THE DZONE GUIDE TO ENTERPRISE INTEGRATION

2015 EDITION

BROUGHT TO YOU IN PARTNERSHIP WITH

3scale

ca technologies

Ready! API

Dear Reader,

Enterprise Integration is a toughie. And let's not even talk about "Enterprise Application Integration," because that *still* leaves a bad taste after 70% of EAI projects failed a decade ago.

Both conceptual components of EI are fuzzy, frustrating, and more or less intractable. The "enterprise" part is hard because "enterprise" often means "grew heterogeneous organically so nobody really knows the whole." The "integration" part is hard because insides are always different from outsides; because half the time nothing is documented (and the original coders left the company ten years ago); and, irreducibly, because no software black box really works as purely as a mathematical function. Integration is where engineering veers most sharply away from math, where you can't afford to look at either forests or trees alone. Multiple levels of architectural modeling are absolutely necessary but massively fail to capture the actual work being done by the code.

So what's an enterprising software engineer to do? These days the short answer is: microservices. A few years ago the spiritually similar answer was: service-oriented architecture. Often the broader answer to complexity is simply the timeless let-objects-act-like-grown-ups vision of object-orientation. Entities should mostly take care of themselves and otherwise send lightweight messages to other entities when they need something *only those other entities* can provide. Nobody can build an enterprise-scale pure mathematical function; and even if you could snapshot the world and functionalize it just this once, every component would still need to be able to grow on its own, presenting new powers to others and letting other systems—each with its own lifecycle—respond appropriately.

So we've addressed both fresh paradigms (like microservices and continuous delivery) and perennial problems (such as distributed computing) in our **2015 Guide to Enterprise Integration**. Try out our contributors' suggestions; see if our research results chime with your experience integrating software systems. And as always, let us know what you think.

JOHN ESPOSITO
EDITOR-IN-CHIEF
JESPOSITO@DZONE.COM

TABLE OF CONTENTS

EDITORIAL
John Esposito
research@dzone.com
EDITOR-IN-CHIEF

G. Ryan Spain
DIRECTOR OF PUBLICATIONS

Mitch Pronschinske
SR. RESEARCH ANALYST

Matt Werner
MARKET RESEARCHER

Moe Long
MARKET RESEARCHER

Allen Coin
EDITOR

Tom Smith
RESEARCH ANALYST

BUSINESS
Rick Ross
CEO

Matt Schmidt
PRESIDENT & CTO

Kellet Atkinson
VP & PUBLISHER

Matt O'Brian
DIRECTOR OF BUSINESS DEVELOPMENT

Jane Foreman
VP OF MARKETING

Alex Crafts
sales@dzone.com
DIRECTOR OF MAJOR ACCOUNTS

Chelsea Bosworth
PRODUCT MARKETING ASSOCIATE

Chris Smith
PRODUCTION ADVISOR

Jim Howard
SALES ASSOCIATE

Chris Brumfield
CUSTOMER SUCCESS ASSOCIATE

ART
Ashley Slate
DESIGN DIRECTOR

Yassee Mohebbi
GRAPHIC DESIGNER

Special thanks to our topic experts Markus Eisele, Daniel Bryant, John Davenport, Christian Posta, Tyler Treat, Chris Haddad, Arun Gupta, and our trusted DZone Most Valuable Bloggers for all their help and feedback in making this report a great success.

WANT YOUR SOLUTION TO BE FEATURED IN COMING GUIDES?
Please contact research@dzone.com for submission information.

LIKE TO CONTRIBUTE CONTENT TO COMING GUIDES?
Please contact research@dzone.com for consideration.

INTERESTED IN BECOMING A DZONE RESEARCH PARTNER?
Please contact sales@dzone.com for information.

Executive Summary

DZone's 2015 Guide to Enterprise Integration contains a wide range of information around the Integration space. This year's Guide includes articles on what's ahead for microservices, cloud integration, CD for service-oriented architecture, distributed-systems issues in integration, and SOA statelessness; it also contains executive-level insights on integration, a checklist for microservices best practices, an infographic examining the Richardson Maturity Model for REST, an extensive listing of integration solutions, results from our integration survey of almost 600 developers, and more. This executive summary will reveal some of the highlights of our research into the current state of integration.

RESEARCH TAKEAWAYS

01. MICROSERVICES STILL ON THE MIND

DATA: 27% of respondents from this year's Enterprise Integration survey use microservices, and 18% more have concrete plans to implement microservices models in the near future; even more respondents decompose services as needed. Still, the number of respondents who claim to use microservices currently is lower than in last year's survey, when 39% said they used microservices.

IMPLICATIONS: The decrease in respondents "using microservices" does not necessarily indicate a decrease in the popularity of microservices. Developers may be gaining a more realistic expectation of what microservices can do and are best for; it's also possible that there's a better understanding of what microservices architectures actually entail, and that the definition of microservices is becoming more concrete.

RECOMMENDATIONS: Don't jump on board with microservices because of stories you've heard about the architectural style's successes. Decompose your services as necessary, but really look into what works best for your app. Monoliths aren't necessarily anti-patterns, and sometimes may be right for the application you're trying to develop. Markus Eisele's article in this guide, "The Future of Microservices," can tell you more about where microservices are headed, and this guide's checklist, created with Arun Gupta, gives some microservices best practices.

02. REST IS (SOMETIMES) BEST

DATA: Over half of respondents (60%) use REST for APIs whenever they can, most commonly seen as REST with HTTP verbs and multiple resources (level 2 on the Richardson Maturity Model), with RESTful developers using this level of REST in 45% of their applications. Only 7% of RESTful users' apps make it to level 3 of the Richardson maturity model. The highest percentage of RMM usage occurs in the "We have no REST policy" camp, which uses level 3 REST in 9% of their apps.

IMPLICATIONS: REST provides countless advantages and makes the most of HTTP for data exchange. Still, RESTfulness doesn't fit in every situation (e.g. async messaging), but where applicable, highly-mature REST (HATEOAS) allows for cleaner integrations. Those without REST policies may have higher HATEOAS usage because of a mentality of "when we need to use it, we'll use it right."

RECOMMENDATIONS: Start leveling up when RESTing. Create more mature REST APIs to ease integration with other systems. Aim for REST at the HATEOAS level as much as possible, and get the most out of your transfers. Finally, don't try to jam REST into your API just to say it's there. Plain RPC isn't doing much to help your system integrate with others. This guide has an infographic that further dives into the different levels of the Richardson Maturity Model.

03. CLOUD INTEGRATION PLATFORMS

DATA: iPaaS usage is incredibly low (77% of respondents do not use any iPaaS), but usage has still increased from last year (when 80% did not use). IBM WebSphere and JBoss xPaaS win the cloud integration platform space with 12% and 5% usage, respectively. Other Cloud Integration Platforms, like CloudHub and SnapLogic, had around 1 - 3% usages.

IMPLICATIONS: iPaaS has plenty of room for growth. Slow adoption may result from hesitancies based on iPaaS's relative newness, or from "good enough" mindsets regarding on-premise integration platforms. But with growing needs for integration involving mobile and IoT applications, iPaaSes are quite likely to continue increasing in popularity.

RECOMMENDATIONS: Cloud Integration Platforms aren't an all-or-nothing game, and adoption of hybrid solutions for integration will likely benefit a majority of integrated applications, especially as CD and SOA/microservices continue to become organizational focuses, as Cloud Integration Platforms can help organizations handle multiple integration tasks more easily. Read Chris Haddad's article "Bypass Roadblocks With Cloud Integration" in this guide to find out more about how Cloud Integration Platforms can help you.

Key Research Findings

Close to 600 developers responded to DZone's 2015 Enterprise Integration survey. The respondents' demographics are as follows:

- The most common roles were developers/engineers (38%) and developer team leads (36%)
- 47% of respondents work at organizations with 500 or more employees; 37% work at organizations employing between 20 and 499; and 10% work at organizations with fewer than 20 employees or are self-employed.
- Most respondents work within organizations that are headquartered in Europe (40%) or the U.S. (35%).
- 45% of respondents have over 15 years experience as an IT professional; 26% have 10 – 14 years; 17% have 6 – 9 years; and 12% have 5 years experience or less.
- 89% of respondents work in organizations that use Java. Other popular languages/language ecosystems used within respondents' organizations are JavaScript (58%), .NET (40%), and Python (24%).

01. MORE DECOMPOSITION; FEWER "MICROSERVICES"

Last year's DZone Enterprise Integration survey showed almost 40% of respondents using microservices—unsurprising due to their success in web trendsetters like Netflix, not to mention the hype surrounding the concept. This year, however, we found a decrease in respondents who say they use microservices architectures. Only 27% claim to use microservices, while another 18% say they plan on using microservices in the future (time until implementation has a mean of 8 months, with a median and mode of 6 months). Still, the majority of respondents—whether they use microservices or not—decompose at least one of their application services, which may indicate a trend of decomposing services as needed, rather than using microservices for microservices' sake.

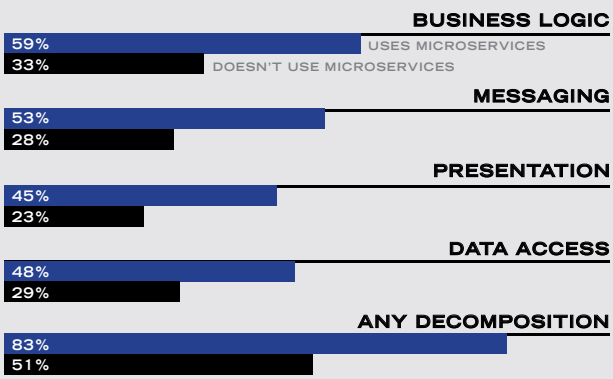
02. REST MAY NOT BE SO RESTFUL

Another shift from our survey results from last year involved REST usage. In the 2014 Enterprise Integration survey, 74% of respondents claimed to use REST APIs wherever possible. This year, we decided to dig a little deeper into how developers used REST. 60% of respondents say they use REST whenever they can, with another 7% saying they use REST consistently excepting specific situations (e.g. legacy code or SOAP requirements); 6% have plans to move towards full REST in the future (estimating about a year, on average), and 25% have no REST policy in place at all. Still, even those who claim to use REST don't necessarily utilize it's full potential. When asking respondents how they used REST in their web apps based on the Richardson Maturity Model (RMM), respondents overall estimated using REST at its highest level (HATEOAS) only 6% of the time. Even respondents who say they use REST whenever they can only use HATEOAS 7% of the time, with most of their RESTfulness falling into RMM2 (45% "REST whenever we can" respondents, 40% overall).

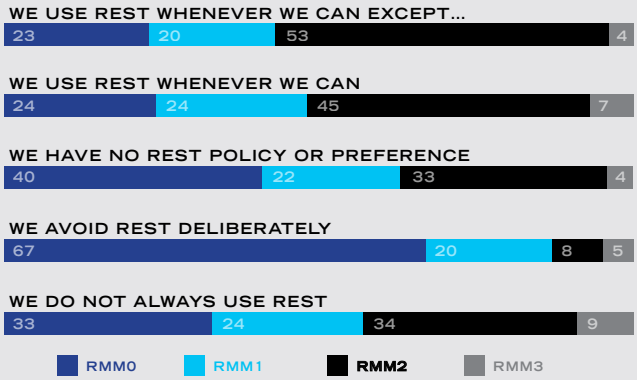
03. XML WIDELY USED; JSON WIDELY LOVED

When we asked about data serialization and interchange formats, we weren't surprised to find that the two most popular were XML and JSON—most respondents had never even used other formats we asked about, such as YAML and BSON. 97% of respondents use XML in some way, but only

01. **SERVICE DECOMPOSITION VS. MICROSERVICES USAGE**



02. **REST USAGE AT EACH MATURITY MODEL LEVEL**



24% say they enjoy using it. JSON, on the other hand, is used by 96% of respondents, and over half (52%) actually enjoy using it. Interestingly, the formats that respondents haven't used have a large effect on how RESTful their web services are. Only about 1% of respondents' web services are considered HATEOAS of respondents who have never used JSON. For respondents who have never used XML, an average of 13% of their web services make it to level 3 of the Richardson Maturity Model. Respondents who enjoy JSON are the least likely to use plain RPC.

04. LARGER ORGANIZATIONS UTILIZE MORE INTEGRATION ARCHITECTURE PATTERNS

We asked our respondents this year to identify the types of integration architecture patterns they utilize. The point-to-point style was the most popular, with 69% using that pattern in some way. 58% use message buses, 52% service buses, and only 24% use the hub-and-spoke model. We found that the number of different integration architecture patterns used depended on the size of the respondent's organizations. On average, respondents use about two different integration patterns for different needs. Respondents working in organizations with less than 100 employees all fell below this average, while respondents at larger organizations meet (or far exceed) this average, with respondents at the largest organizations using an average of 2.31 integration architecture patterns.

05. USAGE OF INTEGRATION TOOLS SCATTERED

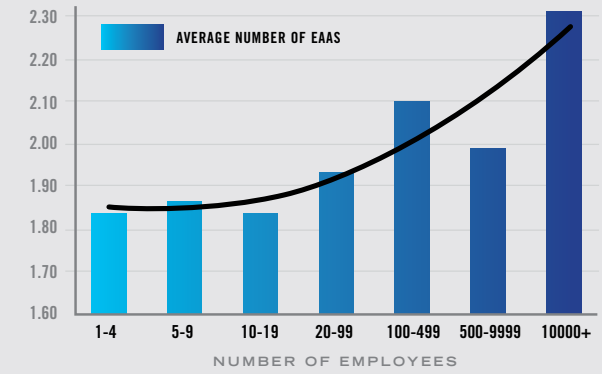
As far as integration tools go, there are many to choose from based on integration needs, and the usage of these tools is spread widely. In the realm of frameworks, suites, and ESBs, Spring Integration is the most popular, with 35% usage among respondents. Apache Camel is ahead of the curve, too, with 28% usage. A variety of other tools here have some popularity, such as Mule ESB (13%) and JBoss Fuse (12%), but among 8 other tools, usage ranged from just 5 – 10%. 32% of respondents are not currently using any integration frameworks, suites, or ESBs.

06. DIFFICULTIES ARE BECOMING LESS DIFFICULT

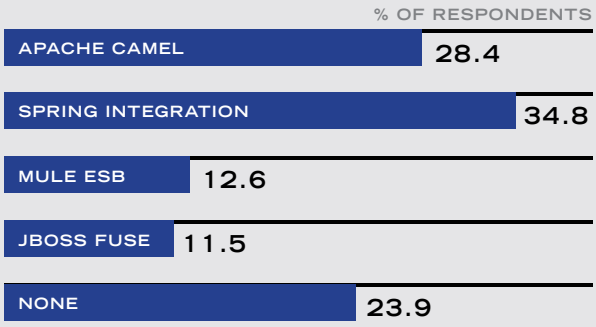
There are plenty of difficulties that can emerge when integrating. The most common difficulty respondents have is different standards interpretations between integration systems, with 36% of respondents having issues with that in their integrations. Other common issues are propagating changes to integrated systems (34%), managing asynchronous communications (24%), and needing to enrich data (13%). Still, of all difficulties we asked about, almost all have a significant drop from last year—difficulties propagating changes and different standards interpretations dropped over 20%. The only increase from last year to this year involves cloud to on-premise integrations, with 12% of respondents dealing with that difficulty vs. last year's 9%.

03. **JSON VS. XML: RESTFULNESS**

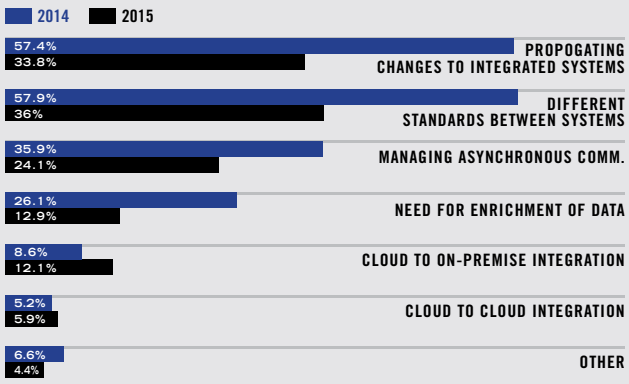
04. **AVG. NUMBER OF ENTERPRISE APPLICATION ARCHITECTURES VS. COMPANY SIZE**



05. **DOES YOUR ORGANIZATION USE ANY OF THE FOLLOWING INTEGRATION FRAMEWORKS, SUITES, OR ESBs?**



06. **ORGANIZATIONAL INTEGRATIONS DIFFICULTIES: 2014-2015**



4 Things Every API Deserves

SCALABILITY AND UPTIME

Whatever traffic control architecture you choose, it shouldn't be a bottleneck or a single point of failure. It should be flexible enough to work with your preferred infrastructure, with a deployment method (such as a proxy or CDN) that fits your specific use case and needs. Know the details on scalability and reliability—how quickly can you increase capacity during a traffic spike? Caching, fault tolerance, and load balancing will help ensure minimal downtime for API users.

ACCESS CONTROL AND SECURITY

Authentication is essential, but by itself provides insufficient protection for your API. You should be able to manage credentials, establish different levels of access for different types of users, and control how client applications are permitted to interact with your API.

DEVELOPER EXPERIENCE TOOLS

To improve adoption and keep developers engaged, you'll need to provide tools that simplify the onboarding process. Example code, documentation, quick-start guides, and other reference materials all make it easier for developers to get started and help to enable success. Make it easy for new users to get a foot in the door by providing a centralized developer portal and interactive documentation.

YOUR API CAN BE AN INCREDIBLY VALUABLE ASSET—TREAT IT LIKE ONE.

INSIGHTFUL ANALYTICS

You need insight into API performance. Which applications generate the most traffic? Which APIs are most popular, and which APIs or endpoints are used the least? You should have visibility into usage trends, and ongoing monitoring for key metrics. Automated alerts should be configured to flag any unusual behavior or unexpected changes, which could indicate a major issue.

WRITTEN BY STEVEN WILLMOTT
CEO AND CO-FOUNDER AT 3SCALE

3scale API Management Platform by 3scale

3scale

3scale's unique hybrid architecture creates flexibility, performance, and scale not achievable or cost-effective with other solutions.

CATEGORY
API Management and Integration

NEW RELEASES
Quarterly

OPEN SOURCE?
No

CASE STUDY

In order to transition from a crowd-sourced idea to the most-used dataset of startup information, the Crunchbase team needed to build an API that functioned as a strategic business asset, in line with overall growth strategy. The team sought an API management solution that would both reduce operational costs and provide a flexible foundation for growth; ease of implementation and management were also key considerations. After a straightforward implementation of 3scale, Crunchbase saw a dramatic improvement in performance. Developer signups and API usage skyrocketed, and implementing 3scale led to decreased maintenance time, allowing Crunchbase's engineering team to spend more time working on improvements to the API itself.

STRENGTHS

- Flexible deployment options including an API gateway, CDN, or on-premise.
- Single interface for control and visibility.
- Scalable, high-performance architecture.
- Built-in developer portal CMS and interactive documentation.
- Highly customizable security and access control features.

NOTABLE CUSTOMERS

- UC Berkeley
- Johnson Controls
- Typeform
- Coldwell Banker
- Adidas
- CareerBuilder
- Campbell's Soup
- Crunchbase

BLOG 3scale.net/blog | TWITTER @3scale | WEBSITE 3scale.net

The Future of Integration With Microservices

BY MARKUS EISELE

QUICK VIEW

01 Instead of selecting a single ESB or integration product you now need to find a solid combination of necessary components.

02 Through a modern cloud platform, individual services can easily be deployed, scaled, and exposed as needed.

03 Microservices will lead to distributed and segregated data volumes that require new approaches like streaming data solutions to manage.

04 The segment architecture approach is still too new to give recommendations. For a while, it will still be your responsibility to understand the capabilities you need by doing your own research.

THE WORLD OF IT AS WE KNOW I.T. TODAY IS CHANGING DRASTICALLY. JUST A COUPLE OF YEARS AGO, DEVELOPERS spent months or even years developing infrastructures and working on the integration of various applications. Huge projects with multiple participants were required to implement desired features. With the advent of DevOps, various Platform-as-a-Service (PaaS) environments, containers, and microservices, many complex requirements can be met within a much shorter time. The Internet of Things (IoT) is also expected to change established applications and infrastructures. As a result of these converging trends, the way in which system integration will work is set to undergo a fundamental change in the coming years.

System integration has come a long way, from point-to-point connections between individual systems towards the first integration solutions that helped in standardizing those connections. And in the advent of a much more business-centered design—and the broader shift into more service-oriented organizations—the Enterprise Service Bus (ESB) evolved from a pattern to a variety of products. They all promised to deliver reusability and exchangeability by standing up as a centralized and managed infrastructure component.

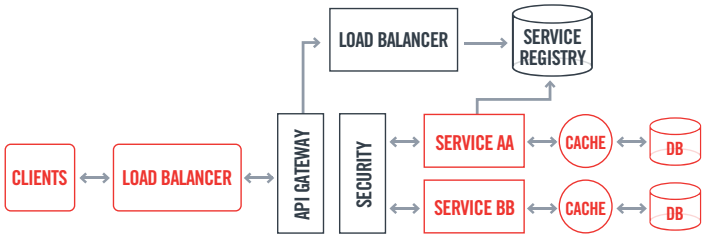
As a direct result, applications had to be sliced—and even partly rebuilt—to support exchangeable services. Service-oriented architectures (SOAs) were the new paradigm behind this. Unfortunately, the interface technology of choice tended to be Web services (WS). Web services transport data between systems by encoding the data into XML and transmitting it via the Simple Object Access Protocol (SOAP), and they introduced a significant amount of additional code and descriptors into most projects. With the extra code and configuration came extra complexity on various levels. Government of service versions and cross-service security, as well as documentation and requirement engineering, had to be tweaked to focus on services instead of features. All of this had to be managed.

OUTER ARCHITECTURE GAINS MORE IMPORTANCE

With the next technology evolution—Microservices—the need to manage even more potentially-polyglot and distributed services became overwhelming.

Looking back, we can see that integration complexity moved from the inner architecture of applications towards the outer architecture of the complete system. The outer architecture refers to the platform capabilities you need to help all those simple little microservices work together.

And it isn't enough just bringing the technology aspects together. The outer architecture also has to support development teams and the Software Development Lifecycle (SDLC) to deliver on the promises of flexible and scalable development and deployment.



Looking at the architecture diagram above, it becomes very obvious that the most important parts of your application are now outside of your services. Instead of solely focusing on

integration flow and logic, the correct approach to run your services becomes even more important than it ever was.

The most important parts of your application are now outside of your services.

PICKING A BASELINE - XPAAS

Instead of selecting a single ESB or integration product you now need to find a solid combination of all of the needed parts. While the acronym iPaaS (integration platform as a service) started to emerge a couple of years ago, you will end up with even more than one xPaaS (everything as a service) service. The best starting point for the outer architecture is an aPaaS (application platform as a service) platform or framework, which has the potential to integrate seamlessly with the relevant parts of the underlying PaaS. There are many options available. Classical enterprises might still stick with a Java EE based aPaaS, while others will quickly move on to something more lightweight. But the platform language is no longer the critical aspect here: more important is how the aPaaS hooks into the centralized cross-cutting and operational capabilities.

OPERATIONAL CAPABILITIES

While aPaaS and iPaaS both refer to complete product stacks mostly, they have one thing in common: the base PaaS offering on which they rely on. A modern cloud platform knows a lot more about the applications it runs than the ones years ago. With the help of deployment templates and descriptors, the individual services can easily be deployed, scaled, and exposed as needed. And even the service orchestration is encapsulated. All of this works because other emerging technologies like containers and container orchestration (e.g. Kubernetes) is built into the core of today's PaaS. What very quickly started to be a commodity under the hood is enhanced by some vendors with operational consoles—and enhanced by a very few vendors with additional developer tooling. Developer support, especially, mostly ends with what DevOps teams and continuous delivery best practices demand.

DEVELOPER SUPPORT

But there is a lot more effort needed to develop highly-distributed systems. While complex IDE plug-ins from ESB times were able to tweak the centralized service repository, loosely-coupled microservices do need more runtime information to build an application. Instead of centrally wiring services together, we now want to look up service endpoints at runtime. Registration, versioning, and additional meta-information like SLAs also need to be resolvable by every service from the centralized registry. Dispatching requests to one of the available instances will be done by the underlying PaaS. A developer will have to provide most of the relevant information at time of development, and the services will auto-register themselves while the platform uses their individual information to distribute the workload most efficiently. Tooling to browse service documentation or look up existing services will also be a very critical feature. When a microservice-based application is finally in production, it is time to have the ability to follow the distributed request-flow throughout the system. Being able to track down errors and assist with debugging is a complex task, which the outer architecture of our integration solutions also needs to support.

THE CHANGING FACE OF DATA INTEGRATION

The last—but no less important—part of new integration architectures will be the next level of data-integration approaches. With every microservice being responsible for its own database, the classical ETL (Extract, Transform, Load), data-warehousing, and data federation systems will quickly reach an unmaintainable state. New approaches to master data management will be required to handle these kind of distributed and segregated data volumes. Among the possible solutions on the horizon are Big Data or stream data solutions, which take data-changing events and allow operations on a copy of the data. But virtualized data-models will also help for reporting or warehousing requirements. More complex solutions might also allow the exposure of data services, which themselves can be consumed by business microservices.

READY FOR PRIME TIME?

Vendors have already started to "microservice-wash" their tools and platforms to get your attention. And the segment architecture approach is still too new to give recommendations. For a while, it will still be your responsibility to understand the capabilities you need by doing your own research. Some promising candidates are evolving out of the open-source think tank at this very moment. First and foremost, projects like OpenShift Origin, WildFly Swarm, Fabric8, and APIMan will help you put together most of the puzzle pieces in your microservices-based architecture.

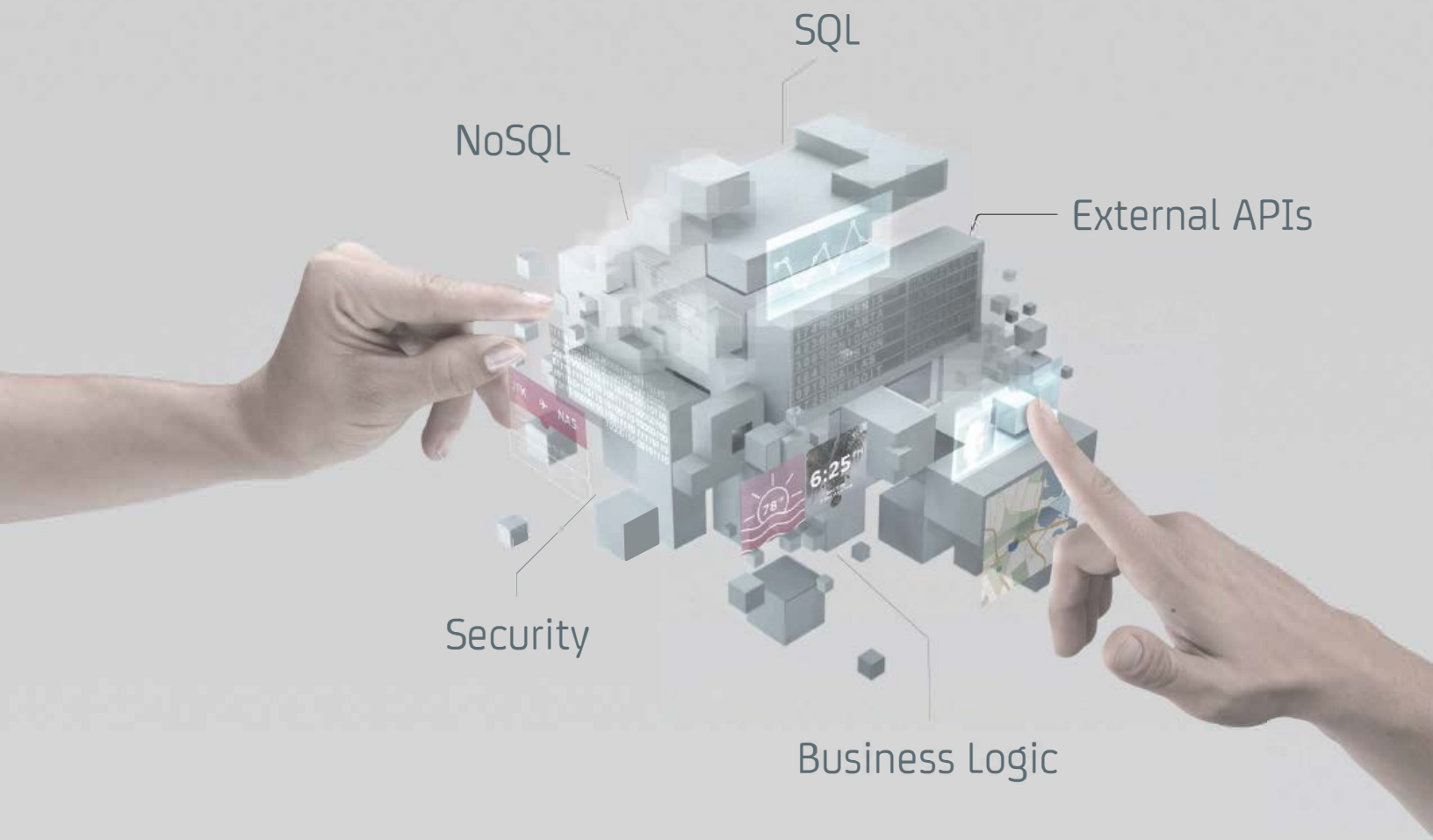
Vendors have already started to "microservice-wash" their tools and platforms to get your attention.

Microservices and containers will change the way we build, maintain, operate, and integrate applications. When architectured with discipline and a careful selection of the outer architecture, they will help applications become more portable and more adaptive. In the end, better application or service integration will be very different to today's approaches: it will become a number-one key requirement for distributed microservices applications.

MARKUS EISELE is a Developer Advocate at Red Hat and focuses on JBoss Middleware. He is a Java Champion and former ACE Director as well as a prolific blogger, community leader, and book author. He has worked with Java EE servers for 14 years.

Launch new apps and integrations faster than ever with CA Live API Creator.

Quickly build APIs from diverse data sources, applications and business logic using a point-and-click approach.



Discover how >

ca.com/CreateAPIs

ca technologies

How API Management Can Solve the Enterprise Architect's Dilemma

The application economy is driven by an always-connected, mobile, application-based world. To meet the demand of consumers today, an enterprise architecture needs to be capable of supporting a diverse set of endpoints—internal applications, legacy systems, external partners, customers, mobile devices, IoT devices, etc. The architecture also should be able to scale on an on-demand basis to support inbound and outbound traffic with volumes exceeding possibly millions of transactions.

So, when an enterprise architect (EA) modernizes their architecture, their dilemma is whether to rip out and replace all that has been built or to extend the existing architecture to seamlessly move into a digital enterprise. The answer lies with new design architectures such as API management.

A NOESB ARCHITECTURE?

Enterprise Service Bus (ESB) or Service-Oriented Architecture (SOA) models were designed a couple of decades ago for internal integration needs. For new digital initiatives around mobile, IoT, and the cloud, ESBs fall short, and so the existing, legacy integration models need to be upgraded using API management as a solution.

APIS POWERING NEW ARCHITECTURES

With an API-enabled solution, you can:

- Expose and manage select APIs externally to customers and partners
- Adopt the right security models to secure your APIs
- Govern APIs and manage change control for minimal impact on consumers
- Bring the needed scalability to match the speed of the Internet and high growth of mobile applications
- Improve IT agility in order to rapidly respond to changes requested by business

Read *An Enterprise Architect's Guide to API Integration for ESB and SOA* to know how API management can enable modern connectivity, offer sophisticated security control, boost developer productivity, and prepare the EA to take on new business models with ease.

WRITTEN BY DINESH CHANDRASEKHAR

DIRECTOR, API MANAGEMENT PRODUCT MARKETING, CA TECHNOLOGIES

CA API Management by CA Technologies

Bring your apps to market faster with CA API Management at the center of your digital strategy.

CATEGORY

API Management and Integration

NEW RELEASES

Quarterly

OPEN SOURCE?

No

CASE STUDY

IceMobile's mission is to add emotion to transactional loyalty in the food retail market. By combining data and technology to deliver personalized mobile experiences, it boosts revenue and increases customer loyalty for retailers worldwide.

With retailers' legacy IT systems putting sales at risk, IceMobile needed to accelerate and simplify integration between IceMobile's Bright Loyalty Platform and retailers' back office systems.

APIs require only limited changes to the food retailers' back office systems. CA API Management simplifies connections with multiple international food retailers while ensuring IceMobile meets retail security standards.

By simplifying integration, IceMobile has cut implementation times from 14 to eight weeks while minimizing impact on retailers' busy IT departments. The ability to integrate with any technology safeguards growth.

STRENGTHS

- **Create APIs and Integrate Everything** - Bring together the data you need to power next-generation cloud, mobile, and IoT applications with broad integration capabilities.
- **Secure the Open Enterprise** - Use a secure, analyst-acclaimed platform for integrating across apps, devices, and businesses.
- **Accelerate Mobile and IoT Development** - Empower developers with tools to streamline the development process, improve productivity and reduce time-to-market.
- **Unlock the Value of Data** - Build API-based ecosystems to extend your brand, develop new digital products and services, or create new revenue channels by monetizing your data.

NOTABLE CUSTOMERS

These CA API Management customers are speaking about their successes at CA World 2015, Nov 16-20 in Las Vegas: Nike, Visa, PepsiCo, Samsung, General Motors, FedEx, The Walt Disney Company, US Bank.

BLOG blogs.ca.com/category/api-management

TWITTER @CaAPI

WEBSITE ca.com/api

Continuously Delivering SOA

BY DANIEL BRYANT & JOHN DAVENPORT

QUICK VIEW

01 Two key trends in the successful development of enterprise software applications are the use of service-oriented architecture (including microservices) and the implementation of continuous delivery

02 Service virtualization is one form of application 'lifecycle virtualization,' and enables a service interface or API to be virtualized or 'faked'

03 Service virtualization can add value throughout the entire development lifecycle, including automated acceptance, functional and performance testing, and ultimately facilitate the creation of an effective continuous delivery quality assurance pipeline

OUTER ARCHITECTURE GAINS MORE IMPORTANCE

Two undeniable trends that have emerged within enterprise IT over the past several years are: 1) the componentization of business functionality into self-contained services (first with service-oriented architectures—"classical SOA"—and more recently via "microservices") and 2) the drive to "continuously deliver" these services, using techniques such as continuous integration, build pipelines, and continuous deployment. Although these trends have provided undeniable benefits—such as allowing the realization of team-owned services (a la Conway's Law), encapsulation of state and increased cohesion of functionality, and the enablement of independent application release lifecycles and scalability—we have also seen increased challenges with the collaboration and orchestration of dependent services.

Testing a system composed of independent services is typically more complex, particularly if several of the services are outsourced or managed by external third-parties, and hence have limited or restricted availability; and developing against external APIs that may not be available—or do not allow configuration of state, lifecycle-managed, or failure scenarios—often leads to delays in delivery (as stated by a recent Forrester report).

The issues mentioned above have lead to the creation of a domain of quality assurance (QA) technology classified as "lifecycle virtualization," which enables parts of a system under development or testing to be "virtualized" or "faked" to varying degrees of complexity and configurability. This allows one or more services or application components to be virtualized, and can remove reliance on the need to access the corresponding "real" services when developing against an application programming interface (API), or running end-to-end functional or performance testing.

A report by Voke Inc., an application lifecycle analyst firm, states that three specific levels of lifecycle

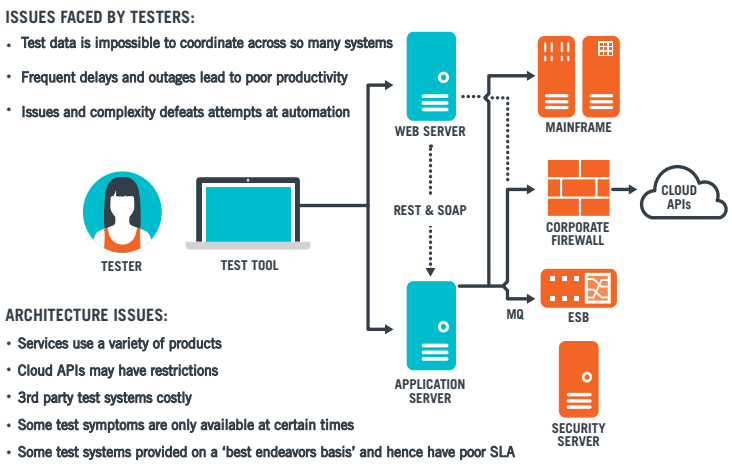


FIGURE 1. TYPICAL SERVICE INTERACTIONS THROUGHOUT DEVELOPMENT LIFECYCLE

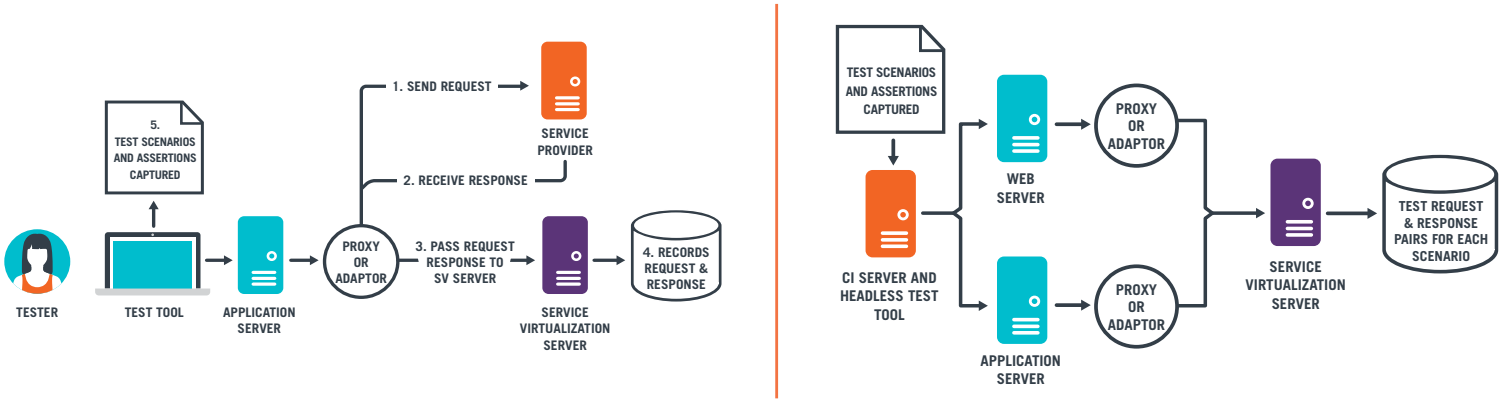


FIGURE 2. **TYPICAL IMPLEMENTATION OF SERVICE VIRTUALIZATION**

virtualization have emerged as commercial and open-source products: service virtualization, where a service interface or API can be virtualized; virtual labs or sandboxes, which offer either a full or partial virtualized test environment that emulates a system or application; and network virtualization, where developers can test an application's operation within a simulated virtualized networking environment that mimics expected deployment conditions.

This article focuses on the use of service virtualization, as—in our experience—this is more generally applicable across organizations, can provide benefit across an entire IT team, and can add value throughout the complete application development lifecycle. With the recent rise in popularity of small, self-contained microservices, it is not uncommon to find a service that interacts with a variety of external components or services as shown in Figure 1.

THE MECHANICS OF SERVICE VIRTUALIZATION

Service virtualization is typically implemented by the use of some form of proxy that either 1) redirects traffic to/from a service being virtualized, or 2) actually performs the virtualization itself. Requests and responses are typically defined using one of three methodologies:

- Manual creation of service endpoint and associated responses (e.g. WireMock service stubbing)
- Creating a series of customizable request/response stubs from a service's descriptor (e.g. WSDL, Thrift IDL descriptor, Swagger resource declaration)
- Recording of request/response, including the ability to identify non deterministic data, such as timestamps or UUIDs, and customizable responses (e.g. VCR service virtualization cassettes)

Choosing to invest in the use of a service virtualization tool can bring many benefits, but it is not a decision to be taken lightly, and may not fit every development requirement. Accordingly, Table 1 contains information that highlights the differences in approaches to "virtualizing" services, and aims to allow comparison of the various methodologies and tooling.

SERVICE VIRTUALIZATION TOOLING

There are a variety of open-source and commercial products that seek to offer service virtualization solutions.

Open-source service virtualization applications:

- Wiremock - a Java-based flexible library with a JSON API that can be used to virtualize, mock, and stub web services
- Mountebank - JavaScript (Node.js) based tool that provides cross-platform, multi-protocol test doubles "over the wire"
- VCR/Betamax - A Ruby/Java-based application that allows the recording and replaying of HTTP interactions
- Mirage - A Python-based service virtualization tool that provides capabilities to record and replay TCP/HTTP interactions. (*Disclaimer: The authors of this article are both employed by companies that have contributed to the creation of this tool.*)

Commercial service virtualization products include:

- CA Service Virtualization
- HP Service Virtualization
- IBM Rational Test Server
- Parasoft Virtualize
- Smartbear ServiceV Pro

	VIRTUAL ENVIRONMENT	VIRTUAL SANDBOX	SERVICE VIRTUALIZATION	MOCKS AND STUBS
Level of Virtualization	Entire system environment	Specific applications within environment	Service interface (e.g. remote API), typically coarse-grained	Service or component API, typically fine-grained
Typical Implementation	Virtualized appliance, or SaaS offering	Virtualized appliance, or SaaS offering	Standalone application	Platform-specific code and libraries
Typical 'Cost' of Acquisition, Configuration, and Running	Prohibitively high (in the majority of cases)	High	Medium	Low
Team Typically Responsible for Operation	External third-party or operations	External third-party or operations	Operations	Developers
Maintenance Issues	Prone to unavailability (typically no SLAs), restricted access, and long initialization times	Prone to unavailability (typically no SLAs), restricted access, and inconsistent versioning of service interfaces	Generally limited issues (if managed properly), with the primary challenge being the synchronization of functionality offered by 'real' and 'virtualized' service	Mocks and stubs can be brittle, as they typically operate at a level of fine granularity
Target Audience	Entire organization (often used in pre-sales activities, right through to development)	Business analysts (BA), QA, developers	QA, developers	Developers
Reusability Potential	The virtual environment is typically shareable, but access must be coordinated, and test scenarios and data may not be shareable	Sandbox is typically shareable, but test scenarios and data are often difficult to share without manual coordination	The SV application is typically shareable across QA and development teams, as are the virtualized interfaces, test scenarios and data.	Typically limited to the owning development teams
Test Automation Potential	Very low	Low	High	Implicit in operation
Scenario (and Associated Data) Creation	Typically not available	Typically manual creation (and curation) of sandboxed test data is the only option	Virtualized response code generation (from WSDL, Swagger, etc.), manual creation of interactions, automated recording of interactions	Typically manually created, but several tools (VCR, Betamax, SOAP UI) allow recording of interactions
Ease of Operation (for Creating and Running Test Scenarios)	Typically complex and managed by third-party	Often complex and customized to the application	Typically easy, as the target audience is non-technical QA	Difficult, as the target audience consists of technically advanced developers

TABLE 1. **COMPARISON OF 'VIRTUALIZED' SERVICE METHODOLOGIES AND APPROACHES**

CONCLUSION

The emergence of service-oriented architectures, and more recently microservices, promised to improve the ability to continuously deliver business functionality to end-users, but the increased complexity of collaboration and orchestration between these services has often slowed their development. Services are also frequently developed by external agents, or are consumed using the "software as a service" (SaaS) model, and so may not be reliably available for development or testing against. Service virtualization is one technique that can overcome these problems by allowing the configurable "virtualization" of a service interface or API (and associated scenarios and data), which can then be integrated within the development lifecycle, the automated acceptance, functional and performance testing process, and the continuous delivery quality assurance pipeline.

DANIEL BRYANT (@danielbryantuk) is the Chief Scientist at OpenCredo, a London-based IT consultancy that specializes in helping clients to deliver better software, faster and more cost-effectively. Daniel is currently focused on enabling continuous delivery, DevOps tooling, cloud/container platforms, and microservice implementations. He is also a leader within the London Java Community (LJC), and regularly presents at international conferences such as QCon, JavaOne, and Devoxx.

JOHN DAVENPORT (@spectolabs) is the CTO of SpectoLabs, a company specializing in the creation of open source service virtualization and testing products. John has over 30 years experience of IT development in a variety of sectors. Prior to SpectoLabs he worked on the development of several major systems at British Airways, including ba.com, and also had general responsibility for development tools and automation. Seeing the difficulties of enterprise development at first hand, he became convinced some years ago that Open Source frameworks, infrastructure & tooling would be delivering the most effective solutions needed by business.

diving deeper

INTO ENTERPRISE INTEGRATION

TOP 10 #EI TWITTER FEEDS

 @MARTINFOWLER

 @SIMONBROWN

 @GHOHPE

 @ROSSMASON

 @DANIELBRYANTUK

 @MULESOFT

 @3SCALE

 @WSO2

 @OPENCREDO

 @FLOWGEAR

DZONE ENTERPRISE INTEGRATION ZONES

Enterprise Integration

dzone.com/integration

Enterprise Integration is a huge problem space for developers, and with so many different technologies to choose from, finding the most elegant solution can be tricky. The EI Zone focuses on communication architectures, message brokers, enterprise applications, ESBs, integration protocols, web services, service-oriented architecture (SOA), message-oriented middleware (MOM), and API management.

DevOps

dzone.com/devops

DevOps is a cultural movement, supported by exciting new tools, that is aimed at encouraging close cooperation within cross-disciplinary teams of developers and IT operations/system admins. The DevOps Zone is your hot spot for news and resources about Continuous Delivery, Puppet, Chef, Jenkins, and much more.

Cloud

dzone.com/cloud

The Cloud Zone covers the host of providers and utilities that make cloud computing possible and push the limits (and savings) with which we can deploy, store, and host applications in a flexible, elastic manner. This Zone focuses on PaaS, infrastructures, security, scalability, and hosting servers.

TOP EI REFCARDZ

Getting Started With Microservices
Still re-deploying your entire application for one small update? Microservices deploy modular updates and increase the speed of application deployments.

Foundations of RESTful Architecture
Introduces the REST architectural style, a worldview that can elicit desirable properties from the systems we deploy.

SOA Patterns
Covers SOA Patterns and includes Pattern names, Icons, Summaries, Problems, Solutions, Applications, Diagrams, Results, and Examples.

TOP EI WEBSITES

Microservice Architecture
microservices.io

Enterprise Integration Patterns
enterpriseintegrationpatterns.com

Java Community Process
jcp.org/en/jsr/platform?listBy=3&listByType=platform

TOP EI TUTORIALS

Monolithic to Microservices Refactoring for Java EE Applications
bit.ly/micro-javaee

Understanding Trade-Offs in Distributed Messaging
bit.ly/distributedmessaging

Martin Fowler on the Microservice Premium
bit.ly/micros-prem

HOW RESTful ARE YOU?

RESTfulness is a concept that is often thrown around to refer to communications between integrated systems. But REST in itself is a nebulous idea, and what some may consider REST, others may think of as a mere transfer of data over (usually) HTTP. The Richardson Maturity Model, created by Leonard Richardson, tries to demystify the idea of RESTfulness by breaking down communications into stages of REST maturity. Level 0 attempts to incorporate REST into communications by simply using HTTP as a system of data transportation, without taking advantage of its benefits; level 3 describes the ideal REST style. This infographic shows real-world examples of "RESTful" communications. The statistics included refer only to the 60% of our survey respondents that claimed "We use REST whenever we can," in an attempt to examine how RESTful REST practitioners really are.

0: PLAIN RPC

Alice is driving through. Bob is at the Food window

SCENARIO

This level of the RMM represents the transmission of data through remote procedure calls, without utilizing other benefits of web transmission—essentially using HTTP as nothing more than a way to send messages back and forth. Below, Alice POSTs her intention to spend money, and Bob POSTs what she can spend money on; once the money has been POSTed, there is no way to distinguish Alice's specific order. RESTful respondents, on average, estimate 24% of their web services are conducted over plain RPC.

Alice: "I would like to POST money here."

BOB: "You can POST $2 For a soda. You can POST $3 For Fries."

ALICE: "POST $2 For a soda."

BOB: "Your order has been received (200 OK)." "In case of an error" "There was an error processing your order."

1: INTRODUCING RESOURCES

Alice is buying food. Bob is behind the counter

SCENARIO

Level 1 of the RMM routes requests to specific resources for specific functions, separating actions and objects. Here, changes can be made on an object-by-object basis. As opposed to the last scenario, in level 1, Alice receives an order number to the resource she requested. 24% of RESTful respondents' web services are estimated to use level 1 interactions.

Alice: "I would like to POST money here."

BOB: "You can POST $2 For a soda. You can POST $3 For Fries."

ALICE: "POST $3 For Fries."

Bob: "Order #555 has been received (200 OK). Order coming up."

2: MULTIPLE HTTP VERBS

Alice is entering and talking to Bob the bartender

SCENARIO

This level of REST uses HTTP to relay not only what should be communicated, but also how messages should be communicated. Appropriate verbs are used, often based on whether data needs to be transformed or merely gathered. At the level 2 bar, Alice harmlessly asks what she can GET; and now, once the order is made, an actual created confirmation reassures her that her order has indeed been placed—and for the right item. An average of 45% of web services by RESTful respondents are estimated to be level 2—by far the most popular level of RESTfulness among our survey respondents.

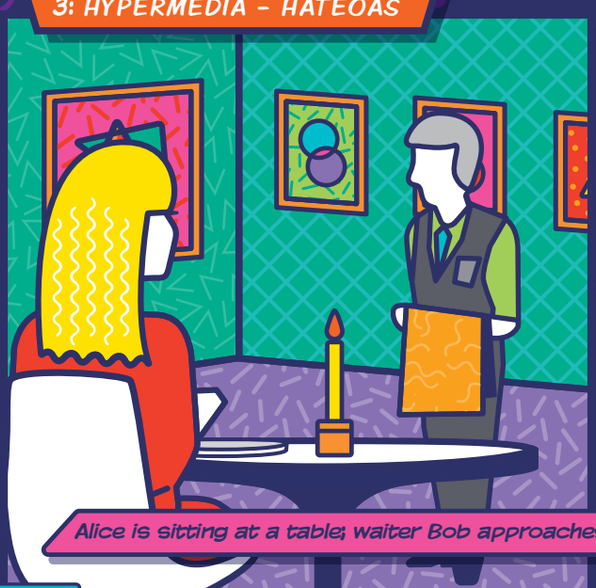
Alice: "What can I GET here?"

Bob: "We have soda For $2. We have Fries For $3. We have beer For $4."

ALICE: "POST $4 For a beer."

BOB: Order #555 has been received (201 Created). Beer coming up."

3: HYPERMEDIA - HATEOAS



Alice is sitting at a table; waiter Bob approaches

SCENARIO

The highest level of REST maturity fully utilizes the 'H' in HTTP—using hypertext to respond to actions in order to provide next steps with specific, accessible URIs. In this restaurant, when Alice asks what she can GET, Bob tells her exactly how she can order each item. Furthermore, when she does POST an order, Bob tells her what additional options she now has for ordering. RESTful respondents, on average, estimate they only use HATEOAS practices for 7% of their web application communications.

Alice: "What can I GET here?"

BOB: You can say "soda" to order a soda For $2. You can say "French Fries" to order Fries For $3. You can say "lager" to order a beer For $4. You can say "hamburger" to order a burger For $5."

ALICE: "POST $5 For a 'hamburger!"

BOB: Order #555 has been received (201 Created). Burger coming up. You can say "ketchup" to add ketchup to that burger; you can say "pickles" to add pickles to that burger; you can say "lettuce" to add lettuce to that burger...

CREATED BY DZONE.COM

Integration Is Still A Distributed-Systems Problem

QUICK VIEW

01 Integrating systems is hard, and now we have to deal with SaaS, mobile, Big Data, and other integration obstacles.

02 Copying what others are doing because they appear successful is not going to lead to a successful result.

03 Focus on core distributed-systems principles; integration is still a distributed-systems problem.

04 Tackling domain complexity, API evolution, unreliable networks, and organizational structures are key.

BY CHRISTIAN POSTA

INTEGRATION IS NOT A SEXY JOB, BUT SOMEONE HAS TO DO IT. WHETHER YOU'RE A SMALL STARTUP OR AN AGELESS, large corporation, chances are high that you'll need to integrate systems as part of an overall solution. On top of that, integration solutions that we have to provide these days are becoming more and more complex. We have to think about Big Data, IoT, Mobile, SaaS, cloud, containers—the list goes on endlessly. IT is being asked to deliver solutions smarter, faster, and more efficiently. All the while the business is in a competitive environment that literally changes every day as technology changes, and we're being strangled by domain complexity. Rest assured, however; we've been promised solutions.

Today, it's microservices. Yesterday it was SOA , ESBs, and Agile. Even before that we had integration brokers, client-server RPC (CORBA/DCOM/EJBs), EAI hubs, etc. Even though we've known that none of these are silver bullets, we latched onto these movements and rode them to glory. Or at least we thought.

I spend a lot of time talking with organizations about architecture, technology, open source, and how integration can help extract more value from their existing enterprise systems, as well as position them for future flexibility and adaptivity. Invariably, the discussion makes its way to microservices. This isn't surprising, since every person in the world with a computer seems to either blog or speak about them . "Will microservices help me?" I've been asked this question quite a few times over the past few months. There seems to be a general assumption that if we just do X because "they" are successful and are doing X, we'll be successful. So let's change things from SOAP to REST, application servers to Dropwizard, VMs to containers, and we'll be doing Microservices. And Microservices will help us move faster, scale to unicorn levels, be lightweight and agile, and be the cure-all we know doesn't exist but still hope that this time we've found it.

The answer to the question is "you're asking the wrong question." Adrian Cockcroft recently said: "people copy the process they see, but that is an artifact of a system that is evolving fast." You're not going to be a connected, adaptive, agile company delivering IT solutions the way company X does just because you copy what company X "appears" to do today. There are years of evolution, culture, and failures that led to what Company X does today and will do tomorrow. Unfortunately, you cannot skip those parts by sprinkling some technology makeup into your organization and IT systems.

So what can we do? We want to be an adaptive, agile organization and we need to integrate existing technology to align with it, but that's a bit harder than just choosing a "no-app-server" strategy.

Believe it or not, whether you call it microservices, SOA, client/server, etc., we are still tasked with getting systems to work together that may not have been designed to work together. To do this successfully, we have to focus deeply on the fundamental principles that will help us build agile, distributed systems and not get too caught up in hype. Domain modeling, boundaries, runtime dependencies, loose coupling, and culture/organizational structure are all prerequisites to building an adaptive organization, even in the face of constantly changing business and technological landscapes. Integration is a distributed-systems problem, so let's focus on some of the building blocks of successful distributed systems.

DOMAIN MODELING

As developers, we are intrigued (or downright giddy) with technology. With new technology unfolding every day, it's not hard to understand why. However, for most systems, the technology is not the complicated part: the domain is. Most developers I've spoken with aren't interested in becoming domain experts to facilitate building better software. It's not exactly a highly sought-after skill, either (search most job boards... do you see domain

knowledge or domain modeling high up on the list of requisites?). But domain modeling is a powerful and underused technique that is a vital prerequisite to building integrations and applications. We use models in our daily lives to simplify otherwise complex environments. For example, we use the GPS on our phones for navigating a city, but the map on our phones is a model geared toward accomplishing that goal. Using GPS and maps on our phones may not be a helpful model for navigating a battlefield. Understanding the nuances of the domain and where to elucidate ambiguities is key to iterating on your understanding of what the software should do and how to model it in your code. Each discussion with a domain expert should be reflected in the code so that they evolve together. Once you're able to uncover the right model for the purpose of the software, you're ready to explore the right boundaries.

BOUNDARIES

We deal with a lot of systems when we set out to build integrations, oftentimes with systems that were never designed to talk to each other. Those boundaries are fairly straightforward. But there are nuances in a domain that can cause ripple effects if not accounted for and designed explicitly with boundaries. For example, when I go to place an order on Amazon.com or Walmart.com, I can add things to my shopping cart and checkout. I will be prompted for payment information and delivery information and submit my order. So we could capture this as an "Order" in the domain and carry on. However, there's an obvious difference between how I place orders with these websites and, say, how a large corporation would place orders. Company A could place an order for 10,000 widgets from one of these online retailers, but they probably won't do it through the website the way I do; they'll most likely submit a purchase order. The process for placing an order for them is quite different. You can even throw in customer status (Gold, Silver, Bronze, etc.) as classifications that may impact how an order is placed and received in the system. If these concepts are not modeled directly, you can end up with a "canonical model," which becomes the source of constant conflict when changes are needed (or understandings of the model becomes clearer). Once you've established seams or boundaries around your models, you must think about how to expose them to collaborating agents. In other words, you must think about your integration relationships and how those are expressed.

APIS AND MODULARITY

Modularity isn't a new concept. Still, it seems difficult enough to get right that people bend (or blend) architectures and seams so they don't have to deal with modularity outright. *Oh you have an order system over there? Go ahead and share these implementations of Order objects.* No. Stop sharing domain code thinking it will save you time (or whatever your justification is), and focus instead on designing modules that hide their implementation details and expose only certain concepts over APIs or contracts. We want modules to be independent insofar as they can change their implementation details without affecting other modules. That's the goal. But it seems we get too preoccupied with defining the *right* API up front and focusing on WSDL-like contracts. Just like domain models and boundaries, APIs also evolve (like they must); you won't ever arrive at the right API "up front."

RUNTIME COUPLING

When we think of coupling, we tend to think of technological coupling. *Let's not use Java RMI, because that's a specific platform; Let's use XML or JSON instead. Or let's not inherit from dependency injection containers, because that ties us to the dependency injection framework (just in case we want to change that in the future).* These are all noble goals, but in my experience, we get overly focused on design-time or technology-specific coupling and forget about much bigger coupling phenomena. For example, the fallacies of distributed computing still hold here. The network is not a reliable transport. Our service collaborators may NOT receive our requests. They may not even be available. When you start to think "entity services," "activity services," "orchestration services," and have large chains of synchronous calls between services—turtles all the way down—you can start to see how this may break down. By designing proper models, boundaries, and APIs, we are aiming toward autonomously deployed and managed systems. But if you have large chains of synchronous calls like this, you've most likely got some bad runtime coupling: making changes to a service necessitates changes to other services you collaborate with (in a ripple effect), and if services are not available you run the risk of outages, etc. Asynchronous, publish-subscribe style architectures can alleviate some of this coupling.

"Will microservices help me?" The answer to the question is "you're asking the wrong question."

CONWAY'S LAW

In 1968, Melvin Conway wrote "organizations which design systems... are constrained to produce designs which are copies of the communication structures of these organizations," and he couldn't be more correct. Large organizations have been designed from the top down for one thing: efficiency. Following reductionist "scientific" management theories since the early 1900s, our companies have been focused on reducing variability and optimizing each part of the organization. Which is why, not surprisingly, in IT organizations we have "DBAs" and "UI experts" and "QA teams." Each team focuses on its area of specialization with scrutiny and optimization. Then, following Conway's Law, we have three-tier applications—or "layers"—that correspond with each of those teams: the UI layer, the Business Logic layer, the Database Layer, and so on. Then we throw things over the fence to Ops and QA, etc. Although this may be the hardest thing to evolve, the organizational structure and the culture of its employees has the most profound implication on how we build our distributed systems. If we want to be an adaptive, connected company, we need to explore what that means to our organizational management philosophies and structures. Then, as a corollary, we have a much better chance at building truly autonomous, decoupled systems that can scale individually, adapt to failures and adverse conditions, and change to meet market challenges.

Without these fundamentals at the forefront, we run the risk of rabidly adopting the latest and greatest fads and choking our businesses in the area they need to be most adaptive: IT and technology.

CHRISTIAN POSTA (@christianposta) is a Principal Middleware Specialist/Architect at Red Hat and well known for being a frequent blogger, speaker, open-source enthusiast, and committer on Apache ActiveMQ and Apache Camel and others. Christian has spent a great deal of time working with large companies creating and deploying large scale distributed architectures—many of which are now called Microservices based. He enjoys mentoring, training and leading teams to be successful with distributed systems concepts, microservices, DevOps, and cloud-native application design.

Navigating SOAP to REST Migrations Like a Pro

For a decade, mainstream API development has been in a torrid transitionary period over how API technologies connect the world. SOAP and REST—two dominant API design paradigms—are at odds, both technically and strategically.

The decade-long conflict between modern, minimalist patterns employed by RESTful APIs and older SOAP-style enterprise service-oriented architecture presents two important challenges for businesses looking to employ faster methodologies on integration projects:

1. Translation between XML-heavy SOAP web services and JSON-centric REST APIs
2. Professional skills and tooling differences between SOAP and REST development practices

Enterprises delivering reliable integrations face serious challenges in the mixed landscape of API technologies; both people and tools must align to your API strategy.

HOW LONG WILL SOAP STILL BE AROUND?

Modern web and mobile markets are pushing people to learn new skills and employ new tools. Since 2011, many businesses have been making the switch internally and externally from SOAP and SOA to API strategies that assume more modern RESTful patterns.

This switch comes at a cost: lack of standards around security, identity, and interoperability hinder rapid migration, but REST has been catching up quickly, as seen in open-source specifications like Swagger, JSON-Schema, JSON-LD, JSON API, and other solutions.

BOTTOM LINE: ENTERPRISES MUST STILL SUPPORT A MIXED INTEGRATION LANDSCAPE

Things we build have a tendency to stick around, as is the case with SOAP in enterprises. REST-style APIs are now the preferred approach when building new systems intended to replace legacy integrations, but enterprise API professionals need to be adept at both SOAP and REST, carrying with them tools that also traverse multiple architectural styles quickly and flawlessly.

People are the driving force behind great technology. Getting your team dynamics right is as important to shipping accurate, safe, and reliable APIs as getting your toolchain streamlined; both work hand in hand. The better your enterprise is at people and tools, the better your APIs will be.

WRITTEN BY PAUL BRUCE
API PRODUCT MANAGER, SMARTBEAR

Ready! API by SmartBear

Ready! API

SmartBear helps you to deliver enterprise-ready APIs with the world's best SOAP/REST design, testing, virtualization, and performance monitoring solutions in a single platform: Ready! API.

CATEGORY
API Lifecycle and Quality Tools

NEW RELEASES
Quarterly

OPEN SOURCE?
Commercial and Open Source

CASE STUDY

Healthcare Data Solutions (part of IMS Health) aggregates large data sets from healthcare providers, using APIs to effectively deliver this data. The time required for the software QA team to test multiple data sets, set up testing artifacts, and diagnose root causes of issues impeded its ability to release software updates. "At one point we had to delay the release of a project for a whole fiscal quarter, which had significant ripple effects on other priorities. It sometimes took us two or three days just to set up our testing processes."

Since deploying SoapUI NG Pro, HDS has gained the ability to data-drive automated API tests—reducing the set-up of their initial testing processes by 80%.

With SoapUI NG Pro, HDS sees "improvements that put us onto the path of even better testing coverage. We knew capabilities such as these would significantly streamline our API testing processes."

STRENGTHS

- Comprehensive capabilities for testing SOAP and REST APIs in SoapUI NG Pro
- API mocking and lightweight service virtualization through ServiceV Pro
- Easy-to-employ API load testing for on-premise and cloud-based services
- API-specific security scans to check for common vulnerabilities
- Integration to API Management, Issue Tracking, Version Control, & description formats

NOTABLE CUSTOMERS

- Intel
- Microsoft
- GE
- Disney
- Cisco
- Bank of America
- Johnson & Johnson
- American Airlines

BLOG blog.smartbear.com

TWITTER @ready_api

WEBSITE smartbear.com

State as a Necessary Evil

BY TYLER TREAT

QUICK VIEW

01 Abstractions are dangerous, especially when it comes to distributed systems.

02 Rather than hiding complexity, we need to engage with it using better tools and patterns.

03 Access patterns drive architecture, and going distributed is not a requirement for system maturity.

04 State is a necessary evil. The more stateful your system is, the harder it is to scale.

05 There is a lot of interesting research going on which can be leveraged to build scalable, stateful services.

MORE AND MORE COMPANIES ARE DESCRIBING THEIR SUCCESS STORIES REGARDING THE SWITCH TO A SERVICE-oriented architecture. As with any technological upswing, there's a clear and palpable hype factor involved (Big Data™ or The Cloud™ anyone?), but obviously it's not just fluff.

While microservices and SOA have seen a staggering rate of adoption in recent years, the mindset of developers often seems to be stuck in the past. I think this is, at least in part, because we seek a mental model we can reason about. It's why we build abstractions in the first place. In a sense, I would argue there's a comparison to be made between the explosion of OOP in the early 90's and today's SOA trend. After all, SOA is as much about people scale as it is about workload scale, so it makes sense from an organizational perspective.

THE PERILS OF GOOD ABSTRACTIONS

While systems are becoming more and more distributed, abstractions are attempting to make them less and less complex. Mesosphere is a perfect example of this, attempting to provide the "datacenter operating system." Apache Mesos allows you to "program against your datacenter like it's a single pool of resources." It's an appealing proposition to say the least. PaaS-like Google App Engine and Heroku offer similar abstractions—write your code without thinking about scale. The problem is you *absolutely* have to think about scale or you're bound to run into problems down the road. And while these abstractions are nice, they can be dangerous just the same. Welcome to the perils of good abstractions.

I like to talk about App Engine because I have firsthand experience with it. It's an easy sell for startups. It handles spinning up instances when you need them, turning them down when you don't. It's your app server, database, caching, job scheduler, and task queue all in one, and it does it at scale. There's vendor lock-in, sure, yet it means no ops, no sysadmins, no overhead. Push to deploy. But it's a leaky abstraction. It has to be. App Engine scales because it's distributed, but it allows—no, *encourages*—you to write your system as a monolith. The datastore, memcache, and task queue accesses are masked as RPCs. This is great for our developer mental model, but it will bite you if you're not careful.

RPC is consistently at odds with distributed systems. I would go so far as to say it's an *anti-pattern* in many cases. RPC encourages writing synchronous code, but distributed systems are inherently *asynchronous*. The network is not reliable. The network is not fast. The network is not your friend. Developers who either don't understand this or don't realize what's happening when they make an RPC will write code as if they were calling a function. It will sure as hell *look* like just calling a function. When we think synchronously, we end up with systems that are slow, fault *in*tolerant, and generally not scalable. To be quite honest, however, this is perfectly acceptable for 90% of startups as they are getting off the ground because they don't have workloads at meaningful scale.

There's certainly some irony here. One of the selling points of App Engine is its ability to scale to large amounts of traffic, yet the vast majority of startups would be perfectly suited to scaling up rather than out, perhaps with some failover in place for good measure. Stack Overflow is the poster child of scale-up architecture. In truth, your architecture should be a function of your access patterns, not the other way around (and App Engine is very much

tailored to a specific set of access patterns). Nonetheless, it shows that vertical scaling can work. I would bet a lot of startups could sufficiently run on a large, adequately specced machine, or maybe a small handful of them.

PEERING THROUGH THE ABSTRACTION

Eventually SOA makes sense, but our abstractions can cause problems if we don't understand what's going on behind the curtain (hence the leakiness). Partial failure is all but guaranteed, and latency, partitioning, and other network pressure happen all the time. We need to *engage* with this complexity using better tools and patterns. Abstractions are nice, but it's important we don't treat them as a fire blanket draped over a pile of burning tires.

Ken Arnold is famed with once saying "state is hell" in reference to designing distributed systems. In the past, I've written how scaling shared data is hard, but with SOA it's practically a requirement. Ken is right though—state *is* hell, and SOA is fundamentally competing with consistency. The FLP Impossibility result and the CAP theorem can prove it formally, but really this should be intuitively obvious if we accept the laws of physics.

"Essentially, the more stateful your system is, the harder it's going to be to scale it because distributing that state introduces a rich tapestry of problems."

"On the other hand, if you store information that I can't reconstruct, then a whole host of questions suddenly surface. One question is, "Are you now a single point of failure?" I have to talk to you now. I can't talk to anyone else. So what happens if you go down?

"To deal with that, you could be replicated. But now you have to worry about replication strategies. What if I talk to one replicant and modify some data, then I talk to another? Is that modification guaranteed to have already arrived there? What is the replication strategy? What kind of consistency do you need—tight or loose? What happens if the network gets partitioned and the replicants can't talk to each other?"

Essentially, the more stateful your system is, the harder it's going to be to scale it because distributing that state introduces a rich tapestry of problems. In practice, we often can't eliminate state wholesale, but basically everything that can be stateless should be stateless.

Making servers disposable allows you a great deal of flexibility. Former Netflix Cloud Architect Adrian Cockcroft articulates this idea well:

"You want to think of servers like cattle, not pets. If you have a machine in production that performs a specialized function, and you know it by name, and everyone gets sad when it goes down, it's a pet. Instead you should think of your servers like a herd of cows. What you care about is how many gallons of milk you get."

This is effectively how App Engine achieves its scalability. With lightweight, stateless, and disposable instances, it can spin them up and down on the fly without worrying about being in an invalid state.

App Engine also relies on eventual consistency as the default model for data-store interactions. This makes queries fast and highly available, while serializable isolation can be achieved using entity-group transactions if necessary. The latter, of course, can result in a lot of contention and latency. Yet, people seem to have a hard time grappling with the reality of eventual consistency in distributed systems. State is hell, but calling SOA "satan" is clearly a hyperbole. It is a tough problem nevertheless.

A STATE OF MIND

In the situations where we need state, we have to reconcile with the realities of distributed systems. This means understanding the limitations and accepting the complexities, not papering over them. It *doesn't* mean throwing away abstractions. Fortunately, distributed computing is the focus of a lot of great research, so there are primitives with which we can build: immutability, causal ordering, strong eventual consistency, CRDTs, and other ideas.

As long as we recognize the trade-offs, we can design around them. The crux is knowing they exist in the first place. We can't have ACID semantics while remaining highly available, but we can use Highly Available Transactions to provide strong-enough guarantees. At the same time, not all operations require coordination or concurrency control. The sooner we view eventual consistency as a solution and not a consequence, the sooner we can let go of this existential crisis. Other interesting research includes BOOM, which seeks to provide a high-level, declarative approach to distributed programming.

State might be hell, but it's a hell we have to live. I don't advocate an all-out microservice architecture for a company just getting its start. The complications far outweigh any benefits to be gained, but it becomes a necessity at a certain point. The key is having an exit strategy. PaaS providers make this difficult due to vendor lock-in and architectural constraints. Weigh their advantages carefully.

Once you *do* transition to an SOA, make as many of those services, or the pieces backing them, as stateless as possible. For those that aren't stateless, know that the problem typically isn't novel. These problems have been solved or are continuing to be solved in new and interesting ways. Academic research is naturally at the bleeding edge while the industry is often lagging behind. OOP concepts date back to as early as the '60s, but it didn't gain widespread adoption until several decades later. Distributed computing is no different. SOA is just a state of mind.

TYLER TREAT is a senior backend engineer and tech lead for Workiva's messaging team. He works primarily on distributed systems and platform infrastructure. In his spare time, Tyler contributes to various open-source projects while blogging about software engineering, system design, and distributed computing.

Bypass Roadblocks With Cloud Integration

BY CHRIS HADDAD

QUICK VIEW

01
A cloud integration approach rapidly creates connections between users, their devices, their apps, and valuable business APIs

02
A cloud integration approach ideally will rely on integration triggers, webhooks, publish-subscribe channels, and event-based sourcing.

03
When selecting a cloud integration service, evaluate whether the vendor excels in the following areas:

- Intuitive Developer Experience
- Pre-built Integration Blocks
- DevOps Visibility
- Topology Flexibility

IN TODAY'S DIGITAL BUSINESS, TEAMS WANT TO CONNECT EVERYTHING: WEAVE MULTIPLE APPS, APIS, DATA, AND devices into a complete user experience that automates tasks and provides unprecedented access to data and insights. Old-school integration approaches don't scale to address hyper-connected, multi-system integration scenarios, and reductionist IT approaches to simplify integration—by forcing adoption into a single, comprehensive ERP suite, central enterprise service bus (ESB), or massive data warehouse—fail to fulfill evolving business requirements. When teams rely on a central bus, warehouse, or business platform, they experience an unwieldy, expensive, and monolithic environment, which inhibits business agility and IT delivery.

In contrast, a cloud integration approach rapidly creates connections between users, their devices, their apps, and valuable business APIs. By decentralizing integration actions, providing snap-in integration templates, delivering ready-made app and API bridges, and offering graphical integration workbenches, cloud integration services reduce reliance on SOA specialists and democratize integration development. Teams using an integration Platform as a Service (iPaaS) can easily create sophisticated integration workflows linking several apps, APIs, and systems in a fraction of the traditional work effort and time frame. Integration at the edge and in the cloud (instead of through a central integration server) provides an opportunity to increase social media, mobile, and Internet of Things connections by orders of magnitude without incurring expensive cost, high-risk single points of failure, or operations management conflicts.

Because many connections start and end outside an on-premise environment, cloud integration services are becoming a popular and prevalent integration option. Cloud integration services don't force data movement on-premise, and are the preferred alternative when moving data or integrating processes across multiple cloud regions. They are the best option when connecting:

- Mobile applications or 'in-the-cloud' applications to cloud APIs
- Software as a Service to 'in-the-cloud' applications or cloud APIs
- IoT devices to cloud analytics
- In-the-cloud applications to cloud APIs

CLOUD-TO-CLOUD INTEGRATION FAST LANE

Cloud-to-cloud integration solutions create a fast lane towards project delivery and business value. Cloud integration solutions overcome on-premise integration barriers. For example, on-premise integration solutions require evaluating, procuring, and deploying complicated and expensive infrastructure (e.g., Enterprise Service Bus, gateways, API Management Platforms). Many teams don't properly size on-premise solutions or know how to monitor and manage the environment, which results in low reliability, poor availability, and reduced scalability. Additionally, on-premise solutions force integration traffic through specific geographical locations (the on-premise data center) and require extra work to lock down network security policies and zones.

In contrast, cloud-to-cloud integration solutions provide instant on-boarding, self-service configurations, high Quality of Service (e.g., reliability, availability, scalability), and allow teams to cost effectively grow their business while defocusing attention on lower level integration infrastructures. Instead of acquiring hardware, building virtual images, and implementing operations management processes, integration teams simply select the appropriate service level agreement (SLA) and start building connections. Cloud-to-cloud integration platforms are fully-managed by the solution provider. The provider offloads tedious work; applying software updates, tuning performance, managing capacity, and monitoring security.

ADOPTION CHALLENGES

Every architecture and technology choice is a tradeoff. Cloud-to-cloud integration solutions must address regulatory compliance,

prescriptive technology environments, and data gravity. Regulatory compliance mandates may require keeping the integration processing within specific sovereign geographical boundaries (e.g. the EU) or execute within certified environments (e.g. PCI). Prescriptive technology services may not allow general-purpose code execution (e.g. Java, C#) or allow connections over legacy protocols (e.g. JMS, MQ). Data gravity—the concept that moving large amounts of data to augment an integration message can be prohibitively expensive—may require cloud-to-cloud integration projects to rely on integration platforms that can flexibly deploy integration agents close to the data and enable smart routing.

CLOUD INTEGRATION BEST PRACTICES

Today's projects span a distributed persistence and process architecture. Cloud integration best practices connect distributed data stores, apps, microservices, and APIs using a loosely-coupled approach. The approach ideally will rely on integration triggers, webhooks, publish-subscribe channels, and event-based sourcing.

IFTTT pioneered straight-forward integration triggers. When a message comes in, it processes the message and conditionally triggers a distributed action. For example, when I update a document in Google Docs, IFTTT triggers notification alerts to document reviewers.

Webhooks are the equivalent of object callbacks. By registering a webhook URL with a data source or integration hub, the integration solution can notify participants when key integration events occur. For example, when a report job has completed, or when the order status changes, the cloud integration service will push the data in real-time or trigger an action.

Publish-subscribe channels compliment webhooks by registering information queues that endpoints can access to pull data, messages, or events. A cloud-based publish-subscribe endpoint offers an always-available channel that can be accessed by all Internet connected apps, APIs, and devices.

Event-based sourcing keeps distributed data stores synchronized. By sending change events across participants (instead of complete data records), each participant can completely rebuild the data store by re-running the events, rewinding the data to a prior temporal state, or replaying events to build a prior temporal view.

CLOUD-TO-CLOUD PROVIDER EVALUATION MATRIX

A new breed of cloud integration startups are delivering innovative cloud services that reduce integration project work effort and shorten delivery time frames. When selecting a cloud integration service, evaluate whether the vendor excels in the following areas:

- Intuitive Developer Experience
- Pre-Built Integration Blocks
- DevOps Visibility
- Topology Flexibility

INTUITIVE DEVELOPER EXPERIENCE

Democratize integration development, enlist citizen integrators, and reduce reliance on integration specialists by choosing solutions offering an intuitive developer experience. The solution should provide a drag-and-drop user experience, enable project self service, and visually map message formats and flows. Evaluate whether the development workbench intuitively promotes building integration workflows with webhooks, triggers, mediation actions, and pre-built connectors.

PRE-BUILT INTEGRATION BLOCKS

Building sophisticated integration flows that tie together multiple apps and APIs is always challenging. Reduce time and effort by selecting an environment offering an extensive library of pre-built connectivity to APIs, apps, and business intelligence tools. The environment should also offer higher-level templates, recipes, and flows. The integration library or cloud service API marketplace should offer hundreds of pre-built integration blocks.

Standard data integration-operations to import, export, cleanse, replicate, and synchronize should be readily available. More sophisticated systems offer pre-built mappings between common apps, APIs, and a canonical model. Ontologies and semantics are used to flexibly connect source messages with target message formats.

DEVOPS VISIBILITY

Moving integration flows through on-premise environments provides high visibility into message movement, integration flow status, and overall performance. Cloud-based integration solutions must provide robust error management and comprehensive performance monitoring, and must correlate interactions with end-to-end transactions.

Teams will want to schedule integration actions, define appropriate Quality of Service (QoS) policies, and report on service level agreement compliance.

TOPOLOGY FLEXIBILITY

Early generation cloud-to-cloud service offerings forced all traffic through a central cloud location. Today's hybrid deployment models respect data gravity and will flexibly deploy and co-locate integration execution agents close to where the data or application resides. While monitoring and transaction correlation information is posted into the cloud, data is accessed, filtered, reduced, and transformed close to the source, reducing network bandwidth and latency.

PLATFORM ALTERNATIVES

New-school integration solutions are available in the cloud, offer an intuitive user experience, promote cloud integration patterns (e.g. webhooks, triggers, API-driven integration), and provide rich connections to applications and cloud APIs. A short list of innovative integration platform alternatives include:

- ClearStory Data
- Cloud Elements
- IFTTT
- Jitterbit
- Mulesoft Anypoint Platform
- Snaplogic
- Skyvia
- Syncano
- Zapier

If you are embarking on a new greenfield integration project or building the next-generation integration platform to serve multiple projects, take time to re-think legacy integration approaches and evaluate new cloud-to-cloud integration service offerings.

CHRIS HADDAD helps teams select and implement cloud integration solutions that address legacy challenges and create hyper-connectivity across apps, devices, data, and APIs. He works with teams to connect everything; weave multiple apps, APIs, data, and devices into a complete user experience that automates tasks and provides unprecedented access to data and insights.

Executive Insights on Enterprise Integration

BY TOM SMITH

QUICK VIEW

01
APIs are driving the present and future of enterprise integration until someone identifies an easier way of getting disparate architectures, frameworks, and software to talk to each other

02
The opportunities for integration—and need to integrate—will only increase as data needs increase and companies realize the benefits of integrating data from different silos and enterprises

03
The ultimate beneficiaries of integration are employees and customers who will have an enhanced user experience (UX) and consequently an enhanced customer experience (CX)

TO MORE THOROUGHLY UNDERSTAND THE STATE OF ENTERPRISE INTEGRATION, AND WHERE IT'S GOING, we interviewed 20 executives with diverse backgrounds and experience integrating technologies and databases for enterprise clients. Specifically, we spoke to:

Asanka Abeysinghe, V.P. Solutions Architecture, WSO2 • **James Jinnette**, Director of Information Technology, unidentified CRO • **Zeev Avidan**, V.P. Product Management, OpenLegacy • **Phil Manfredi**, **Aaron Sandeen**, and **Kiran Chinnagangannagari**, Co-Founders, Zuggand • **Nishant Patel**, CTO, and **Matthew Baier**, COO, Built.io • **Adam Fingerman**, Chief Experience Officer, ArcTouch • **Jon Gelsey**, CEO, Auth0 • **Jon Bock**, Vice President Products, Snowflake Computing • **Tyson Whitten**, API Management Product Marketing, CA Technologies • **Florent Solt**, CTO, Netvibes • **Andrew Warfield**, CTO and Co-Founder, Coho Data • **Mike Han**, V.P. Operations, Liferay • **Sean Bowen**, CEO, Push Technology • **Suchit Bachalli**, President, Unilog • **Uri Sarid**, CTO, and **Conor Curlett**, Principal Architect, MuleSoft • **Gabe Stanek**, Director of Field Engineering, Neo Technology

Here's what we learned from the executives:

01

There's little agreement about the most important elements of enterprise integration; however, four themes emerged from the executives with whom we spoke:

- **Data accessibility.** It's important to enable enterprises to access external data sources and integrate that data with internal sources. This requires unlocking data and making it accessible across the enterprise. We must be able to converge traditional enterprise data with large data stacks to solve business problems.
- **API management.** APIs in the cloud have accelerated enterprise integration projects from taking years to months. APIs are also driving alignment and integration across the enterprise to enable once siloed data to be integrated for analysis.
- **Focus on simplicity.** Make integration as seamless and simple as possible, since you will be adding more and more resources that need to talk with each other as the company sees the benefits of integration. The more the business sees, the more it will want.
- The integration platform must be **secure, stable, and reliable**. Extensibility is key as demand for more integration grows. Have a well-documented protocol so it's easy to make changes and additions. Use templates, samples, and standardization to make changes as painless as possible. Also, consider a data structure that decouples everything—including the business logic from APIs—as this will accelerate integration and flow of data.

02

The most important players in enterprise integration are **clearly IBM and Oracle**, as these two firms were mentioned more than twice as frequently as any other company. Both are recognized for their robust messaging middleware. MuleSoft was also mentioned as a major contributor to the open-source middleware space along with WSO2. TIBCO was the only other company mentioned more than once. Other companies mentioned were Amazon, CA Technologies, Dell, EMC, HP, Metadata, Microsoft, NetApp, SAP, and Snaplogic. Other than IBM and Oracle, enterprise integration providers are fragmented, and the market is competitive given all the money flowing into the space.

03

Customers are the most frequently mentioned source for executives to stay abreast of what's going on in the industry. This includes direct interaction with customers and prospects, fielding questions, and responding to requests for proposals. Analysts, like Gartner and Forrester, and social media, like Twitter, were the second most frequently mentioned sources of information. Other sources include conferences and events, consultants, peers, employees, universities, Flipboard, and Audible.com.

04

The greatest value of enterprise integration is being seen in two areas: 1) legacy data able to be accessed to **solve business problems**; and 2) using data to **improve the customer and employee experience**. Giving employees access to legacy data fosters innovation and empowers employees to solve business problems themselves. Unlocking data can transform a business (e.g. Airbnb and Uber). Integration of previously siloed data enables people to make more intelligent decisions.

At the same time, data can be used to improve the customer experience. By giving employees a 360-degree view of the customer, companies are able to make proactive recommendations, making customers' lives simpler and easier. Social listening enables customer-facing employees to address customer concerns in a timely manner either in person or virtually via mobile devices.

05

The skills that make someone good at enterprise integration are **broad reaching**. It starts with knowing the problem you're trying to solve and then being able to identify the optimal technical solution. Superior performers also understand patterns, scenarios, web protocols, frameworks, and architectures. Problem-solving skills are beneficial, as is understanding the fundamental pain points the technology addresses. It also helps to have familiarity with the systems you are integrating (e.g. CRM, ERP, etc.). The most successful have the skill to see the "butterfly effect"—if I change this, here's how it will affect my client and their customers.

06

The **most frequently used programming language continues to be Java** followed closely by JavaScript. Other languages, platforms, operating systems, or frameworks that were mentioned included Android, C, C++, iOS, Node.js, .NET and Scala.

07

The confluence of APIs, the cloud, mobile, and the Internet of Things (IoT) are driving the evolution of enterprise integration. Unlike distributed computing, mobile and cloud have standards for interconnection. The shift to mobile and API platform services have centralized the work that needs to be done. APIs and integration are critical for IoT to achieve its projected growth levels.

Everything is able to talk to everything else in the cloud, thanks much to RESTful design. The cloud has removed the need for much hardware infrastructure and funding. APIs have made everything faster, reduced costs, and increased speed to market. The data center can now reside solely in the cloud—though the pendulum may be swinging, as some companies prefer hybrid solutions where they have an on-premise appliance, with on-demand, "burst" needs met in the cloud. With all of the devices, connections, and APIs, we're seeing a renaissance around messaging; however, this time it's data rather than voice.

08

Obstacles to success of enterprise integration projects at client sites seem to revolve around **unrealistic expectations** and the failure by vendors to do **proper due diligence** before proposing a solution. It's critical for the vendor to understand the business problem they are solving and the potential for that problem—and solution—to scale over time. Understanding the business problem will help the vendor identify the optimal solution, versus a more complex solution than is necessary. Understanding the potential for scalability will ensure the vendor provides a solution that can scale to meet the client's needs over time without latency becoming an issue.

Some vendors are chasing the money over promising what they can deliver, or providing more complex, expensive solutions than are necessary. Hype can get in the way and create unrealistic expectations for clients. People have a tendency to focus on the short cycle times of Agile without considering the long-term implications of their decisions or the extensibility of the platform.

09

The primary concern around enterprise integration is **explosive complexity and extensibility**. We must be conscientious of the data we're collecting and sending, and ensure we are addressing a business purpose in doing so. Other concerns are companies that are building closed versus open solutions, as well as on-going concerns with security as more and more devices are brought to market. Failure to adhere to proven patterns and architectures will lead to short-cuts, which lead to security flaws. Lastly, moving the enterprise to the cloud is not as easy as it sounds—is it really in the business's best interest to move their entire enterprise into the cloud, or is a hybrid solution more appropriate based on business needs?

10

The future of enterprise integration appears to be **centered around APIs and the ease of accessibility they promise in the cloud**—whether it's robustness or steady accessibility. We are beginning to see the internet of APIs. This is driven by IoT and mobile, with mobile leading the charge right now and IoT on its heels. To ensure successful growth, we need to look for opportunities to standardize platforms that can scale and maintain security. Doing so will result in an improved employee and customer experience.

11

The key advice for developers is to **keep enterprise integration as simple as possible**. Don't try to "boil the ocean." Focus on the business objective, know the business problem, know the business process, and make it as easy as possible for the business to understand and use. Know the user needs for the problem you're solving—the needs of engineering are very different from those of finance or marketing. Know what middleware is available before you start writing code. Lastly, with regards to mobile, know the value of two bytes. This will add up to a lot of savings—of bandwidth and money—over the next few years.

The executives we spoke with are working on their own products or serving clients. We're interested in hearing from developers, and other IT professionals, to see if these insights offer real value. Is it helpful to see what other companies are working on from a senior industry-level perspective? Do their insights resonate with what you're experiencing at your firm?

We welcome your feedback at research@dzone.com.

TOM SMITH is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.

MICROSERVICES QUICK REFERENCE

"Microservices" has emerged as a term to describe the components of applications that are decomposed or initially built with single-purpose, loosely-coupled services managed by cross-functional teams. The idea of microservices has evolved from recent trends focused on increasing the speed and efficiency of developing and managing software solutions, including Agile methods, DevOps culture, PaaS, application containers, and the widespread adoption of Continuous Integration and Continuous Delivery. This reference serves as a quick reminder of the essential principles and benefits of microservices. Thanks to Arun Gupta, author of DZone's Getting Started With Microservices Refcard, for his help assembling this reference.

KEY CHARACTERISTICS OF MICROSERVICES

1 DOMAIN-DRIVEN DESIGN
Functional decomposition can be easily achieved using Eric Evans' DDD approach and his concept of Bounded Contexts.

2 SINGLE RESPONSIBILITY PRINCIPLE
Each service is responsible for a single part of the functionality, and does it well.

3 EXPLICITLY PUBLISHED INTERFACE
A producer service publishes an interface that is used by a consumer service.

4 INDEPENDENT DURS (DEPLOY, UPDATE, REPLACE, SCALE)
Each service can be independently deployed, updated, replaced, and scaled.

5 LIGHTWEIGHT COMMUNICATION
REST over HTTP, STOMP over WebSocket, and other similar lightweight protocols are used for communication between services.

BENEFITS OF MICROSERVICES

1 INDEPENDENT SCALING
Each microservice can scale independently via sharding and/or cloning with more CPU or memory.

2 INDEPENDENT UPGRADES
Each service can be deployed independent of other services. Developers can easily make changes to services without having to coordinate with other teams.

3 EASY MAINTENANCE
Each microservice is restricted to one function/ feature, making the code more readable and compact, improving load times.

4 POTENTIAL HETEROGENEITY AND POLYGLOTISM
Developers are free to pick the language and stack that are best suited for their service.

5 FAILURE AND RESOURCE ISOLATION
A failing service, whether it's caused by a memory leak or an unclosed database connection, will not affect the rest of the application or cause it to go down.

6 IMPROVED COMMUNICATION ACROSS TEAMS
A microservice is typically built by a full-stack team. All members related to a domain work together in a single team, which significantly improves collaboration and communication efficiency.

OPERATIONAL REQUIREMENTS FOR MICROSERVICES

1 SERVICE REPLICATION
Each service needs to replicate, typically using cloning or partitioning. There should be a standard mechanism by which services can easily scale based on metadata. A PaaS can simplify this functionality.

2 SERVICE DISCOVERY
In a microservice world, multiple services are typically distributed in a PaaS environment. Immutable infrastructure is provided by containers or immutable VM images. Services may scale up and down based on certain pre-defined metrics, and the exact address of a service may not be known until the service is deployed and ready to be used. The dynamic nature of a service's endpoint address is handled by service registration and discovery tools.

3 SERVICE MONITORING
Service monitoring and logging allow stakeholders to take proactive action if, for example, a service is consuming unexpected resources. There are open source tools that can aggregate logs from different microservices, provide a consistent visualization over them, and make that data available to business users.

4 RESILIENCY
Software failure will occur, so it's important for services to automatically take corrective action to ensure user experience is not impacted. The Circuit Breaker pattern allows you to build resilient software.

5 DEVOPS
Continuous Integration and Continuous Deployment (CI/CD) are very important in order for microservices-based applications to succeed. These practices are required so that errors are identified early, and so little to no coordination is required between different teams building different microservices.

Solutions Directory

This directory includes integration tools for API management, integration platforms, message queues, frameworks, and SOA governance, letting you find the solution that's right for your integration needs. Solutions are selected for inclusion in the directory based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
3Scale API Management	API Management	No	3scale.com
Actian DataCloud	Integration Platform/PaaS	No	actian.com
ActiveMQ	Message Queue	Yes	activemq.apache.org
Adaptris Interlok	Integration Platform/PaaS	No	adaptris.com
Adeptia Integration Suite	Integration Platform/PaaS	No	adeptia.com
Akana API Management	API Management	No	akana.com
Akana Software Suite	SOA Governance	No	akana.com
Amazon SQS	Message Queue	No	aws.amazon.com
Anypoint Platform by Mulesoft	API Management	No	mulesoft.com
Apache Camel	Integration Framework	Yes	camel.apache.org
API Manager Platform by Cloud Elements	API Management	No	cloud-elements.com
Apigee Edge	API Management	No	apigee.com
apiman	API Management	Yes	apiman.io
Artix by Micro Focus	ESB	No	microfocus.com
AtomSphere by Dell Boomi	Integration Platform/PaaS	No	boomi.com
Aurea Sonic ESB	ESB	Yes	aurea.com
Axway API Management	API Management	No	axway.com
Azuqua	Integration Platform/PaaS	No	azuqua.com
BizTalk Server by Microsoft	Integration Platform/PaaS	No	microsoft.com
Built.io Flow	Integration Platform/PaaS	Yes	built.io
CA API Management	API Management	No	ca.com
CentraSite by SoftwareAG	SOA Governance	No	softwareag.com
Cloak Labs	Integration Platform/PaaS	No	cloaklabs.com
CloudHub by Mulesoft	Integration Platform/PaaS	No	mulesoft.com
CX Messenger by Aurea	ESB	No	aurea.com
DreamFactory	API Management	No	dreamfactory.com
E2E Bridge	Integration Platform/PaaS	No	e2ebridge.com

PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
Elastic Integration Platform by SnapLogic	Integration Platform/PaaS	No	snaplogic.com
Ensemble by Intersystems	ESB	No	intersystems.com
evolution by Mertech	API Management	No	mertech.com
Fabric8	Integration Platform/PaaS	Yes	fabric.io
Fanout Zurl	API Management	Yes	fanout.io
Fiorano API Management	API Management	No	fiorano.com
Fiorano ESB	ESB	No	fiorano.com
Fiorano Integration	SOA Governance	No	fiorano.com
FioranoMQ	Message Queue	No	fiorano.com
Flowgear	Integration Platform/PaaS	No	flowgear.net
Fujitsu Business Operations Platform	Integration Platform/PaaS	No	fujitsu.com
GXS Enterprise Gateway by OpenText	ESB	No	gxs.co.uk
HANA Cloud Integration by SAP	Integration Platform/PaaS	No	sap.com
HornetQ	Message Queue	Yes	hornetq.jboss.org
HP SOA Systinet	SOA Governance	No	hp.com
IBM API Management	API Management	No	ibm.com
IBM Integration Bus	ESB	No	ibm.com
IBM MQ Advanced	Message Queue	No	ibm.com
IBM Tivoli	SOA Governance	No	ibm.com
ILANTUS Xpress Governance	SOA Governance	No	ilantus.com
Informatica iPaaS	Integration Platform/PaaS	No	informatica.com
InterSystems SOA	SOA Governance	No	intersystems.com
IronMQ	Message Queue	No	iron.io
JaxView by Managed Methods	SOA Governance	No	managedmethods.com
JBoss Fuse	ESB	Yes	jboss.org
Jitterbit	Integration Platform/PaaS	No	jitterbit.com
JNBridge	Integration Framework	No	jnbridge.com
JustAPIs by Anypresence	API Management	No	anypresence.com
Kapow Data Integration Platform by Kofax	Integration Platform/PaaS	No	kofax.com
LegaSuite Integration	Integration Platform/PaaS	No	rocketsoftware.com
Liaison Alloy	Integration Platform/PaaS	No	liaison.com
Mashery API Management by Intel	API Management	No	mashery.com
Microsoft Azure API Management	API Management	No	microsoft.com
MID Innovator	SOA Governance	No	mid.de
Mule ESB	ESB	Yes	mulesoft.com
Neuron ESB	ESB	No	neuronesb.com

PRODUCT	CATEGORY	FREE TRIAL	WEBSITE
NServiceBus by Particular Software	ESB	Yes	particular.net
Oracle Service Bus	ESB	Yes	oracle.com
Oracle SOA Suite	SOA Governance	No	oracle.com
Point.io	API Management	No	point.io
Pokitdot	API Management	No	pokitdot.com
ProSyst OSGi by Bosch Software Innovations	API Management	No	prosyst.com
RabbitMQ by Pivotal	Message Queue	Yes	rabbitmq.com
Ready! API by SmartBear	API Lifecycle and Quality Tools	No	smartbear.com
Redis	Message Queue	Yes	redis.io
Restlet Framework	API Management	Yes	restlet.com
RunMyProcess by Fujitsu	Integration Platform/PaaS	Yes	fujitsu.com
Runscope	API Lifecycle and Quality Tools	No	runscope.com
SAIFE	Message Queue	No	saifeinc.com
SAP API Management	API Management	No	sap.com
SAP HANA	Integration Platform/PaaS	No	sap.com
SAP NetWeaver	SOA Governance	No	sap.com
Seeburger	Integration Platform/PaaS	No	seeburger.com
Sikka Software	API Management	No	sikkasoft.com
Spring Integration by Pivotal	Integration Framework	Yes	spring.io
SQS by Amazon	Message Queue	No	aws.amazon.com
SwarmESB	ESB	Yes	swarmesb.com
Talend API Management	API Management	No	talend.com
Talend ESB	ESB	Yes	talend.com
Talend Integration Cloud	Integration Platform/PaaS	No	talend.com
TIBCO	API Management	No	tibco.com
TIBCO ActiveMatrix	Integration Platform/PaaS	No	tibco.com
TIBCO ActiveMatrix	SOA Governance	No	tibco.com
TIBCO EMS	Message Queue	Yes	tibco.com
TIBCO Hawk	SOA Governance	No	tibco.com
UltraESB by AdroitLogic	ESB	Yes	adroitlogic.org
Vigience Overcast	Integration Platform/PaaS	No	vigience.com
WebMethods by SoftwareAG	Integration Suite	No	softwareag.com
WebOTX ESB by NEC	ESB	Yes	nec.com
WSO2 Carbon	Integration Platform/PaaS	Yes	wso2.com
WSO2 ESB	ESB	Yes	wso2.com
WSO2 Governance Registry	SOA Governance	Yes	wso2.com
ZeroMQ	Message Queue	Yes	zeromq.org

diving deeper

INTO FEATURED ENTERPRISE INTEGRATION SOLUTIONS

Looking for more information on individual Enterprise Integration solutions providers?
Nine of our partners have shared additional details about their offerings, and we've summarized this data below.

If you'd like to share data about these or other related solutions, please email us at research@dzone.com.

3Scale API Management Platform

BY 3SCALE

STRENGTHS

- Access control panel to audit API usage
- Several authentication patterns and credentials to choose from
- API traffic controls

COMPONENTS

- API Gateway
- Developer Content Portal

FREE TRIAL Free Tier Available

WEBSITE 3scale.com

Anypoint Platform

BY MULESOFT

STRENGTHS

- Mule ESB built into the platform
- Leverages open standards for developer tools
- Built-in security and uptime tools

COMPONENTS

- API Gateway
- API Publisher
- API Store
- Developer Content Portal

FREE TRIAL 30 Days

WEBSITE mulesoft.com

Axway API Gateway

BY AXWAY

STRENGTHS

- Centralized platform to audit API usage
- Manage end-to-end API lifecycles and SLAs
- Self-service onboarding

COMPONENTS

- API Gateway
- API Publisher
- Developer Content Portal

FREE TRIAL Available By Request

WEBSITE axway.com

CA API Management Tool

BY CA TECHNOLOGIES

STRENGTHS

- Integrate and expose legacy systems and applications as APIs
- Onboard, enable and manage developers to create apps

COMPONENTS

- API Gateway
- API Publisher
- Developer Content Portal

FREE TRIAL Available By Request

WEBSITE ca.com

Carbon

BY WSO2

STRENGTHS

- Apache Zookeeper-based coordination support
- OSGi-based extensibility
- Built-in security, messaging, and throttling quality controls

COMPONENTS

- Application Server
- Artifact Repository
- Version Control Repository

FREE TRIAL Open Source

WEBSITE wso2.com/products/carbon

JBoss Fuse

BY RED HAT

STRENGTHS

- Quickly create and connect APIs
- Deploy or update services while ESB is running
- Visual debugging, testing, and deployment

COMPONENTS

- Apache Camel
- ActiveMQ
- Fabric8

FREE TRIAL Open Source

WEBSITE jboss.org

SmartBear SoapUI NG Pro

BY SMARTBEAR SOFTWARE

STRENGTHS

- Tests REST, SOAP, and other popular API and IoT protocols
- Run ad-hoc tests without maintaining temporary API client code
- Allows analysis of functional test coverage

COMPONENTS

- Functional Testing
- Automated Testing
- Test History

FREE TRIAL 14 Days

WEBSITE smartbear.com

Spring Integration

BY PIVOTAL

STRENGTHS

- Uses the Spring programming model to support integration patterns
- Supports remoting, messaging, and scheduling
- Java DSL available

COMPONENTS

- Implementation of Enterprise Integration Patterns
- Integration With External Systems
- JMX Support

FREE TRIAL Open Source

WEBSITE spring.io

WebMethods Integration Platform

BY SOFTWAREAG

STRENGTHS

- Project management tools included
- Managed file transfer within and across the enterprise
- End-to-end SDLC management

COMPONENTS

- ESB
- Application Server

FREE TRIAL Demo Available By Request

WEBSITE softwareag.com

glossary

API MANAGEMENT PLATFORM Middleware used to oversee the process of publishing, promoting, and configuring APIs in a secure, scalable environment; platforms usually include tools for automation, documentation, versioning, and monitoring.

APPLICATION PROGRAMMING INTERFACE (API) A software interface that allows users to configure and interact with other programs, usually by calling from a list of functions.

BUSINESS PROCESS MANAGEMENT (BPM) A workflow management strategy used to monitor business performance indicators such as revenue, ROI, overhead, and operational costs.

DOMAIN-DRIVEN DESIGN (DDD) A software design philosophy that bases the core logic and architecture of software systems on the model of the domain (e.g. banking, health care).

ENTERPRISE APPLICATION INTEGRATION (EAI) A label for the tools, methods, and services used to integrate software applications and hardware systems across an enterprise.

ENTERPRISE INTEGRATION (EI) A field that focuses on interoperable communication between systems and services in an enterprise architecture; it includes topics such as electronic data interchange, integration patterns, web services, governance, and distributed computing.

ENTERPRISE INTEGRATION PATTERNS (EIP) A growing series of reusable architectural designs for software integration. Frameworks such as Apache Camel and Spring Integration are designed around these patterns, which are largely outlined on EnterpriseIntegrationPatterns.com.

ENTERPRISE JAVABEANS (EJB) A server-side component architecture for modular construction of distributed enterprise applications; one of several APIs for Java Enterprise Edition.

ENTERPRISE SERVICE BUS (ESB) A utility that combines a messaging system with middleware to provide comprehensive communication services for software applications.

EVENT-DRIVEN ARCHITECTURE (EDA) A software architecture pattern that orchestrates behavior around the production, detection, and consumption of events.

EXTRACT TRANSFORM AND LOAD (ETL) A process for integrating large data batches through tools that copy data from various sources, make necessary translations, and send it to the final target.

HYPERMEDIA AS THE ENGINE OF APPLICATION STATE (HATEOAS) A principle of REST application architecture where clients interact with a network application entirely through hypermedia provided by application servers.

INTEGRATION PLATFORM-AS-A-SERVICE (IPAAS) A set of cloud-based software tools that govern the interactions between cloud and on-premises applications, processes, services, and data.

INTERFACE DEFINITION LANGUAGE (IDL) A specification language used to describe a software component's interface in a language-agnostic way, enabling communication between software components that are written in different programming languages.

JAVA MANAGEMENT EXTENSIONS (JMX) A Java technology that provides lightweight management extensions to Java-based applications and interfaces.

JAVA MESSAGE SERVICE (JMS) An API that functions as message-oriented middleware designed for the exchange of asynchronous messages between different Java-based clients.

JAVASCRIPT OBJECT NOTATION (JSON) An open standard data exchange format based on a JavaScript syntax subset that is text-based and lightweight.

MESSAGE BROKER A centralized messaging program that translates and routes message. This is the basis of the hub and spoke messaging topology.

MESSAGE-DRIVEN PROCESSING A computer model where clients send a service request to a program that acts as a request broker for handling messages from many clients and servers.

MESSAGE EXCHANGE PATTERN (MEP) The type of messages required by a communication protocol; the two major MEPs are request-response (HTTP) and one-way (UDP).

MESSAGE GATEWAY An application component that contains messaging-specific code and separates it from the rest of the application.

MESSAGING-ORIENTED MIDDLEWARE (MOM) A layer of software or hardware that sends and receives messages between distributed systems.

MESSAGE QUEUE A software component used for communication between processes/threads that harnesses asynchronous communication protocols.

MICROSERVICES ARCHITECTURE A system or application consisting of small, lightweight services that each perform a single function according to your domain's bounded contexts. The services are independently deployable and loosely coupled.

MIDDLEWARE A software layer between the application and operating system that provides uniform, high-level interfaces to manage services between distributed systems; this includes integration middleware, which refers to middleware used specifically for integration.

OAUTH A common open standard for authorization.

OPEN SOURCE GATEWAY INTERFACE (OSGI) A Java framework for developing and deploying modular programs and libraries.

REMOTE PROCEDURE CALL (RPC) An inter-process communication that causes a subroutine or procedure in another address space to execute without needing to write any explicit code for that interaction.

REPRESENTATIONAL STATE TRANSFER (REST) A distributed, stateless architecture that uses web protocols and involves client/server interactions built around a transfer of resources.

RESTFUL API An API that is said to meet the principles of REST.

SECURITY ASSERTION MARKUP LANGUAGE (SAML) An XML-based language protocol for handling authentication and authorization in a network or for web development.

SERVICE COMPONENT ARCHITECTURE (SCA) A group of specifications intended for the development of applications based on service-oriented architecture.

SIMPLE OBJECT ACCESS PROTOCOL (SOAP) A protocol for implementing web services that feature guidelines for communicating between web programs.

SERVICE-ORIENTED ARCHITECTURE (SOA) An architecture style that uses discrete software services (each with *one* clearly defined business task) with well-defined, loosely-coupled interfaces that are orchestrated to work as a complete system by sharing functionality.

WEB SERVICE A function that can be accessed over the web in a standardized way using APIs that are accessed via HTTP and executed on a remote system.

WEB SERVICES DESCRIPTION LANGUAGE (WSDL) An XML-based language that describes the functionality of a web service, and is necessary for communication with distributed systems.

WEB SERVICES DESCRIPTION LANGUAGE (WSDL) An XML-based language that describes the functionality of a web service, and is necessary for communication with distributed systems.