures, application development and delivery (AD&D) leaders have an imperative to deliver applications faster, yet many struggle with where to start and what comes next. They feel overwhelmed by the apparent immensity of the change. The good news? The path is now well understood and the journey is an incremental one, where progress is made step by step, practice by practice, and team by team. To learn more, see the "Boost Application Delivery Speed And Quality With Agile DevOps Practices" Forrester report.

[4] For more information, watch the following video from the AWS 2015 session where it discussed its release management practices. Source: "AWS re:Invent 2015 | (DVO202) DevOps at Amazon: A Look at Our Tools and Processes," YouTube, October 15, 2015 (https://www.youtube.com/watch?v=esEFaY0FDKc).

[5] For more information on the Netflix Simian Army and how it works, check out the following blog. Source: "The Netflix Simian Army," The Netflix Tech Blog, July 19, 2011 (http://techblog.netflix.com/2011/07/netflix-simian-army.html).

[6] For more information, read the following blog. Source: Allen Wang and Sudhir Tonse, "Announcing Ribbon: Tying the Netflix Mid-Tier Services Together," The Netflix Tech Blog, January 28, 2013 (http://techblog.netflix.com/2013/01/announcing-ribbon-tying-netflix-mid.html).

[7] Why containers rather than (or on top of) VMs? Forrester summarizes the reasons in the following brief. See the "Brief: Why Docker Is All the Rage" Forrester report.

[8] Microsoft's .NET runtime is used for C# and Visual Basic. Zend Technologies' runtime supports only PHP. Java runtimes support a number of languages in addition to Java.

[9] IBM's WebSphere Application Server and Oracle's WebLogic each allow developers to configure virtual app servers within individual installations. The result can be a much denser packing of services into a single physical application server instance. AD&D teams can use app-server virtualization to create "microservice environments" within either WebSphere or WebLogic.

[10] MVC: model-view-controller.

© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

[11] For more information on mobile development and MVCs, see the "The Future Of Mobile Application Development" Forrester report.

[12] Source: Goran (Kima) Kimovski, "Is It Time For Cloud 2.0?" TriNimbus blog, November 14, 2014 (http://www.trinimbus.com/blog/is-it-time-for-cloud-2-0/).

[13] Source: Carl Hewitt, Peter Bishop, and Richard Steiger, "IJCAI'73 Proceedings of the 3rd international joint conference on Artificial intelligence," ACM Digital Library, August 20, 1973 (http://dl.acm.org/citation.cfm?id=1624804).

[14] Source: William Douglas Clinger, "Foundations of Actor Semantics," DSpace@MIT, May 1, 1981 (http://hdl.handle.net/1721.1/6935).

[15] For more on the Halo 4 team's experience, check out the following video. Source: Sergey Bykov and Hoop Somuah, "Using Orleans to Build Halo 4's Distributed Cloud Services in Azure," Channel19, April 2, 2014 (https://channel9.msdn.com/Events/Build/2014/3-641).

[16] For more information on the benefits of the Actor model approach to concurrency, visit the following presentation. Source: Dror Bereznitsky, "The Actor Model - Towards Better Concurrency," SlideShare, January 17, 2010 (http://www.slideshare.net/drorbr/the-actor-model-towards-better-concurrency).

[17] To understand the key segments in the public cloud platforms market and how new requirements and technologies will drive the next phase of growth, see the "Vendor Landscape: Public Cloud Platforms Consolidate, But New Disruptions On The Way" Forrester report.

FORRESTER®

© 2016 Forrester Research, Inc. Unauthorized copying or distributing is a violation of copyright law.
Citations@forrester.com or +1 866-367-7378

FORRESTER®
CHALLENGE THINKING. LEAD CHANGE.

We work with business and technology leaders to develop customer-obsessed strategies that drive growth.

PRODUCTS AND SERVICES

› Core research and tools
› Data and analytics
› Peer collaboration
› Analyst engagement
› Consulting
› Events

Forrester's research and insights are tailored to your role and critical business initiatives.

ROLES WE SERVE

Marketing & Strategy Professionals	Technology Management Professionals	Technology Industry Professionals
CMO	CIO	Analyst Relations
B2B Marketing	› Application Development & Delivery	
B2C Marketing	Enterprise Architecture	
Customer Experience	Infrastructure & Operations	
Customer Insights	Security & Risk	
eBusiness & Channel Strategy	Sourcing & Vendor Management	

CLIENT SUPPORT

For information on hard-copy or electronic reprints, please contact Client Support at +1 866-367-7378, +1 617-613-5730, or clientsupport@forrester.com. We offer quantity discounts and special pricing for academic and nonprofit institutions.

Forrester Research (Nasdaq: FORR) is one of the most influential research and advisory firms in the world. We work with business and technology leaders to develop customer-obsessed strategies that drive growth. Through proprietary research, data, custom consulting, exclusive executive peer groups, and events, the Forrester experience is about a singular and powerful purpose: to challenge the thinking of our clients to help them lead change in their organizations. For more information, visit forrester.com.

Technology doesn’t create culture; it supports it. Red Hat is the leading provider of open source software, using the collaboration and creativity of its community and engineers to develop technology. Red Hat has been an innovator and contributor to technologies like Linux containers, and orchestration, which form the basis of its Red Hat OpenShift Container Platform. Red Hat also has a full middleware set of products, from Java EE, reactive programming, and Eclipse Microprofile to API management to business process engines. These middleware services offer tools to developers which are truly platform independent, working in on-premise physical hardware, public / private clouds, and containers. Red Hat products offer more than technology; there is an open culture and collaboration which can empower developers and technologists to create more. For more information about modern application development visit red.ht/AppMod and to learn how to use multi-runtimes to build new cloud-native apps visit red.ht/rhoar.