DZone
RESEARCH

DZONE RESEARCH PRESENTS

2014 GUIDE TO ENTERPRISE INTEGRATION

BROUGHT TO YOU IN PARTNERSHIP WITH

3SCALE ca technologies MuleSoft

redhat WSO2

WELCOME

Dear Reader,

This year has been a great year for all of us at DZone with the success of five research guides and enormously positive feedback from our ever-enthusiastic tech community. Our research guides have now been downloaded over 150,000 times; we see this figure as positive evidence that the information and data we've collected is being produced for an audience that truly wants to know more about industry topics. These guides are created for readers like you, and I thank you sincerely for all your responses and support throughout this journey.

Enterprise integration is one of the most prominent topics on DZone, and it commands the largest audience for us outside of Java. With several cheatsheets (Refcardz) already published on various related subtopics, it's only fitting for us to publish this research guide at a time when our audience on this topic has never been bigger. Enterprise integration poses huge challenges for developers, and with so many different technologies to choose from, finding the right solution can be difficult.

DZone's 2014 Guide to Enterprise Integration introduces microservices and service-oriented architecture, explores case studies in this field, and provides a comprehensive solution directory for comparing the most prominent message queues, integration frameworks and suites, ESBs, iPaaS solutions, and API management platforms. We've covered both the truly legacy approaches to enterprise architecture, as well as growing strategies that the jury is still out on. We've gathered in this guide a broad range of solutions for integration experts of every background.

We are already looking forward to an exciting 2015 and the new topics that will come with it. We would love to hear your feedback on this report, thoughts on future topics, and ways you think we can make our research guides better suited for your needs.

Thanks again for your time and for being a part of our great community.

Jayashree Gopal
Director of Research
research@dzone.com

TABLE OF CONTENTS

CREDITS

DZONE RESEARCH

JAYASHREE GOPAL
DIRECTOR OF RESEARCH

MITCH PRONSCHINSKE
SENIOR RESEARCH ANALYST

BENJAMIN BALL
RESEARCH ANALYST

MATT WERNER
MARKET RESEARCHER

JOHN ESPOSITO
REFCARDZ COORDINATOR

ALEC NOLLER
SENIOR CONTENT CURATOR

JOHN WALTER
CONTENT CURATOR

RYAN SPAIN
CONTENT CURATOR

DZONE CORPORATE

RICK ROSS
CEO

MATT SCHMIDT
CTO, PRESIDENT

ALEX CRAFTS
VP OF SALES

BRANDON NOKES
VP OF SALES, DEMAND GENERATION

HERNÂNI CERQUEIRA
LEAD SOFTWARE ENGINEER

KELLET ATKINSON
DIRECTOR OF MARKETING

CHELSEA BOSWORTH
MARKETING ASSOCIATE

ASHLEY SLATE
DIRECTOR OF DESIGN

FARHIN DOROTHI
GRAPHIC DESIGNER

CHRIS SMITH
PRODUCTION ADVISOR

Special thanks *to our topic experts Jeremy Likness, David Dossot, James Strachan, Pieter Hintjens, Asankha C. Perera, Markus Eisele, John Page, Ben Mann, Daniel Bryant, and our trusted DZone Most Valuable Bloggers for all their help and feedback in making this report a great success.*

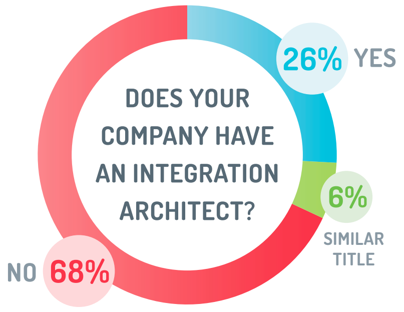
SUMMARY & KEY TAKEAWAYS

Application integration is inherent to the development of a software system of any significant size. It is a complicated process, and while there are many useful solutions, the success of the architecture is often not known for months, or even years. We created this guide to help developers navigate this trial-and-error process. Our research shows that it's the engineering staff, rather than non-technical managers, selecting and implementing enterprise integration products and strategies. No one makes this decision for you, so it's more important than ever to know which solutions fit your needs. DZone's 2014 Guide to Enterprise Integration is a unique resource for developers and architects to learn how industry experts are handling integration in legacy enterprise systems, modern systems, and massive web-scale systems. It contains resources that will help you succeed with modern architectural patterns and application integration. These resources include:

- A side-by-side feature comparison of the most prominent integration frameworks, integration suites, ESBs, message queues, and API management platforms.
- Comprehensive data sourced from 500+ IT professionals on integration solutions, strategies, and experiences.
- A model for assessing four levels of REST API maturity and implementation.
- A guide to decomposition patterns for breaking monolithic applications into smaller services.
- A forecast of how building a large project with multiple integrations might look in the future.

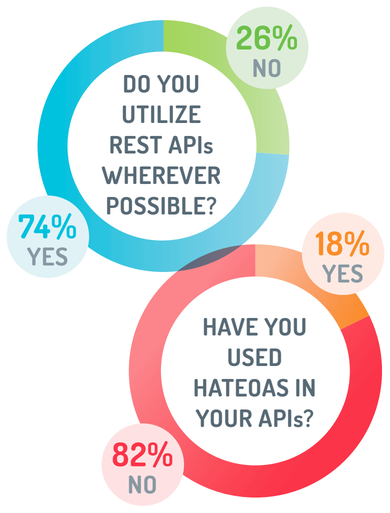
KEY TAKEAWAYS

COMPANIES HAVE DEDICATED INTEGRATION ARCHITECTS While some developers may think of integration as a tedious chore, our research indicates otherwise: it is a critical area deserving of focus. A large portion of companies (32%) have a dedicated Integration Architect to work entirely on integrating systems. This indicates that many companies require constant specialized attention for their integration needs, and they address these by adding team members whose primary duty is to oversee integration architecture.



MOST DEVELOPERS ARE BUILDING THEIR OWN INTEGRATIONS While a growing number (but not a majority) of companies have a dedicated integration expert, most companies (68%) don't yet have an employee dedicated to integration. Companies have the option of using connectors and integrations built by a vendor, but these options are often extremely expensive, or not particularly customized to the systems they need to integrate. We found that 78% of respondents are building their own integrations, compared to only 22% that utilize integrations built by vendors. This could be due to the need for highly customized integrations, or the high cost of vendor tools and connectors [1].

MICROSERVICES ARCHITECTURE IS GAINING TRACTION A significant shift in IT over the past decade has been moving towards service-oriented architectures as an alternative to monolithic architecture, a trend that's currently reflected in the rising popularity of microservice architectures. Microservices have been popularized by the success of web giants like Amazon and Netflix, who utilized microservice architecture for massive web scalability—and were more than willing to talk about it [2]. What we discover is that 39% of organizations are utilizing microservice architecture in at least one of their systems. So, while many of our respondents are creating microservices, it's not yet acquired a majority, and still has room to grow in adoption rates.



ORGANIZATIONS ARE UTILIZING REST APIS WHEREVER THEY CAN The implementation of REST principles with APIs has become extremely widespread. We found that 74% of companies were building and utilizing REST APIs whenever and wherever they could. However, we also asked these same respondents whether or not they were utilizing HATEOAS (Hypermedia as the Engine of Application State), which is sometimes considered necessary for a complete REST API [3]. Only 18% of respondents have utilized HATEOAS in their APIs, which illustrates the disagreement of developers on the importance of certain REST API attributes.

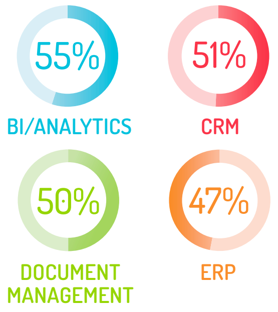
[1] http://www.infoworld.com/article/2678499/operating-systems/ibm-to-ship-db2-integration-software.html
[2] http://www.slideshare.net/stonse/microservices-at-netflix
[3] http://restcookbook.com/Basics/hateoas/

KEY RESEARCH FINDINGS

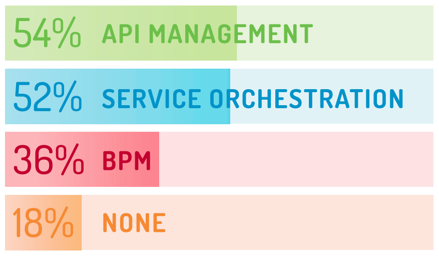
More than 500 IT professionals responded to **DZone's 2014 Enterprise Integration Survey**. Here are the demographics for this survey:

- **Developers** (45%) and development **team leads** (30%) were the most common roles.
- 69% of respondents come from **large organizations** (100 or more employees) and 31% come from **small organizations** (under 100 employees).
- The majority of respondents are headquartered in **Europe** (48%) or the **US** (28%).
- Over half of the respondents (70%) have over **10 years of experience** as IT professionals.
- A large majority of respondents' organizations use **Java** (94%). **JavaScript** is the next highest (47%).

MOST COMMON SYSTEM INTEGRATIONS



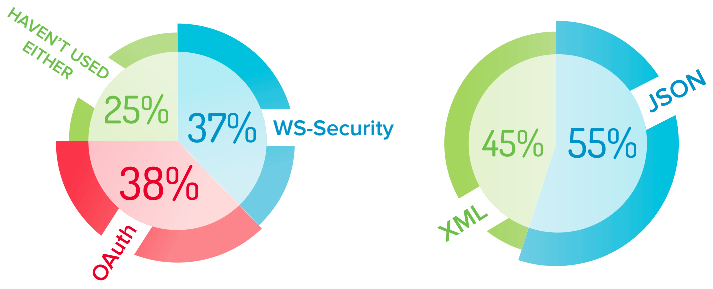
ADDITIONAL TOOLS NEEDED IN AN EI SOLUTION



ANALYTICS SYSTEMS ARE THE MOST COMMON INTEGRATION BI/Analytics is the most common type of system integrated in respondents' architectures (55%), followed closely by CRM (51%), document management (50%), and ERP (47%). Analytics and Big Data systems have clearly come front and center in the integration space, despite the varieties and speeds of data translation and integration required. Data integration is an often underappreciated aspect of building and utilizing analytics systems.

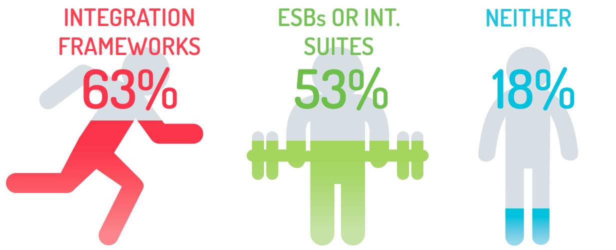
LESS MANUAL API INTEGRATION IS BECOMING PREFERABLE 62% of respondents say they often have difficulty manually integrating APIs. Another data point that supports this sentiment is the prominence of API management tools as the most common additional tool that respondents need for their enterprise integration solutions (54%). In contrast, only 10% of all respondents actually use an API management platform. Service orchestration (52%) is the other major tool that respondents want in an EI solution. 18% don't need any additional tools.

WHICH PROTOCOLS/LANGUAGES ARE YOU MORE COMFORTABLE WITH?

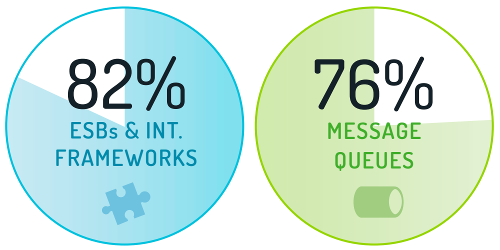


JSON FAVORED SLIGHTLY FOR APIs There are a few significant dichotomies in the IT community centered around API development. Some aspects of API development have competing protocols and languages, such as JSON vs. XML and WS-Security vs. OAuth. When asked which technologies they are more comfortable with, developers split evenly between the WS-Security protocol (37%) and OAuth (38%). For JSON vs. XML, JSON had a slight edge (55%) over XML (45%).

ESBs VS LIGHTWEIGHT INTEGRATION FRAMEWORKS



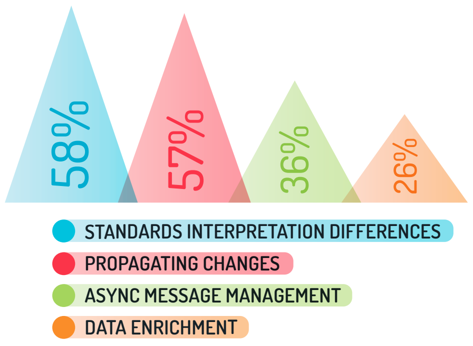
EAI TOOLS VS MQs



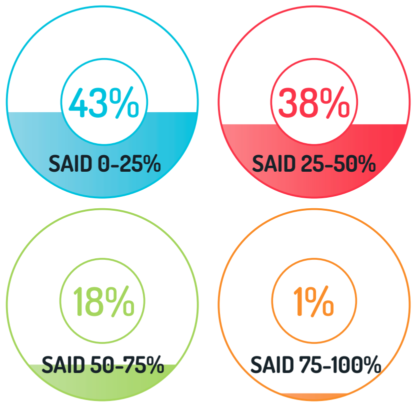
LIGHTWEIGHT EIP FRAMEWORKS ARE THE NORM FOR MOST SYSTEMS Out of all integration frameworks (EIP), ESBs, and integration suites, Spring Integration is the most popular (42%) and Apache Camel is a close second (38%). The three most popular ESBs are Mule ESB (16%), WebSphere ESB (15%), and Oracle ESB (13%). Overall, 63% of respondents use an integration framework (e.g. Spring Integration, Camel) and 53% use an ESB or Integration Suite (e.g. Mule ESB, Biztalk), while 18% say they use neither. *Note that 69% of respondents are from large organizations, where bigger integration scenarios would be more common.*

COMPREHENSIVE MESSAGE QUEUES LEAD IN POPULARITY Although there has been a recent surge in popularity for bare-bones, low-latency message queues like ZeroMQ (4%), full-featured messaging systems like ActiveMQ (46%) and RabbitMQ (20%) are the clear leaders in message queue popularity. HornetQ (9%), IBM WebSphere MQ (8%), and Microsoft MQ (8%) are also fairly popular among the rest of the queues, and 24% don't use any message queues. Compare this with integration frameworks and ESBs, and we find that 82% use integration frameworks and ESBs, while 76% use MQs.

WHAT ARE YOUR ORGANIZATION'S BIGGEST INTEGRATION DIFFICULTIES?



INTEGRATION COSTS AS A % OF TOTAL IT COSTS



CLOUD INTEGRATION IS NOT A COMMON ISSUE Only 37% of organizations have to manage on-premise-to-cloud integrations, and 15% have to manage cloud-to-cloud integrations. While integration with newly built or purchased cloud services is often cited as a major technical undertaking, only 9% cite cloud-to-on-premise integration as a major integration issue, and 5% cited cloud-to-cloud integration. The two biggest integration difficulties for organizations are handling different interpretations of standards between systems (58%) and propagating changes to integrated systems (57%).

INTEGRATION COSTS NOT HUGE, BUT SIGNIFICANT Regarding the number of integrations in an organization, 74% of respondents say they integrate 10 systems or fewer in their largest applications. Less than 1% integrated only one system. As for the cost of those integrations, 43% say integration accounts for less than 25% of their total IT costs. A smaller, but still concerning segment (38%) says that integration accounts for between 25% and 50% of total IT costs. So, about one third of businesses are spending significant portions of their IT budgets on integration.

INTEGRATION ARCHITECTURE: HOW WE GOT HERE

BY ASANKHA PERERA & MITCH PRONSCHINSKE

Name some of the most interesting software systems you know and I'll bet that they've had to deal with integration.

Particularly in enterprises (even the small ones), multiple applications, systems, and components need to work together and give employees, partners, and customers the performance and feel of a single application. However, many architects design integrations the way they would design a single application, unaware that the nature of integration significantly alters the way applications must be developed. Let's briefly look at when enterprise integration first became a topic of intense focus in the modern IT industry.

EAI: THE ORIGINS

Back when enterprises were built on large mainframes and minicomputer-based application systems, Enterprise Application Integration (EAI) evolved as a mechanism to allow the exchange of information between systems without having each application create information silos within organizations. Methods for integration included file transfers, shared databases, and application functionality access via remote procedures. These methods are still used today wherever they are appropriate. A good example is the Extract Transform and Load (ETL) method, which is a file transfer integration strategy often favored for large, batch data analysis.

Many systems from the mainframe era were batch-oriented and generally based on files and databases, which lead to EAI technologies evolving around the ETL method. Although ETL mechanisms allowed the integration of applications implemented in different technologies or residing across different systems, it did not provide a solution for real-time integration needs. Shared databases and remote procedure invocations also didn't perform well for real-time integration because of one persistent obstacle: the unpredictable interruptions of computer networks.

MOM: A PARADIGM SHIFT IN EI

In the 1990s, Point-to-Point (P2P) integration using Messaging-Oriented Middleware (MOM) emerged as one of the most versatile technologies to support both real-time and non real-time needs. Messaging systems use channels (queues) that other applications can read and recognize. Each application using the queue has to have a sender/producer program or a receiver/consumer program. These endpoints contain some form of message storage so that, even if the network connection is down, the program will know that a message hasn't gone through and will queue it. This type of communication is more efficient because it can be asynchronous, meaning that the consumer can handle messages received in any order, and it can wait to handle messages until the appropriate time. The communication is also non-blocking, so if one message is taking a long time, the channel and the consumer can still handle other messages while it's waiting. Message queues can even support systems with one producer and many consumers, such as the publish-subscribe paradigm. These messaging paradigms defined the message bus topology, which became widely used with the advent of Java Messaging Service (JMS) APIs.

Another popular topology for message queues is the broker model (hub and spoke topology), which uses a central hub that all communications pass through for inspection, management, translation, and routing. This allows for better governance of systems, but the downside is that forcing each message to pass through the hub introduces latency, and can be a possible single point of failure.

Message queues are widely used today. Hardware and software design enhancements have significantly improved the performance of these systems over the years. The market has also made significant advancements toward open protocols for messaging, such as the Advanced Message Queuing Protocol (AMQP), which started the trend in 2004. These developments have brought down the cost of MOM and pushed distributed systems into wider use.

THE ENTERPRISE SERVICE BUS

Inside enterprises, where the central governance of a broker topology is traditionally more appealing, the Enterprise Service Bus (ESB) emerged as a technology that took advantage of the strengths of both broker and message bus topologies. It's not a perfect analogy, but you could say that message queues are like the popular code editors (e.g. SublimeText), while ESBs are like IDEs (e.g. Eclipse). Queues can do a lot and still be lightweight, but ESBs try to cover more scenarios and provide more tools out of box.

ESB solutions also emerged at a time when web services, especially the SOAP protocol, were becoming widespread. ESBs quickly moved to offer built-in support for web services along with other protocols and message formats. Although some of the early ESB solutions were implemented on top of older MOM implementations, or required Java EE containers for the run-time, the space quickly evolved to support higher performance and lightweight deployment patterns. Today, there are a few vendors with more than one ESB offering, depending on the needs of customers.

A fully-formed ESB product or open source distribution isn't the only option either. Minimal frameworks that harness all of the well-known enterprise integration patterns (e.g. Apache Camel, Spring Integration) can be paired with a message queue and other open source frameworks to provide smart routing, transformation, orchestration, and other features of ESBs.

SERVICE-ORIENTED ARCHITECTURE

Loose-coupling and modular applications were nothing new in the 2000s, but a stronger emphasis on these strategies started when messaging integration paradigms made it easier to build modular pieces of software that worked together to form complete systems and applications. This is when the idea of Service-Oriented Architecture (SOA) came into mainstream use. The goal of SOA is to make software architecture more flexible to change. It is an architecture style that uses discrete software services, each having one clearly defined business task, with well-defined, loosely-coupled interfaces that are orchestrated to work as a complete system by sharing functionality.

WILL MICROSERVICES BECOME THE NEW STANDARD OF SCALABLE ARCHITECTURE?

Developers generally try to build loosely-coupled software components in their applications, so the basic concept of SOA is intuitive to many developers. In the late 2000's, however, the perception of SOA became tied to the ESB architecture, which many argue is not the best way to build SOA. Though I've given the basic definition of SOA above, it is trickier to define in practice [1]. ESBs and SOA received some backlash from the development community because enterprise-scale ESBs often seem to have too many unnecessary features or too strict a tie to a vendor's product suite. Developers using ESBs also tend to use them as a place to hide complexity, instead of dealing with it more effectively.

MICROSERVICES

Major web companies like Netflix and SoundCloud have recently put the spotlight back on SOA after sharing their methods for integration and scalability, which center around an architecture composed of microservices. Some developers are saying that microservices are what SOA is supposed to be, but regardless of whether microservices fall under that broad definition, their definition is much clearer.

The community that has formed around microservices favors an integration approach that differs from the classic ESB strategy. Instead of a smart ESB mediator in the middle of the services-based architecture, microservices will connect to each other by having smart endpoints on each service and having dumb pipes—which are fast messaging channels with very little logic present in the transfer stage [2]. Microservices, like their name implies, are also strict about the size of each service. Each microservice tends to make up one business feature of the system. A microservice is a unit of software that can be independently removed, plugged in, and upgraded. Unlike software libraries, they are out-of-process components that communicate via web services, remote procedure calls, or other similar methods. This strict componentization makes it easy to have cross-functional developers or teams dedicated to each service, and that's exactly what Netflix does.

API ARCHITECTURES

Integrating systems with websites and applications made by other people was another challenge in the history of integration that is handled today through web APIs. The architectural patterns of web APIs have made it easier for developers to connect unfamiliar software to their own projects, whether they're from another department in a large enterprise, or from an entirely different company. There are certainly strong opinions about the mechanisms being used in these APIs (SOAP vs REST, XML vs JSON, or HATEOAS vs pragmatic REST), but each option has its pros and cons. An organization needs to find the right fit for their needs. The current focus for many toolmakers is building solutions that can manage an API-dense architecture and marketplace.

IPAAS

As cloud services proliferated several years ago, integration with those services and on-premise systems was another major challenge for the industry. While there are many possible solutions to this obstacle—many involving a new way of thinking given the distributed, elastic nature of cloud infrastructure—an obvious solution was to build integration utilities into cloud development platforms. The iPaaS (integration Platform-as-a-Service) has emerged as a viable option for providing integration utilities similar to those found in ESBs, but with various cloud integration scenarios in mind.

THE EI OUTLOOK

The core messaging paradigm of modern integration is likely here to stay for many more years as we enter a new era of connected devices through the Internet of Things. ESBs are still a great solution for many tech firms given the variety of options from lightweight to large legacy-integration scale. SOA may fade into obscurity as a term, but its patterns and lessons will live on in the fabric of modern development, especially in emerging microservices architecture. Will microservices become the new standard of scalable architecture? Loose-coupling is already a standard, but microservices will take some time to become mainstream, and if they do, there will always be enterprises that misuse its patterns and give it a bad name.

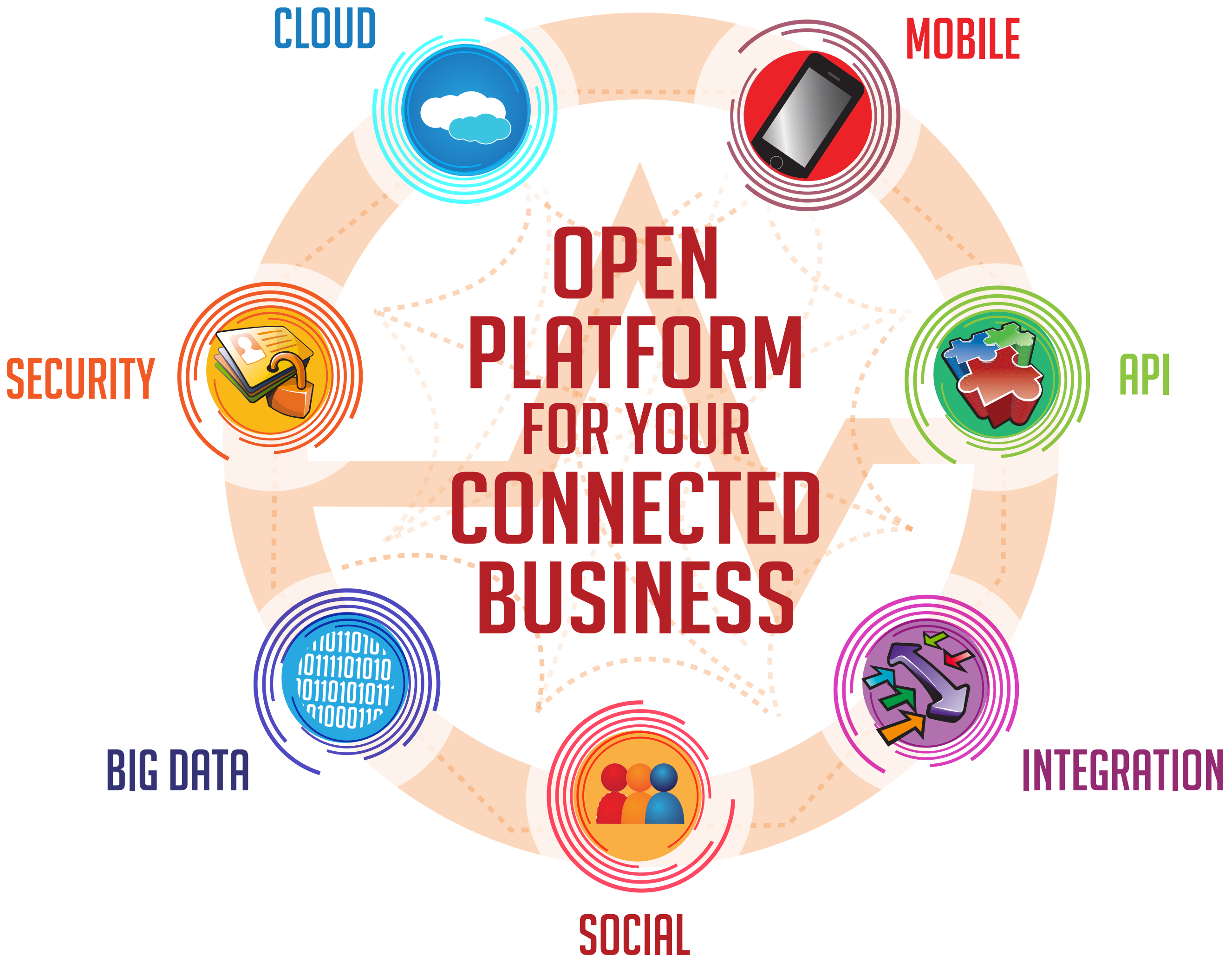
References:
[1]: http://martinfowler.com/bliki/ServiceOrientedAmbiguity.html
[2]: http://martinfowler.com/articles/microservices.html

WRITTEN BY ASANKHA PERERA

Asankha Perera is the founder and CTO of AdroitLogic—the creators of the free and open source Enterprise Service Bus UltraESB. He is a member of the Apache Software Foundation and the VP of the Apache HttpComponents project.

WRITTEN BY MITCH PRONSCHINSKE

Mitch Pronschinske is the Head Analyst for DZone's Research division. He has been writing, curating, and editing content for an audience of IT professionals for over five years. In that time, he has understood the complexity that software producers deal with on a daily basis and strives to make their world easier to understand and digest.


CLOUD
MOBILE
OPEN PLATFORM FOR YOUR CONNECTED BUSINESS
SECURITY
API
BIG DATA
INTEGRATION
SOCIAL

PICK THE RIGHT APPLICATION SERVER TO FULLY BENEFIT FROM JAVA EE

Java Enterprise Edition (Java EE), the widely adopted platform for developing and running enterprise applications and web services, has evolved significantly over the years. The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications. To effectively extend the benefits of using Java EE, an enterprise must have an application that's multi-tenant and lightweight. Most enterprises today are looking to migrate from traditional application servers to those that support Java EE specifications.

WSO2 Application Server (WSO2 AS)—an Apache Tomcat-based web application hosting platform—is one that meets these enterprise requirements; it currently supports Java EE, and the next major version is set to look at supporting a complete set of Java EE WP technology specifications. Switching to WSO2 AS enables Java EE developers to leverage a number of platform-level features supported by the WSO2 middleware platform, such as multi-tenancy, SaaS development, embedded API management, and real-time analytics.

WSO2 AS inherits an in-built user store from the WSO2 Carbon framework, and it's possible to plug any existing user store such as LDAP, Active Directory, or custom JDBC easily through configuration options. Hence, when migrating applications, it is possible to reuse existing custom and standard user stores with WSO2 AS through configuration options.

The following table describes supported Java EE WP technologies on WSO2 AS.

TECHNOLOGY	AS VERSION	INTEGRATED FRAMEWORK
Servlet 3.0	5.0.0	Apache Tomcat 7 (*)
JSP 2.2 , EL 2.2 & JSTL 1.2	5.1.0	Apache Tomcat 7 / Jasper (*)
JSF	5.2.0	Apache MyFaces
JPA 2.0	5.2.0	Apache OpenJPA
JTA 1.1	5.2.0	Atomikos JTA
CDI 1.0	Future	Apache OpenWebBeans
Baan Validation 1.0	Future	Apache BVal
EJB - Lite	Future	Apache OpenEJB (**)

(*) WSO2 AS plans to use Apache Tomcat 8 when it matures
(**) Some of the Apache TomEE components will be used for WSO2 AS - Apache OpenEJB integration

Take a look at the WSO2 whitepaper, Evaluating Java EE Application Migration and Java EE Service Migration to WSO2 Application Server for more details.

WRITTEN BY **SAGARA GUNATHUNG**, SENIOR TECHNICAL LEAD, WSO2

ESB

WSO2 ESB By WSO2

WSO2's ESB operates at high performance and has a wide range of integration capabilities and connectors, including support for EIPs.

DESCRIPTION

WSO2 ESB is a 100% open source, lightweight, high performance ESB which supports various enterprise integration scenarios. It's 100% compliant with all EIPs and offers built-in support for cloud (multi-tenancy, etc.). WSO2 ESB offers adaptors and connectors from various web APIs such as Salesforce to proprietary systems such as SAP. It's a part of the comprehensive WSO2 middleware platform.

CUSTOMERS

- eBay
- T-Systems
- Trimble
- AAA
- Cisco

DRAG-N-DROP

- ✓ ROUTE CREATION
- ✓ WEB SERVICES
- ✓ DATA SERVICES

RUNTIME MODE
Standalone

OPEN SOURCE
Yes

DEPENDENCIES
Java

FEATURES:

- ✓ BPEL
- ✓ SERVICE ORCHESTRATION
- ✓ FAILOVER HANDLING
- ✓ FEDERATION
- ✓ WEB DEPLOY CONSOLE
- ✓ SERVICE MOCKING
- ✓ GUI FOR MONITORING SERVICES
- ✓ INTEGRATED ARTIFACT REPOSITORY
- ✓ AUTOMATIC RECOVERY OF FLOWS
- ✓ DISTRIBUTED TRANSACTION SUPPORT
- ✓ CAN BUILD OWN ADAPTERS
- ✓ EXPOSE METRICS THROUGH JMX

BLOG wso2.com/library/ | **TWITTER** @wso2 | **WEBSITE** wso2.com

Integrating Microservices into the Enterprise

by Daniel Bryant

The microservice hype is everywhere, and although the industry can't seem to agree on an exact definition, we are repeatedly being told that moving away from a monolithic application to a Service-Oriented Architecture (SOA) consisting of small services is the correct way to build and evolve software systems.

Companies of all sizes are beginning to evaluate this architectural style despite concerns about the actual benefits and the higher potential for complexity. Whether or not you agree with the proposed benefits of this latest incarnation of SOA, the desire to investigate microservice implementations is rapidly growing in enterprise organizations. This article will serve as a useful primer for that research.

INTERFACES–GOOD CONTRACTS MAKE GOOD NEIGHBORS

Whether you are starting a greenfield microservice project or are tasked with deconstructing an existing monolith into services, the first task is to define the boundaries and corresponding APIs to your new components. A greenfield project often has more flexibility, and the initial design stage can define Domain Driven Design (DDD) inspired bounded contexts with explicit responsibilities and contracts between service provider and consumer (for example, using Consumer Driven Contracts). However, a typical brownfield project must look to create "seams" within the existing applications and implement new (or extracted) services that integrate with the seam interface. The goal is for each service to have high cohesion and loose coupling; the design of the service interface is where the seeds for these principles are sowed.

COMMUNICATION–TO SYNCHRONIZE, OR NOT TO SYNCHRONIZE?

In addition to the definition of service interfaces, another primary design decision is whether to implement synchronous or asynchronous communication. Asynchronous communication is often favored, as this can lead to more loosely-coupled, and hence less brittle, services. However, my consulting firm's experience has shown that each use case should be examined on its own merits, and requirements around consistency, availability, and responsiveness (e.g. latency) should be considered carefully. In practice, we find that many companies will need to offer both synchronous and asynchronous communication in their services.

The current de facto synchronous communication mechanism in microservice architecture is provided by creating a REST-like API that exposes resources on a service, typically using JSON over HTTP. Interface Definition Language (IDL) implementations utilizing binary protocols, such as Thrift or Avro, can also be used if RPC-like communication or increased performance is required. However, caution should also be taken against utilizing too many different protocols within a single system, as this can lead to integration problems and maintenance issues.

An implementation of an IDL interface typically comes with its own implicit (and often versioned) contract, which can make integration and testing against endpoints easier. Explicit tooling must be implemented for contract-based development of REST-like interfaces. The PACT framework is a good example that provides both service virtualization for consumers and contract validation against an API for providers.

Once services and API endpoints are created, they can be deployed and managed in-house. However, many enterprises are instead choosing to offload API management and are instead using API Gateway services, such as Apigee, 3Scale, or Mashery. Regardless of deployment methodology, the problems of fault tolerance and resilience must be handled with synchronous communication. Design patterns such as retries, timeouts, circuit-breakers, and bulkheads must be implemented in a system of any size, especially if load will be high or the deployment fabric is volatile (e.g. a cloud environment). One excellent framework that provides an implementation of these patterns is Netflix's Hystrix, which has also been combined with Dropwizard in the "Tenacity" project and Spring Boot in the "Spring Cloud" project.

Lightweight message queue (MQ) technology is often the favored method for implementing asynchronous communication. RabbitMQ and ActiveMQ are the two most popular choices [1]. Frequently these MQs are combined with reactive frameworks, which can lead to the implementation of event-driven architectures, another emerging hot topic within our industry. Enterprises seeking high performance messaging can look at additional MQ technologies such as ZeroMQ, Kafka, or Redis. If their requirements include high throughput/low latency processing of messages or events, then one of the current stream-based Big Data technologies may be more appropriate, such as Spring XD or Apache Storm. Interface contracts tend to be loose with asynchronous communication since messages or events are typically sent to a broker for relay onto any number of interested services. Postel's law of "be conservative in what you send, be liberal in what you accept" is often touted as a key principle for both provider and consumer services. If your development team is exploring asynchronous communication for the first time, then some care must be taken, because the programming model can be significantly different in comparison with synchronous blocking communication. Finally, although many MQ services provide fault tolerance, it is highly recommended that you confirm your requirements against the deployment configuration of your MQ broker.

MIDDLEWARE–WHAT ABOUT THE TRADITIONAL ENTERPRISE STALWARTS?

With the mention of REST-like endpoints and lightweight messaging, many enterprise middleware vendors are understandably becoming nervous. When my consulting firm discusses the microservice architecture with enterprise clients, many look at implementation diagrams and immediately ask whether the message broker sitting in between all of the services is a commercial Enterprise Service Bus (ESB)—such has been the success of the ESB marketing campaign. We typically answer that it could be, but we usually find that a lightweight MQ platform is more suitable because we believe the current trend in SOA communication is towards “dumb pipes and smart endpoints” [2]. In addition to removing potential vendor fees and lock-in, other benefits of using lightweight MQ technologies include easier deployment, management, and simplified testing. Although many heavyweight ESBs can perform some very clever routing, they are frequently deployed as a black box. Jim Webber once joked that ESB should stand for “Egregious Spaghetti Box,” because the operations performed within proprietary ESBs are not transparent, and are often complex. If requirements dictate the use of an ESB (for example, message splitting or policy-based routing), then open source lightweight ESB implementations such as WSO2 ESB, Mule ESB, and Fuse ESB should be among the first options you consider.

On a related topic, we have found that although many companies would like to split monolithic systems, the Enterprise Integration Patterns (EIPs) contained within the corresponding applications are often still valid. Accordingly, we believe the use of EIP frameworks such as Spring Integration and Apache Camel still have their place within a microservice architecture. These frameworks typically provide a large amount of “bang for your buck” in relation to the amount of code written (which may be an important factor when creating a microservice), and they nicely abstract over archetypal EIP solutions. These frameworks can also be introduced as an intermediate step when migrating to a microservice architecture. Refactoring existing code to utilize EIPs, such as “pipes and filters,” may allow components to be extracted to external services more easily at a later date.

DEPLOYMENT–“DONE” MEANS DEPLOYED TO PRODUCTION

Deploying microservices can be a challenge for new adopters because existing frameworks, platforms, and processes for releasing a monolith into production may not scale well with multiple services in the mix. Bob Martin has recently discussed the “component scalability scale” on his blog. Martin suggested that components/services deployment configuration can range from multiple microservices deployed across a cluster to a service created by dynamically linking components (e.g. via JARs) and deploying them into a single VM. This is a useful model, and below is a list of related technologies that can be explored depending on your use case:

- **Microservices deployed onto a cluster:** Spring Boot, Dropwizard, Ratpack services, or container-based services (e.g. Docker) deployed on Mesos with Marathon, Kubernetes, CoreOS Fleet, or a vendor-specific PaaS. Service discovery via Consul, Smartstack, or Curator ensemble.

- **Small number of servers, each running more than one microservice:** Spring Boot or Dropwizard services deployed as a WAR into containers or running as fat JARs containing an embedded container (e.g. Jetty, Tomcat). Service discovery via Consul, Smartstack, or Curator ensemble.

- **Single server, multiple microservices:** Spring Boot or Dropwizard services running on configurable ports deployed within multiple embedded containers (e.g. Jetty or Tomcat). Service discovery via Curator, local HAProxy, or properties file.

- **Services running as threads in a single VM:** Akka actors or Vert.x Verticles deployed as a JAR running on a single JVM. Service (actor) discovery implicit within Akka and Vert.x frameworks.

- **Dynamically linked components within a single service:** OSGi bundle deployed via Apache Felix or Apache Karaf. No need for service discovery, but correct bundling of components is vital.

The list above contains just a few examples. However you choose to deploy microservices, it is essential that a build pipeline include rigorous automated testing for functional requirements, fault-tolerance, security, and performance.

SUMMARY–APIS, LIGHTWEIGHT COMMS, AND CORRECT DEPLOYMENT

Regardless of whether you subscribe to the microservice hype, it would appear that this style of architecture is gaining traction within enterprise integration development. This article has attempted to provide a primer for understanding key concepts within this growing space. Ultimately, microservices are a deployment detail and if you practice good loose-coupling strategies, then changing deployment styles should not incur large development costs. Simon Brown, a well-known software architecture consultant, said it best: “If you can’t build a structured monolith, what makes you think microservices is the answer?” [3]. This question, although outside the scope of this article, is extremely pertinent and is something that we should all consider before embarking on a new microservice project. Microservices may provide a very useful architectural and deployment style, but they certainly are not a panacea to all of your current software development problems. However, if you do decide that the microservice approach is appropriate for your application, then I hope that this article is a useful springboard for your research and evaluation.

REFERENCES:

[1]: Key Research Findings, pg. 5

[2]: http://martinfowler.com/articles/microservices.html

[3]: http://www.codingthearchitecture.com/2014/10/01/modularity_and_testability.html

WRITTEN BY **DANIEL BRYANT**

Daniel Bryant is a Principal Consultant for OpenCredo, a software consultancy and delivery company dedicated to helping clients deliver better software faster. Currently, Daniel specializes in enabling agility, continuous integration, continuous delivery, and other DevOps methodologies. He is also a leader within the London Java Community, and a contributor to the Java Community Process (JCP).

SCALE IT
LIKE YOU MEAN IT.

The all-in-one API management platform that lets you deploy on-premise or in the cloud, no black box required. Learn why over 450 companies use 3scale.

- ✓ Access control and security
- ✓ API contracts and rate limits
- ✓ Analytics and reporting
- ✓ Developer portal and docs
- ✓ Billing and payments

Get started at
3scale.net/dzone

3SCALE

YOU ARE WHAT YOU API

5 STEPS TO A ROCK SOLID API PROGRAM

PROVIDE A VALUABLE SERVICE

APIs are becoming mainstream, but a "me too" API is a poor investment – an API must deliver value both for the provider and its potential audience. You need to get this right.

It can be hard to pinpoint one audience, and focus is often on the API project itself. Instead, the focus should be on of the effect the API can deliver. Ask yourself – what job should it do for the business? Is it in line with broader goals? What job does it do for its users?

TIE THE MODEL TO YOUR BUSINESS

Instead of asking, "What business model should I use for my API?" you should be asking, "What API can best support my business model?" Start with your business and extend to how an API can enable it.

MAKE IT SIMPLE, FLEXIBLE, AND EASY

Sounds obvious, but simplicity can be thwarted by things like complex use cases or legacy code. Make sure important use cases are easy to execute and the common workflows are supported. Good documentation is a must.

PUT MANAGEMENT IN PLACE

Your API is a window to your data assets and functionality. Powerful, but it creates dependencies between you and your users. Who has rights? How do they get access? How is it secured and versioned? How are changes detected and communicated? Are there restrictions, charges, or SLAs?

API Management Platforms provide the security, performance monitoring, and analytics tools needed to keep APIs operating effectively.

ENSURE DEVELOPER DELIGHT

The best APIs excel at support. They put portals and people in place to help developers succeed. Support needs to be a first class citizen in the planning process — not an after-thought.

START WITH YOUR BUSINESS AND EXTEND TO HOW AN API CAN ENABLE IT.

WRITTEN BY **STEVEN WILLMOTT**, CEO, 3SCALE

API MANAGEMENT

3scale API Management Platform By 3scale

3SCALE

API management delivered as-a-service via unique, high performing hybrid/cloud architecture. Delivers unmatched flexibility and scale.

DESCRIPTION

3scale provides a comprehensive API management platform that lets API providers easily package, distribute, manage, and monetize APIs through a SaaS infrastructure that is flexible, secure, and Web scalable. The platform enables the distribution of a company's data, content, or services to multiple devices or mobile/Web applications, as well as the ability to easily productize APIs.

FEATURES:

- ✓ WEB CONSOLE
- ✓ PARALLEL VERSION
- ✓ 3RD PARTY GATEWAYS
- ✓ EXTERNAL KEY SYSTEMS
- ✓ PROXY-BASED FORWARDING
- ✓ FULL-FEATURED CMS
- ✓ SECURITY FIRE WALLING
- ✓ API GATEWAY
- ✓ API PUBLISHER
- ✓ API DEV PORTAL
- ✓ SCHEDULED REPORTING
- ✓ BATCH REPORTING
- ✓ MULTI-TENANT ARCHITECTURE

HOSTING OPTIONS

On-premise; SaaS

OPEN SOURCE

No

CUSTOMERS

- Johnson Controls
- Crunchbase
- SITA
- Transport For London
- UC Berkeley
- Skype
- wine.com
- JustGiving

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

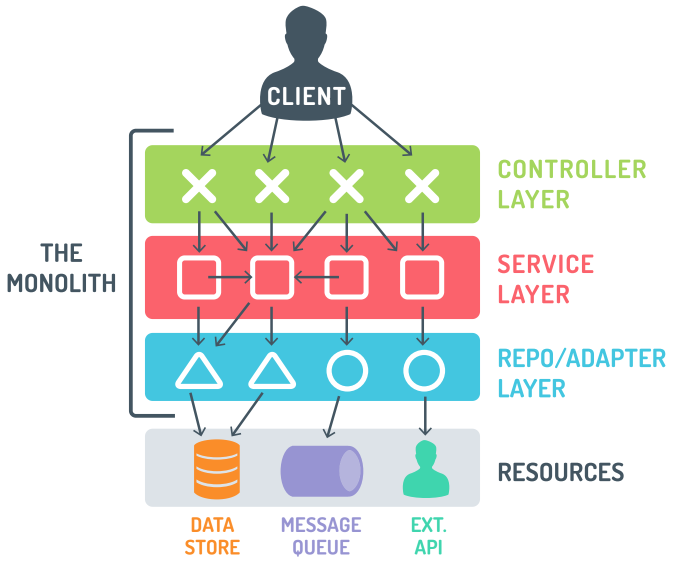
DECOMPOSITION PATTERNS FOR MONOLITHIC APPLICATIONS

CONTENT AND GRAPHICS AUTHORED BY **DANIEL BRYANT**

Many enterprise organizations are looking to decompose monolithic applications into smaller services.

As a result, we are also seeing the emergence of common patterns for breaking down the monoliths. Outlined below are three key decomposition patterns with information about the reasons to use, reasons not to use, and suggested implementation techniques.

First, you'll want to familiarize yourself with the diagram key here:



DECOMPOSING BY **BUSINESS FUNCTIONALITY**

This decomposition pattern involves creating (or extracting) an entire business unit function into a service and providing a corresponding API that can be called by services further upstream in the application stack. A typical example of this pattern is creating a standalone "user service." This is the canonical microservice pattern, which provides a bounded context that models part of the business domain.

REASONS TO USE:

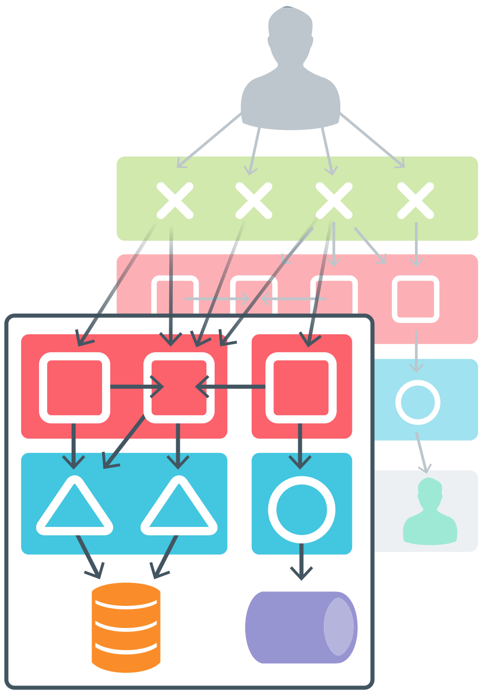
- Increased (or independent) scalability is needed for isolated business functionality.
- Existing application code can easily be divided into logical units of business functionality (i.e. bounded contexts).
- New or modified features specified for the business units being extracted are well-defined.

REASONS NOT TO USE:

- Pre-existing monolith already provides highly cohesive functionality.
- Wide distribution or low cohesion of upstream call sites (e.g. code calling a new service is spread haphazardly throughout the application).
- Requirements of the business unit are in flux, which can lead to failure of project regardless of technical success.

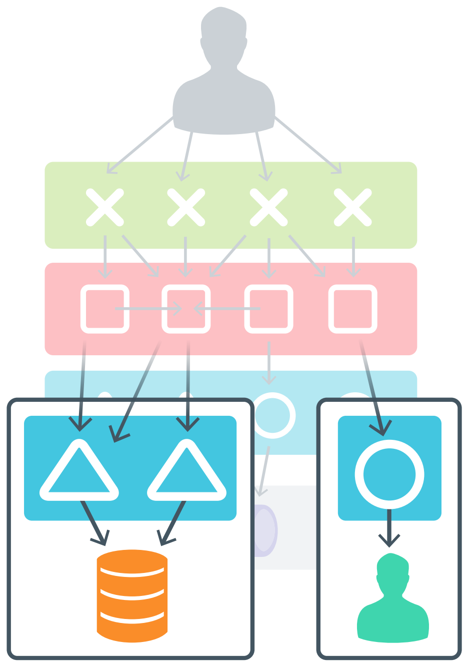
IMPLEMENTATION:

- Implement synchronous operations as REST-like APIs and asynchronous operations via message queue (MQ).
- Consider use of Command Query Responsibility Segregation (CQRS) and Event-Sourcing (ES) patterns.
- Coordination services must be created to orchestrate upstream calls, effectively implementing the Front Controller or Mediator design pattern.



DECOMPOSING BY **TECHNICAL FUNCTIONALITY**

This type of decomposition involves the extraction of a purely technical function into a service, which is often called by a number of separate upstream business unit services or the monolith. An example of this pattern is the creation of an email service.



REASONS TO USE:

- Increased scalability is required for technical functionality (e.g. increased throughput or reduced latency).
- Existing technical functionality can be logically grouped, but is strongly coupled with existing application code.

REASONS NOT TO USE:

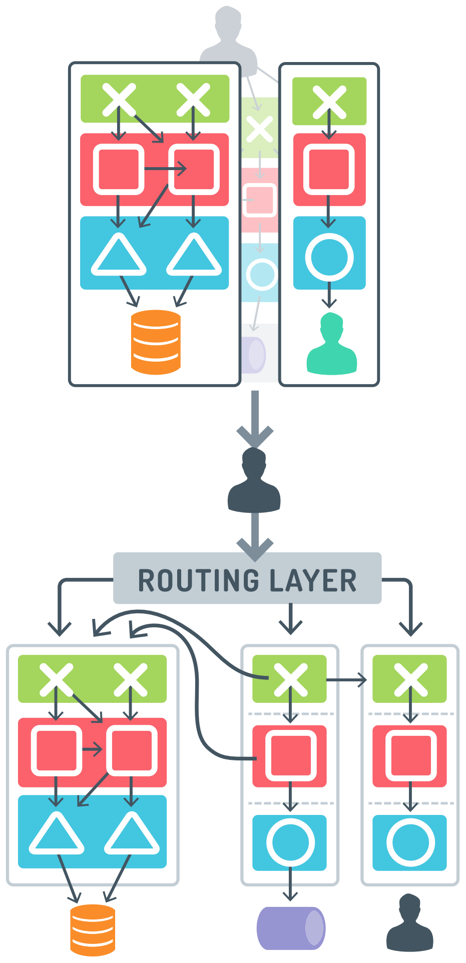
- Unclear requirements from the underlying technology (e.g. implementation of integration to a new third-party service).
- Wide-ranging or long-running transactions in upstream services (distributed transactions are inherently complex).
- Domain-Driven Design implementation required (this decomposition does not provide true bounded contexts).

IMPLEMENTATION:

- Isolate the underlying data store, middleware, or external services behind service façades using a REST-like API for sync communication or MQ platform for async.
- Evaluate the requirements for existing transaction boundaries and consider introducing eventual consistency; confirm if traditionally implemented ACID guarantees are essential throughout application stack.
- Favor async communication to increase robustness.

DECOMPOSING BY **VERTICAL SLICE**

This pattern involves the extraction of a slice vertically through the entire application stack. It differs from "decomposing by business functionality," as the vertical slice includes the provision of a service from the highest (ingress) layer—for example, the UI or public API—whereas the business functionality decomposition only extracts part of the vertical (typically at the bottom of the stack). It also provides an API that upstream services or the monolith may call.



REASONS TO USE:

- Isolated deployment of functionality within slice required (i.e encapsulate what varies).
- Increased or independent scalability of functionality within slice required (e.g. user registration page requires high throughput).
- Good opportunity to prototype and evaluate the microservice approach throughout entire application stack.
- Easy isolation of microservice work (and resulting artifacts) from normal development activities.

REASONS NOT TO USE:

- Large number of layers in stack.
- Convoluted application layering (e.g. use of multiple middleware platforms).
- Unclear requirements for slice functionality.
- Lack of stakeholder commitment; this pattern can be the most difficult to orchestrate across a typical development team.

IMPLEMENTATION:

- Utilize proxy (HAProxy, nginx) as "front controller" to redirect appropriate ingress traffic to new vertical.
- Implement REST-like API and delegate async operations to MQ (potentially using callbacks at ingress UI or public API layer).
- API gateway platform can be utilized to deploy and manage new services.

Connect with
MuleSoft Anypoint Platform.™

The only complete platform for
SOA, SaaS integration and APIs

MuleSoft®

BEYOND TRADITIONAL ENTERPRISE INTEGRATION: BI-MODAL IT

Traditional integration technology and tools are no longer enough for enterprises to stay competitive. Trends like big data, cloud and the Internet of Things are driving a massive proliferation of apps, data and devices. Couple with this the increasing consumer demand for mobile apps and it becomes clear that a company's competitive advantage is tied to how well they can connect their apps, data and devices. Companies that try to keep up with custom code soon find that the tight coupling of their applications locks them down. They can't change quickly enough to meet the demands on the enterprise. A new approach is needed.

The new trend in IT is "Bi-Modal IT." Mode 1 is actually well known. Mode 1 is the "heads down" traditional IT we all have known. Change doesn't happen quickly, but important development and governance happens in Mode 1. Mode 2 is innovation mode, Rapid IT. Mode 2 makes rapid connectivity and app development possible. These two modes fit together and rely upon one another. For example, Mode 1 - through governed APIs - provides access to 'system of record' data. Mode 2 then makes it possible for existing investments in APIs (or SOA) to be leveraged. Mode 2 creates a landscape where application data is mostly available, reducing demands on IT to perform one-off integrations. Mode 2 also allows Line of Business IT to be more autonomous, but work with central IT rather than going around them to get data for their applications.

It sounds good, but it leaves IT wondering how they are going to run everything in two modes. This is where an agility layer is critical for achieving Mode 2: Rapid IT. This layer sits between traditional enterprise integration and new digital business channels such as mobile apps, partner connections and micro apps. This layer combines legacy integration with SaaS integration and API publishing. It provides unified connectivity across all enterprise assets to unlock and maximize the value of their data. At MuleSoft we built the leading integration platform to address this agility layer. MuleSoft's Anypoint Platform is the only complete integration platform for SaaS, SOA and APIs. Anypoint Platform gives IT the freedom to connect what they want, when they want, whether they are event-driven, real-time data or batch data, on-premises or in the cloud.

WRITTEN BY **ROSS MASON,**
FOUNDER AND VICE PRESIDENT OF PRODUCT STRATEGY, MULESOFT INC.

ESB, IPAAS, API MANAGEMENT

Anypoint Platform By Mulesoft

MuleSoft

Anypoint Platform includes core integration runtime environments: **Mule ESB** (for on-premises) and **CloudHub** (an iPaaS). The platform also includes a unified graphical design environment, connectivity to any data source, management, monitoring, and **Anypoint Platform for APIs** (platform and tools to build new APIs, design new interfaces for existing APIs, and more efficiently manage all your APIs).

DESCRIPTION

ANYPOINT INTEGRATION PLATFORM FOR SOA
- SOAP and REST web services
- Service orchestration and governance
- Legacy system modernization

ANYPOINT INTEGRATION PLATFORM FOR SAAS
- Connect applications on-premises and the cloud
- Real-time data integration
- Hybrid security

ANYPOINT INTEGRATION PLATFORM FOR APIS
- API management and developer portal
- API gateway
- Enterprise mobility

FEATURES:

ESB
- ✓ SERVICE ORCHESTRATION
- ✓ FAILOVER HANDLING
- ✓ FEDERATION
- ✓ WEB DEPLOY CONSOLE
- ✓ SERVICE MOCKING

iPaaS
- ✓ SLA MGMT
- ✓ TEAM COLLABORATION
- ✓ BAM
- ✓ LIVE FLOW ACTIVITY
- ✓ SERVICE MOCKING
- ✓ BUSINESS RULES ENGINE

API
- ✓ WEB CONSOLE
- ✓ PARALLEL VERSION
- ✓ 3RD PARTY GATEWAYS
- ✓ EXTERNAL KEY SYSTEMS
- ✓ PROXY-BASED FORWARDING
- ✓ API GATEWAY
- ✓ API PUBLISHER
- ✓ API STORE
- ✓ API DEV PORTAL

DRAG-N-DROP
- ✓ ROUTE CREATION
- ✓ WEB SERVICES
- ✓ DATA SERVICES

HOSTING OPTIONS
On-premise; iPaaS; SaaS

RUNTIME MODE
On app server

ADAPTERS URL
mulesoft.com/cloud-connectors-full

OPEN SOURCE
Yes (with some exclusions)

CUSTOMERS

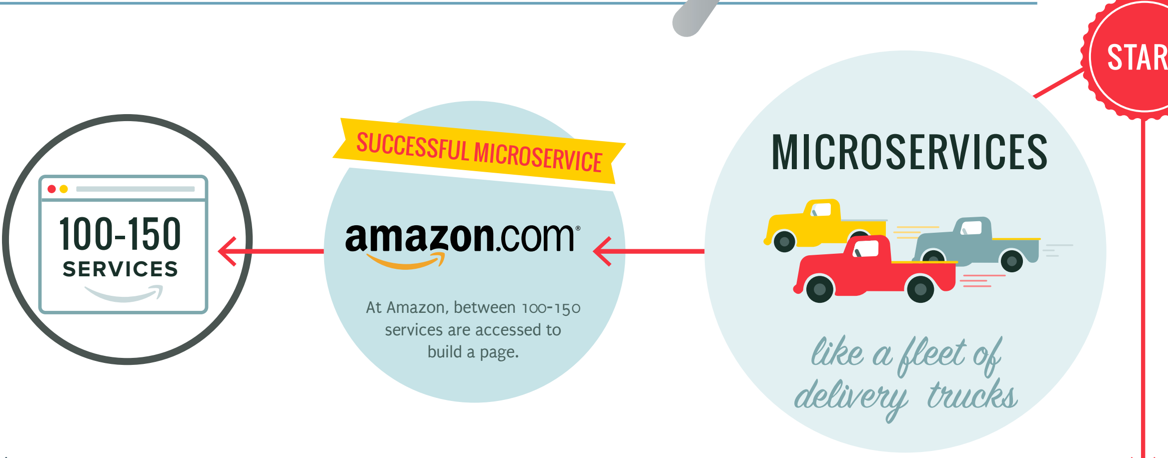
- Amazon
- Barclays
- Ebay
- Ericsson
- Hershey's
- Honeywell
- Mastercard
- NCR
- Nestle
- Sky
- Tesla
- T-Mobile
- Toyota
- UCSF

FULL LIST HERE:
mulesoft.com/customers

BLOG blogs.mulesoft.org | **TWITTER** @muledev @mulesoft | **WEBSITE** mulesoft.com

MICROSERVICES

TAKING A CLOSER LOOK


STAR
MICROSERVICES
like a fleet of delivery trucks
SUCCESSFUL MICROSERVICE
amazon.com
At Amazon, between 100-150 services are accessed to build a page.
100-150 SERVICES

Microservices are just one way to
very scalable method), but it's not t
of software modularity. If you're
architecture, microservices are
something you have to start

Microservice Disadvantages

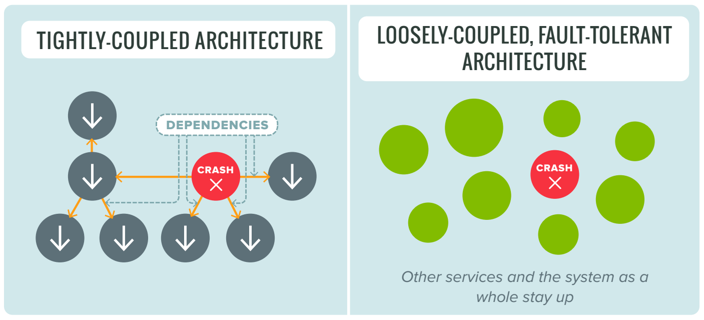
01 Microservices use more remote calls, which are more expensive than in-process calls and don't allow fine-grained APIs.

02 Refactoring and moving code between microservices can be difficult if they're using different technologies.

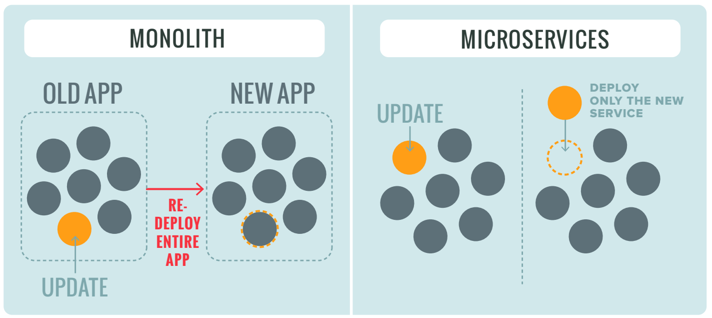
03 Distributed applications are always more complicated. However, most modern web development is distributed, so many developers are already familiar with these challenges.

BENEFITS OF MI

01 Microservices force you to build failure tolerance into your architecture so that if one service goes down, it doesn't bring the whole application down (ex. Netflix's "Popular on Facebook" feature goes down, but the rest of Netflix doesn't go down). Enterprise integration patterns like timeouts, circuit breakers, and bulk-heads will help your software avoid cascading failures.


TIGHTLY-COUPLED ARCHITECTURE
DEPENDENCIES
CRASH
LOOSELY-COUPLED, FAULT-TOLERANT ARCHITECTURE
CRASH
Other services and the system as a whole stay up

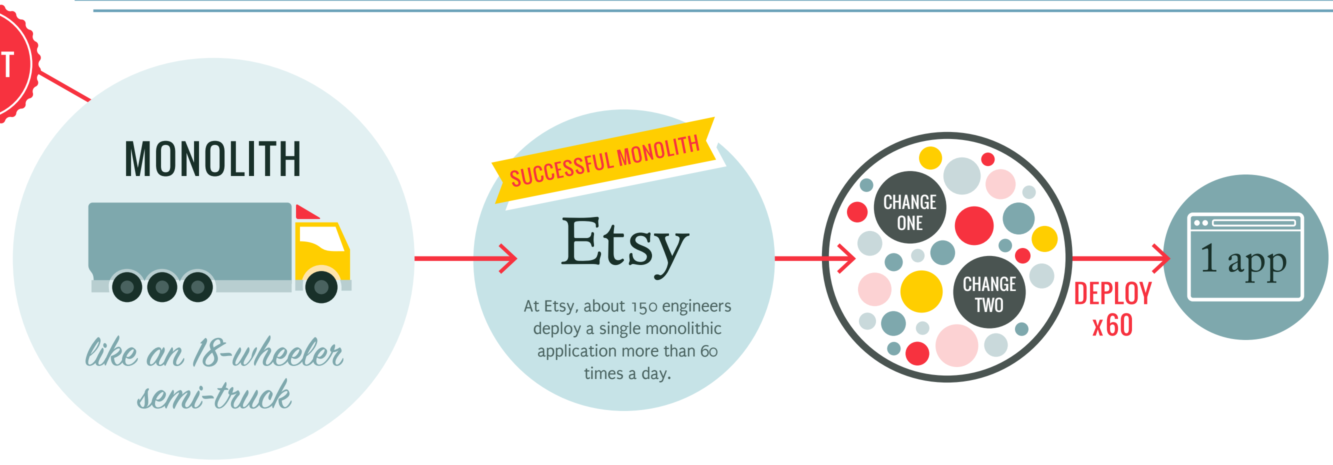
02 Microservices guide you toward establishing independently deployable software components that can be hotfixed or rapidly iterated on without having to re-deploy the entire application.



Microservices are small pieces of functionality that handle single features of an application or system, working together over a distributed environment to create the entire functionality of a software product.

Here are the defining traits of microservices:

- They provide functionality for a small problem domain
- They are built and deployed as separate units with minimal effect on the entire system
- Each one runs in its own process
- Each one can include its own isolated data storage
- They integrate via well-known interfaces


T
MONOLITH
like an 18-wheeler semi-truck
SUCCESSFUL MONOLITH
Etsy
At Etsy, about 150 engineers deploy a single monolithic application more than 60 times a day.
CHANGE ONE
CHANGE TWO
DEPLOY x60
1 app

deploy a software component (a
he only way to have the benefits
building a good componentized
just a deployment option, not
with from the beginning.

Who uses microservices?

NETFLIX

ebay

amazon.com

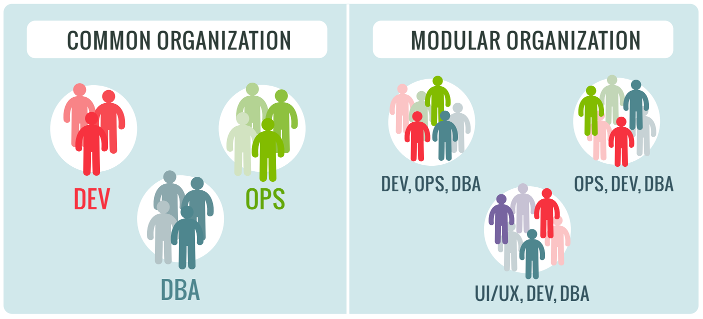
SOUNDCLOUD

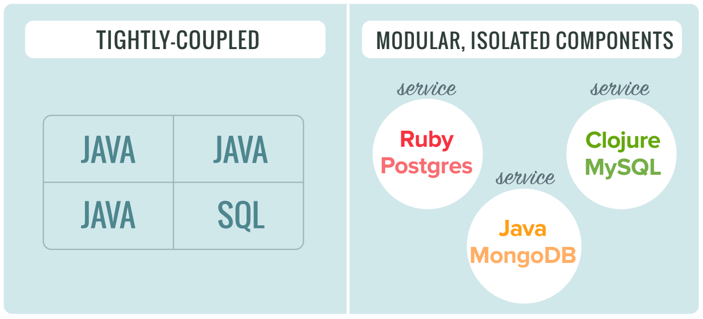
GILT

CROSERVICES

03 Microservices allow separate teams to be highly focused on their service or services since they can modify and deploy the services at will without affecting other teams often. If these are cross-functional teams with specialization in all areas of the software lifecycle, then you're already following a lot of good DevOps practices.

04 Microservices don't need to use the same tools or languages. Each team can use the tools and technology they think are the best fit for the job. Each microservice can have its own database as well.





redhat.

RED HAT® JBOSS®
MIDDLEWARE

BUILD YOUR BUSINESS FASTER, IN A SMARTER WAY

Connect information using a lightweight, flexible, and open platform.

Red Hat JBoss Middleware takes advantage of the open source development model to provide enterprise-grade technologies to:

- Unlock information
- Connect applications, services, devices
- Increase efficiencies
- Accelerate innovations

▸ Learn more at www.redhat.com/jboss

ACCELERATE INNOVATIONS WITH **A CONNECTED BUSINESS**

In today's world, business information is more distributed than it's ever been. Organizations are using new mobile applications to provide differentiated services to their customers. In addition, new modern applications are being developed in the cloud, creating a flourishing new development ecosystem. The ubiquity and simplicity of mobile development and cloud services are driving tremendous innovations that add value, but also lead to more distributed architectures and hybrid deployment strategies.

IN TODAY'S WORLD, BUSINESS INFORMATION IS MORE DISTRIBUTED THAN IT'S EVER BEEN.

Organizations are looking for new systems of engagement and ways to delight their customers by providing additional and sustaining value that differentiates them from competitors. Integration plays a big role in providing the services necessary to attract these customers. Integration of the business information distributed in mobile, cloud, and on-premise applications is required to provide a holistic solution. A connected business is more prepared to provide exciting new systems of engagement and is agile enough to make quick changes and adapt to changing market dynamics.

Red Hat JBoss middleware is a family of lightweight, cloud-friendly, enterprise-grade products that include integration technologies to:

- Unlock business information by connecting all elements of the business.
- Connect applications, services, devices across distributed systems.
- Increase efficiencies between all elements of integration.

Red Hat JBoss integration technologies provide the agility, flexibility, and capability to connect applications, integrate multiple systems in a distributed environment, and accelerate innovations for a connected business. JBoss technologies facilitate communication between all your critical business elements, and allow for a distributed ecosystem that creates a truly connected business.

WRITTEN BY **SAMEER PARULKAR**, PRODUCT MARKETING MANAGER, RED HAT

IPAAS, ESB

JBoss Fuse for xPaaS By Red Hat

redhat.

Fuse for xPaaS features pattern-based implementation based on Apache Camel, and is packaged with ActiveMQ and over 150 connectors to enable rapid development with robust management.

DESCRIPTION

JBoss Fuse for xPaaS bring the same powerful integration capabilities, available on-premise using the JBoss Fuse technology, to the cloud. Integration services on the OpenShift PaaS. JBoss Fuse for xPaaS enables rapid integration, quick prototyping and enables integration solutions to be aligned with CI or DevOps practices. With Fuse for xPaaS integrate everything, everywhere!

FEATURES:

- ✓ CAN BUILD OWN ADAPTERS
- ✓ EXPOSE METRICS VIA JMX
- ✓ SERVICE ORCHESTRATION SUPPORT
- ✓ FAILOVER HANDLING
- ✓ FEDERATION
- ✓ SIMULATES MESSAGING SERVICES
- ✓ WEB-BASED DEPLOYMENT CONSOLE
- ✓ INTEGRATED ARTIFACT REPOSITORY
- ✓ AUTOMATIC RECOVERY OF FLOWS
- ✓ REAL-TIME EVENT-DRIVEN INTEGRATION
- ✓ TRANSFORMS FLOWS INTO WEB APIS
- ✓ TRANSFORMS FLOWS INTO SOA INTERFACES

DRAG-N-DROP

- ✓ ROUTE CREATION
- ✓ WEB SERVICES AND REST APPLICATION CREATION
- ✓ DATA SERVICES/CREATION

CUSTOMERS

- Vodafone
- CenturyLink
- Sabre
- eTrade
- Booz Allen Hamilton
- Sprint
- Dell
- HP

BLOG developer.jboss.org | **TWITTER** @JBoss | **WEBSITE** openshift.com/xpaas

The Future of Developing & Integrating Applications

by Markus Eisele

Think about how far software development has come in just the past five years. It's hard to believe that there was a time when generations of developers spent months or years setting up infrastructures and integrating different applications and backends with each other.

With the widespread adoption of DevOps principles, Platforms as a Service (PaaS) technologies, and the dawn of the Internet of Things, we'll need even more streamlined technologies to build and integrate the complex, distributed systems of the future—systems that will be comprised of countless numbers of consumers and lightweight services.

This article gives you my vision of how building a large project in the future might look. We begin at the stage where you've spec-ed out the application and you sit down with your blank laptop. Nothing is installed yet.

FAST, REVERSIBLE SETUP

Every time I start a project from scratch, I need to keep answering the same questions: which middleware, which IDEs, which frameworks? You have to be convincing and argue for your particular technology choices because major installations or purchases were involved with those choices. I think tomorrow will be different. There are already some PaaS vendors that allow you to leaf through an online repository of software, like an app store/service store, and pick the stack that fits your needs. In the future, this will be even more customizable.

Profiles and containers will be the landscape of your projects in the future.

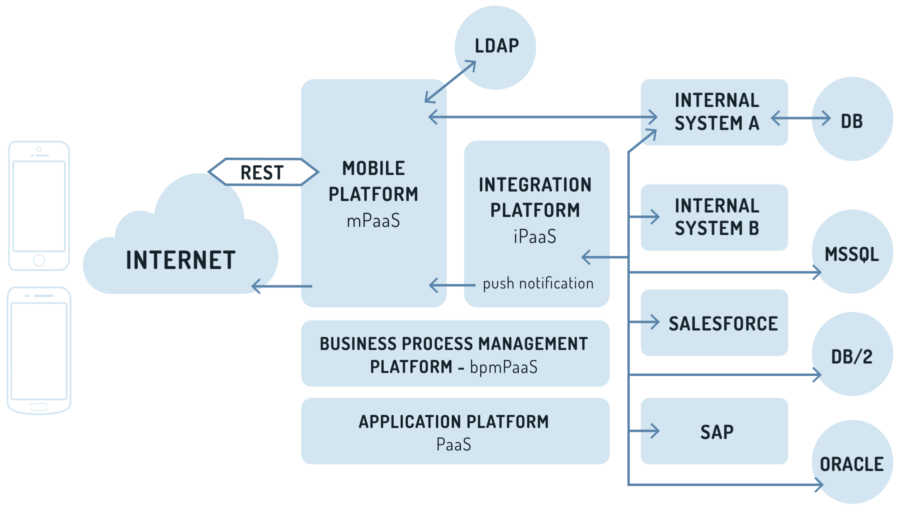
You'll start with your core development environment and just pick a profile and a target. That might be your local machine for now. If you deploy the same profile in the cloud, you will have an identically configured web-based IDE. You won't need to worry so much about where you're using different components at the start, because everything can be changed later. For now, you would pick packaged downloads that come with all the needed dependencies. For example, your IDE will already have the source code repository attached and it will be generated for exactly the type of project you selected. You will also have a complete DevOps lifecycle in place that is already aware of your target environment and supports staging changes.

INTEGRATING CONTAINERS & DYNAMIC RUNTIME CONFIGURATION

Although the initial setup was easy enough for one developer, you'll need more developers on the team as the project grows. In a lot of these highly-scalable future applications, microservices will be a popular architecture. Microservices will encourage organizations to have cross-functional teams with specialists in each area of the stack, rather than separating these specialists into siloed departments. They will also allow developers to focus on the smallest units of work and build larger applications that are modular, lightweight, and resilient to change. Those architectures are unlikely to ever be deployed on one single laptop, so at this point you would want to migrate your requirements.

In future development scenarios, a variety of PaaS tools will make it easy to switch containers from a local configuration to the cloud while maintaining integrations with other containers, including your IDE. Profiles and containers will be the landscape of your projects in the future. The only difference is that you now rely heavily on a network connection that might involve latencies and minor wait times. Local containers and core services will become cloud services.

Messaging (Messaging-as-a-Service) and virtualized data models will allow a seamless connection to a variety of different systems. This new integration layer might be called iPaaS, but there's more to it than the current definition. By having all systems on a cloud platform that knows all the relevant meta information, you won't need to configure most brokers or endpoints. The iPaaS will know which services run on which host, and to which other system they should be connected: the process of scaling, clustering, and failover. Every single switch in hardware properties like IP addresses and ports will be managed and transparently applied without the need to redeploy or configure anything. Whenever your PaaS decides to spin up a new instance, or parts of your system get moved to another instance, the necessary configuration changes will happen automatically. This will also be true for the data (Database-as-a-Service). A profile will include your deployment and be able to dynamically change relevant bindings.



QOS MONITORING

As the team of developers continue to work, using their shared code repository and infrastructure, you'll want to turn your attention to the non-functional aspects of the finished application. These include Quality-of-Service (QoS) requirements, such as average uptime and peak load requirements. Instead of estimating workloads and usages, and making intelligent guesses on how to reach your QoS goals, you'll just define the metrics and thresholds in your PaaS. The system will know how to fulfill them automatically by making the necessary system changes. There is no need for any further human intervention, except to look at the relevant dashboards to check if the metrics are being met.

BUSINESS PROCESSES, MOBILE, & THE INTERNET OF THINGS

As your software grows, you may get new categories of users with varying requirements. To accommodate this it may be easier to think in terms of workflows or task flows, and your PaaS may even be able to suggest that you add other tools like a Business Process Management (BPM) service to your project. Like other cloud add-ons, this would be tightly integrated and able to discover the services you've created.

New devices, servers, and IoT client-devices just need to be registered with your cloud infrastructure and they'll be automatically integrated. Most of the common challenges will be solved by mobile-focused PaaS solutions with back-end services and AeroGear-like client libraries. Devices can be registered for push messaging, and even in a bring-your-own-device organization, there will be suitable security features. By securing the front-end connections, the mPaaS will allow you to connect securely to your iPaaS by bridging complex network topologies and initiating VPN connections to on-premise data sources and systems.

Once this whole system is up and running, integrated consoles will allow you to drill down into specific machines, services, users, and individual lines of code. All of this has been done by your team with no help from external service providers, IT teams, or others not core to developing the application. Running the application requires no administration by you or the team since the infrastructure is self-monitoring and adaptable to changing requirements, potentially including the ability to patch problem areas.

THE NEXT GENERATION OF DEVELOPMENT

Hopefully this doesn't sound too far-fetched. While there's still some way to go before we have all of this, there are already tools that are shifting our industry in a promising direction, including projects and companies like CloudBees, Fabric8, and OpenShift.

Another key component to this utopian future is the typical cloud billing mechanism: pay for what you need, when you need it. Ideally developers wouldn't have to buy development licenses or support ad infinitum, but instead only for the duration of the development process (but with options for support and maintenance services). The customer pays only for what they use. I believe we're just starting to see this kind of technology and workflow on the horizon, and I'm looking forward to when it fully arrives.

WRITTEN BY MARKUS EISELE

Markus Eisele is a Developer Advocate at Red Hat who has worked with Java EE servers for more than 14 years. He speaks at conferences all over the world and is the Java Community leader of the German DOAG e.V. He is also a Java Champion, former ACE director, and DZone Most Valuable Blogger. His Twitter handle is @myfear.

The REST API Maturity Model

Roy Fielding derived REST from a series of constraints [1]. Each constraint makes the system more scalable by requiring less coupling on the client side. Based on these constraints, Leonard Richardson built a maturity model to help developers build increasingly RESTful (and therefore increasingly web-optimized) systems [2] [3]. This is an adaptation of that maturity model, explained and expanded with examples.

[1] https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf [2] http://www.crummy.com/writing/speaking/2008-QCon/act3.html [3] http://shop.oreilly.com/product/0636920028468.do

LEVEL 0: PLAIN OLD RPC

ESSENTIAL ATTRIBUTES: One URI, one HTTP method

KINDA LIKE: Buying food at a drive-thru

NATURAL LANGUAGE EXAMPLE:

Alice: *How much is a Coke?*
Bob: *$2 cash, $2.25 credit.*
Alice: *I'll take one Coke. Here's $2 cash.*
Bob: *Here's your Coke.*

HTTP EXAMPLE:

```
POST /drinkService HTTP/1.1
<checkPrice brand = “Coca Cola Classic”
size = “20oz”/>
HTTP/1.1 200 OK
<checkPriceInfo>
   <price = ”2” paymentType = “cash”/>
   <price = “2.25” paymentType = “credit”/>
</checkPriceInfo>
```

LEVEL 1: INTRODUCING RESOURCES

ESSENTIAL ATTRIBUTES: One URI for each resource

KINDA LIKE: Buying food at a counter

NATURAL LANGUAGE EXAMPLE:

Alice: *What drinks do you have?*
Bob: *Coke for $2, Pepsi for $1.85, Sprite for $2.*
Alice: *I want a Sprite. Here's $2.*
Bob: *Here's your Sprite.*

HTTP EXAMPLE:

```
POST /drinks HTTP/1.1
<checkPrice />
HTTP/1.1 200 OK
<checkPriceItem>
   <brand = “coke” price = “2”/>
   <brand = “pepsi” price = “1.85”/>
   <brand = “sprite” price = “2”/>
</checkPriceItem>
POST /drinks/sprite HTTP/1.1
<buyItem paymentID =
“2343b23930932bfd90ac4”/>
```

LEVEL 2: MULTIPLE VERBS

ESSENTIAL ATTRIBUTES: Multiple URIs, multiple HTTP methods

KINDA LIKE: Entering a pub for the first time

NATURAL LANGUAGE EXAMPLE:

Alice: *I've never been to Bob's Pub before. What can I do here? I'm legally allowed to drink.*
Bob: *You can order a drink for $3, order food for $6, or play darts for $1.*
Alice: *I'll take a set of darts. Here is $1.*
Bob: *Here is a set of darts.*

HTTP EXAMPLE:

```
GET /bobspub/menu?drinkingAge=legal HTTP/1.1
HTTP/1.1 200 OK
<pubOfferings>
   <offering itemid=”43634” item = “drink”
   price = “3”/>
   <offering itemid=”43635” item = “food”
   price = “6”/>
   <offering itemid=”43637” item = “darts”
   price = “1”/>
</pubOfferings>
POST /bobspub/menu/43637
   <paymentInfo>
      <accountName = “alice1234”/>
      <amount = “1”/>
   </paymentInfo>
```

LEVEL 3: HYPERMEDIA

ESSENTIAL ATTRIBUTES: Resources explain themselves

KINDA LIKE: Transactions at a bank

NATURAL LANGUAGE EXAMPLE:

Alice: *How can I deposit my gold?*
Bob: *You can put gold into your safe deposit box #23438aa40fd3 or convert it to cash and deposit the cash in your checking account #9909n339.*
Alice: *Here is $5000 in gold for safe deposit box #23438aa40fd3.*
Bob: *OK. Your safe deposit box #23438aa40fd3 now has $5000 in gold. Also, just call me at BankNumber and have your safe deposit box number (#23438aa40fd3) ready if you want to withdraw anything.*

[months later, Ted purchases Bob's Bank]

Alice: *I'd like to withdraw $3000 in gold from safe deposit box #23438aa40fd3.*
Ted: *Here's your gold. Your safe deposit box #23438aa40fd3 now has $2000 in gold.*

HTTP EXAMPLE:

```
GET /accounts/alice1337 HTTP/1.1
HTTP/1.1 200 OK
<accountsList>
   <account name = “safeDepositBox” currentValue = “0”>
      <link rel = “deposit” uri = “/accounts/alice1337/23438aa40fd3”/>
      <link rel = “withdraw” uri = “/accounts/alice1337/23438aa40fd3”/>
   </account>
   <account name = “checkingAccount” currentValue = “2000”>
       <link rel = “deposit” uri = “/accounts/alice1337/9909n339”/>
       <link rel = “withdraw” uri = “/accounts/alice1337/9909n339”/>
   </account>
</accountsList>
POST /accounts/alice1337/23438aa40fd3 HTTP/1.1
<transactionInfo>
   <deposit medium = “gold” value = “5000”/>
</transactionInfo>
HTTP/1.1 201 Created
<accountInfo>
   <account name = “safeDepositBox” currentContents = “gold”
   currentValue = “5000”>
      <uri = “/accounts/23438aa40fd3”/>
   </account>
   <link rel = “deposit” uri = “/accounts/23438aa40fd3”/>
   <link rel = “withdraw” uri = “/accounts/23438aa40fd3”/>
</accountInfo>
POST /accounts/alice1337/23438aa40fd3 HTTP/1.1
<transactionInfo>
   <withdraw medium = “gold” value = “3000”/>
</transactionInfo>
```

SOLUTIONS DIRECTORY

***Notice to readers:** The data in this section is impartial and not influenced by any vendor.*

This directory of enterprise integration tools, frameworks, and platforms provides comprehensive, factual comparison data gathered from third-party sources and the tool creators' organizations. Solutions in the directory are selected based on several impartial criteria including solution maturity, technical innovativeness, relevance, and data availability. The solution summaries underneath the product titles are based on the organization's opinion of its most distinguishing features. Understand that not having certain features is a positive thing in some cases. Fewer unnecessary features sometimes translates into more flexibility and better performance.

NOTE: The bulk of information gathered about these solutions is not present in these quarter-page profiles. **To view an extended profile of any product, simply go to:** dzone.com/research/enterpriseintegration

API MANAGEMENT | RESEARCH PARTNER

3scale API Management 3scale

API management capabilities delivered as-a-service. A unique hybrid cloud architecture that allows best-in-class web scalability.

- ✓ API GATEWAY
- ✓ API PUBLISHER
- ☐ API STORE
- ✓ API DEV PORTAL

FEATURES:

- ✓ WEB CONSOLE
- ✓ PARALLEL VERSION
- ✓ 3RD PARTY GATEWAYS
- ✓ EXTERNAL KEY SYSTEMS
- ✓ PROXY-BASED FORWARDING

OPEN SOURCE
No

HOSTING OPTIONS
On-premise; SaaS

TWITTER @3scale

WEBSITE 3scale.net

MESSAGE QUEUE

ActiveMQ Apache

Apache ActiveMQ supports many cross-language clients and protocols, easy-to-use integration patterns, and supports JMS 1.1 and J2EE 1.4.

TYPE
Library + bindings

JMS
Version 1.1

SPECIALTY
Broker (central node)

✓ TLS ON ALL MESSAGES

OPEN SOURCE
Yes

FEATURES:

- ✓ JOURNALING
- ✓ TRACING SUPPORT
- ✓ DISK WRITE SUPPORT
- ☐ ALL MSGS IN MEMORY
- ✓ ALL TASKS ASYNC
- ✓ THREAD-TO-THREAD

TWITTER @activemq

WEBSITE activemq.apache.org

MESSAGE QUEUE

Amazon SQS Amazon

Amazon SQS moves data between distributed components of applications without losing messages or requiring components to be always available.

TYPE
Cloud service

JMS
Non-Java interfaces supported

SPECIALTY
Low-latency P2P

✓ **TLS ON ALL MESSAGES**

OPEN SOURCE
No

FEATURES:

☐ JOURNALING
✓ TRACING SUPPORT
✓ DISK WRITE SUPPORT
✓ ALL MSGS IN MEMORY
✓ ALL TASKS ASYNC
✓ THREAD-TO-THREAD

TWITTER @awscloud **WEBSITE** aws.amazon.com/sqs

API MANAGEMENT RESEARCH PARTNER

Anypoint Platform for APIs Mulesoft

A platform for designing great APIs, managing them throughout their lifecycle, and gleaning business insights through analytics.

✓ API GATEWAY
✓ API PUBLISHER
✓ API STORE
✓ API DEV PORTAL

OPEN SOURCE
No

FEATURES:

✓ WEB CONSOLE
✓ PARALLEL VERSION
✓ 3RD PARTY GATEWAYS
✓ EXTERNAL KEY SYSTEMS
✓ PROXY-BASED FORWARDING

HOSTING OPTIONS
On-premise; SaaS

TWITTER @Mulesoft **WEBSITE** mulesoft.com

API MANAGEMENT

Apigee Edge Apigee

Apigee Edge provides tools to manage the digital value chain from API exposure to API consumption, and to measure the success of API programs with analytics.

✓ API GATEWAY
✓ API PUBLISHER
✓ API STORE
✓ API DEV PORTAL

OPEN SOURCE
Yes

FEATURES:

✓ WEB CONSOLE
✓ PARALLEL VERSION
✓ 3RD PARTY GATEWAYS
✓ EXTERNAL KEY SYSTEMS
✓ PROXY-BASED FORWARDING

HOSTING OPTIONS
On-premise; SaaS

TWITTER @apigee **WEBSITE** apigee.com

IPAAS

AtomSphere Dell Boomi

AtomSphere iPaaS enables customers to integrate any combination of cloud or on-premises applications without software, appliances, or coding.

DEPENDENCIES
N/A

PROJECT TYPES
A2A SOA B2B
Data Mgmt API Mgmt
MBaaS

HOSTING OPTIONS
iPaaS; On-premise

OPEN SOURCE
No

FEATURES:

✓ SLA MGMT
✓ TEAM COLLAB
✓ BAM
✓ LIVE FLOW ACTIVITY
✓ SERVICE MOCKING
✓ BUSINESS RULES ENGINE

TWITTER @boomi **WEBSITE** boomi.com

ESB

Aurea Sonic ESB Aurea Software

Aurea Sonic ESB features a highly distributed dynamic routing architecture as well as an automated deployment function to facilitate continuous delivery.

DEPENDENCIES
Java 1.6

DRAG-N-DROP
- ✓ ROUTE CREATION
- ✓ WEB SERVICES
- ✓ DATA SERVICES

RUNTIME MODE
Standalone; On app server

OPEN SOURCE
No

FEATURES:
- ✓ BPEL
- ✓ SERVICE ORCHESTRATION
- ✓ FAILOVER HANDLING
- ✓ FEDERATION
- ✓ WEB DEPLOY CONSOLE
- ✓ SERVICE MOCKING

ADAPTERS URL
aurea.com

TWITTER N/A | **WEBSITE** aurea.com

API MANAGEMENT

Axway API Management Axway

Axway provides an API management and security platform, featuring a highly configurable gateway for enterprise-grade integration and analytics.

- ✓ API GATEWAY
- ✓ API PUBLISHER
- ☐ API STORE
- ✓ API DEV PORTAL

OPEN SOURCE
No

FEATURES:
- ✓ WEB CONSOLE
- ✓ PARALLEL VERSION
- ✓ 3RD PARTY GATEWAYS
- ✓ EXTERNAL KEY SYSTEMS
- ✓ PROXY-BASED FORWARDING

HOSTING OPTIONS
Hardware appliance; On-premise; SaaS

TWITTER @axway | **WEBSITE** axway.com

INT. SUITE, IPAAS

BizTalk Server Microsoft

BizTalk server is an integration platform that handles orchestration, business rules, and has adaptors for protocols such as REST and JSON.

DEPENDENCIES
.NET

PROJECT TYPES
A2A SOA B2B BPM

HOSTING OPTIONS
On-premise

OPEN SOURCE
No

FEATURES:
- ✓ SLA MGMT
- ✓ TEAM COLLAB
- ✓ BAM
- ✓ LIVE FLOW ACTIVITY
- ✓ SERVICE MOCKING
- ✓ BUSINESS RULES ENGINE

TWITTER @MS_BizTalk | **WEBSITE** microsoft.com/BizTalk

IPAAS

Business Operations Platform Fujitsu

Fujitsu's Business Operations Platform features SOA Integration, case management capability, and a collaborative workspace.

DEPENDENCIES
Java 1.7

PROJECT TYPES
A2A SOA B2B BPM Data Mgmt

HOSTING OPTIONS
iPaaS; On-premise

OPEN SOURCE
No

FEATURES:
- ✓ SLA MGMT
- ✓ TEAM COLLAB
- ✓ BAM
- ✓ LIVE FLOW ACTIVITY
- ✓ SERVICE MOCKING
- ✓ BUSINESS RULES ENGINE

TWITTER @interstage | **WEBSITE** fujitsu.com

API MANAGEMENT — RESEARCH PARTNER

CA API Management CA

CA's API management tool includes in-depth security and performance options, global API management, and several deployment options.

- ✓ API GATEWAY
- ✓ API PUBLISHER
- ✓ API STORE
- ✓ API DEV PORTAL

FEATURES:

- ✓ WEB CONSOLE
- ✓ PARALLEL VERSION
- ✓ 3RD PARTY GATEWAYS
- ✓ EXTERNAL KEY SYSTEMS
- ✓ PROXY-BASED FORWARDING

OPEN SOURCE
No

HOSTING OPTIONS
Hardware appliance; On-premise; SaaS

TWITTER @CAapi

WEBSITE ca.com/api

INT. FRAMEWORK

Camel Apache

Camel empowers developers to define routing and mediation rules in a variety of domain-specific languages, including a Java-based fluent API.

CURRENT VERSION
2.14

UPDATES
Monthly

DSLS
XML Java Groovy Scala Kotlin

LANGUAGE ECOSYSTEM
Java

OPEN SOURCE
Yes

STRENGTHS

- Includes several DSLs including Scala and a Java-based fluent API
- Uses URIs to work with any kind of transport or messaging models such as HTTP
- Minimal dependencies allow for easy embedding in Java applications
- Provides integration with Spring, Blueprint, and Guice
- Connectors to different enterprise systems, SaaS, cloud, protocols, and file.

DEPLOYMENT OPTIONS
Web container; JEE container; OSGi environment; Spring container; Spring XD container

COMPONENTS: http://camel.apache.org/components.html

IPAAS — RESEARCH PARTNER

CloudHub Mulesoft

CloudHub is a global cloud service with load balancing, multi-tenancy, high-availability, horizontal/vertical scaling, and PCI compliance.

DEPENDENCIES
N/A

PROJECT TYPES
A2A SOA B2B Data Mgmt API Mgmt

HOSTING OPTIONS
iPaaS

OPEN SOURCE
No

FEATURES:

- ✓ SLA MGMT
- ✓ TEAM COLLAB
- ✓ BAM
- ✓ LIVE FLOW ACTIVITY
- ✓ SERVICE MOCKING
- ✓ BUSINESS RULES ENGINE

TWITTER @mulesoft

WEBSITE cloudhub.io

ESB

Ensemble Intersystems

Ensemble is an integration platform providing SOA, EDA, and composite service solutions supporting a wide range of technologies and document formats.

DEPENDENCIES
N/A

DRAG-N-DROP

- ✓ ROUTE CREATION
- ☐ WEB SERVICES
- ☐ DATA SERVICES

RUNTIME MODE
Standalone

FEATURES:

- ☐ BPEL
- ✓ SERVICE ORCHESTRATION
- ✓ FAILOVER HANDLING
- ✓ FEDERATION
- ✓ WEB DEPLOY CONSOLE
- ✓ SERVICE MOCKING

OPEN SOURCE
No

ADAPTERS URL
docs.intersystems.com

TWITTER @intersystems

WEBSITE intersystems.com

IPAAS

Elastic Integration Platform SnapLogic

Snaplogic provides a multi-tenant cloud service, with EAI and ETL for Big Data integration in a single platform.

DEPENDENCIES
N/A

PROJECT TYPES
A2A SOA
Data Mgmt API Mgmt

HOSTING OPTIONS
iPaaS

OPEN SOURCE
No

FEATURES:

- ☐ SLA MGMT
- ✓ TEAM COLLAB
- ☐ BAM
- ✓ LIVE FLOW ACTIVITY
- ✓ SERVICE MOCKING
- ☐ BUSINESS RULES ENGINE

TWITTER @snaplogic **WEBSITE** snaplogic.com

IPAAS

Flowgear Flowgear

Flowgear features integration prototyping and API-binding, and connects disparate data sources without VPN.

DEPENDENCIES
.NET

PROJECT TYPES
A2A SOA B2B
Data Mgmt API Mgmt
MBaaS

HOSTING OPTIONS
iPaaS

OPEN SOURCE
No

FEATURES:

- ☐ SLA MGMT
- ✓ TEAM COLLAB
- ✓ BAM
- ✓ LIVE FLOW ACTIVITY
- ✓ SERVICE MOCKING
- ☐ BUSINESS RULES ENGINE

TWITTER @flowgear **WEBSITE** flowgear.net

INT. SUITE, IPAAS

Fujitsu RunMyProcess Fujitsu

RunMyProcess features a drag-and-drop interface to create business applications, bundled with a database and version control systems as well as an app server and repository.

DEPENDENCIES
N/A

PROJECT TYPES
A2A SOA B2B BPM
Data Mgmt API Mgmt
MBaaS

HOSTING OPTIONS
iPaaS

OPEN SOURCE
No

FEATURES:

- ✓ SLA MGMT
- ✓ TEAM COLLAB
- ✓ BAM
- ✓ LIVE FLOW ACTIVITY
- ✓ SERVICE MOCKING
- ✓ BUSINESS RULES ENGINE

TWITTER @runmyprocess **WEBSITE** runmyprocess.com

IPAAS

HANA Cloud Integration SAP

HANA includes prebuilt integration flows from popular SAP cloud applications, and SAP HANA-optimized tools to natively extract and load data.

DEPENDENCIES
N/A

PROJECT TYPES
A2A SOA BPM
Data mgmt

HOSTING OPTIONS
iPaaS

OPEN SOURCE
No

FEATURES:

- ☐ SLA MGMT
- ✓ TEAM COLLAB
- ✓ BAM
- ✓ LIVE FLOW ACTIVITY
- ✓ SERVICE MOCKING
- ✓ BUSINESS RULES ENGINE

TWITTER @SAPTechnology **WEBSITE** sap.com

MESSAGE QUEUE

HornetQ Red Hat

HornetQ is message-oriented middleware designed to build multi-protocol, embeddable, clustered, and asynchronous messaging systems.

TYPE
Library + bindings

JMS
Version 2.0

SPECIALTY
Broker (central node)

✓ **TLS ON ALL MESSAGES**

OPEN SOURCE
Yes

FEATURES:

- ✓ JOURNALING
- ✓ TRACING SUPPORT
- ✓ DISK WRITE SUPPORT
- ☐ ALL MSGS IN MEMORY
- ✓ ALL TASKS ASYNC
- ✓ THREAD-TO-THREAD

TWITTER @hornetq **WEBSITE** hornetq.jboss.org

MESSAGE QUEUE

IBM MQ Advanced IBM

IBM MQ Advanced features a wide set of supported platform configurations, programming interfaces, and languages.

TYPE
Standalone software

JMS
Version 2.0

SPECIALTY
Broker (central node)

✓ **TLS ON ALL MESSAGES**

OPEN SOURCE
No

FEATURES:

- ✓ JOURNALING
- ✓ TRACING SUPPORT
- ✓ DISK WRITE SUPPORT
- ✓ ALL MSGS IN MEMORY
- ☐ ALL TASKS ASYNC
- ✓ THREAD-TO-THREAD

TWITTER @IBMmessaging **WEBSITE** ibm.com

MESSAGE QUEUE

IronMQ Iron.io

IronMQ is a highly available HTTP & REST-based message queue with persistence, one-time delivery, push, poll, and long-polling.

TYPE
Standalone software

JMS
Non-Java interfaces supported

SPECIALTY
Broker (central node)

✓ **TLS ON ALL MESSAGES**

OPEN SOURCE
No

FEATURES:

- ✓ JOURNALING
- ✓ TRACING SUPPORT
- ✓ DISK WRITE SUPPORT
- ☐ ALL MSGS IN MEMORY
- ✓ ALL TASKS ASYNC
- ☐ THREAD-TO-THREAD

TWITTER @getiron **WEBSITE** iron.io/mq

ESB RESEARCH PARTNER

JBoss Fuse Red Hat

Lightweight JBoss Fuse can be deployed in several different configurations, and enables businesses to integrate everything, everywhere.

DEPENDENCIES
Java 1.7

DRAG-N-DROP

- ✓ ROUTE CREATION
- ✓ WEB SERVICES
- ✓ DATA SERVICES

RUNTIME MODE
On app server

FEATURES:

- ✓ BPEL
- ✓ SERVICE ORCHESTRATION
- ✓ FAILOVER HANDLING
- ✓ FEDERATION
- ☐ WEB DEPLOY CONSOLE
- ☐ SERVICE MOCKING

OPEN SOURCE
Yes

ADAPTERS URL
access.redhat.com/documentation

TWITTER @jboss **WEBSITE** jboss.org/products/fuse

IPAAS | RESEARCH PARTNER

JBoss Fuse for xPaaS Red Hat

JBoss Fuse for xPaaS extends the pattern-based integration capabilities on OpenShift PaaS, enabling rapid integration, quick prototyping, and improved developer productivity.

DEPENDENCIES
N/A

PROJECT TYPES
A2A SOA B2B

HOSTING OPTIONS
iPaaS

OPEN SOURCE
Yes

FEATURES:

- ☐ SLA MGMT
- ✓ TEAM COLLAB
- ☐ BAM
- ✓ LIVE FLOW ACTIVITY
- ✓ SERVICE MOCKING
- ☐ BUSINESS RULES ENGINE

TWITTER @Jboss | **WEBSITE** openshift.com

INT. SUITE, IPAAS

LegaSuite Integration Rocket Software

LegaSuite Integration provides web services for applications running on IBM z, IBM i, UNIX, or Windows server.

DEPENDENCIES
Java 1.7

PROJECT TYPES
A2A SOA B2B BPM Data Mgmt API Mgmt MBaaS

HOSTING OPTIONS
On-premise; iPaaS

OPEN SOURCE
No

FEATURES:

- ☐ SLA MGMT
- ✓ TEAM COLLAB
- ☐ BAM
- ✓ LIVE FLOW ACTIVITY
- ✓ SERVICE MOCKING
- ☐ BUSINESS RULES ENGINE

TWITTER @Rocket | **WEBSITE** rocketsoftware.com

API MANAGEMENT

Mashery API Management Intel

Mashery is a SaaS-based API Management solution with a single user console to scale, monitor, and distribute APIs.

- ✓ API GATEWAY
- ✓ API PUBLISHER
- ☐ API STORE
- ✓ API DEV PORTAL

FEATURES:

- ✓ WEB CONSOLE
- ✓ PARALLEL VERSION
- ☐ 3RD PARTY GATEWAYS
- ✓ EXTERNAL KEY SYSTEMS
- ✓ PROXY-BASED FORWARDING

OPEN SOURCE
No

HOSTING OPTIONS
On-premise; SaaS

TWITTER @mashery | **WEBSITE** mashery.com

ESB | RESEARCH PARTNER

Mule ESB Mulesoft

Mule ESB is a solution for SOA, SaaS, and APIs, allowing developers to connect what they need to on-premises or in the cloud.

DEPENDENCIES
Java 1.7

DRAG-N-DROP

- ✓ ROUTE CREATION
- ✓ WEB SERVICES
- ✓ DATA SERVICES

RUNTIME MODE
Standalone; On app server

FEATURES:

- ✓ BPEL
- ✓ SERVICE ORCHESTRATION
- ✓ FAILOVER HANDLING
- ✓ FEDERATION
- ✓ WEB DEPLOY CONSOLE
- ✓ SERVICE MOCKING

OPEN SOURCE
Yes

ADAPTERS URL
mulesoft.com/cloud-connectors-full

TWITTER @Mulesoft | **WEBSITE** mulesoft.com

ESB

Neuron ESB Neudesic

Neuron ESB is an application integration and web service platform built entirely on Microsoft .NET to provide real-time consistent messaging services.

DEPENDENCIES
.NET

DRAG-N-DROP
- ✓ ROUTE CREATION
- ✓ WEB SERVICES
- ✓ DATA SERVICES

RUNTIME MODE
Standalone; On app server

FEATURES:
- ☐ BPEL
- ✓ SERVICE ORCHESTRATION
- ✓ FAILOVER HANDLING
- ✓ FEDERATION
- ☐ WEB DEPLOY CONSOLE
- ✓ SERVICE MOCKING

OPEN SOURCE
No

ADAPTERS URL
neuronesb.com

TWITTER @neuronESB

WEBSITE neuronesb.com

INT. FRAMEWORK

NServiceBus

NServiceBus is a popular integration framework for the .NET ecosystem that uses a WCF-remote-procedure-call type of API.

CURRENT VERSION
5.0.1

UPDATES
Weekly

DSLS
None

LANGUAGE ECOSYSTEM
.NET

OPEN SOURCE
Yes

STRENGTHS
- Has capabilities usually associated with business process engines
- Monitoring includes a unique SLA violation countdown metric
- Largest community for a .NET integration framework

DEPLOYMENT OPTIONS
Web container

COMPONENTS: http://docs.particular.net/nservicebus/containers

ESB

Oracle Service Bus Oracle

OSB transforms complex architectures into agile networks by connecting, virtualizing, and managing interactions between services and apps.

DEPENDENCIES
Java

DRAG-N-DROP
- ✓ ROUTE CREATION
- ✓ WEB SERVICES
- ✓ DATA SERVICES

RUNTIME MODE
Standalone; On app server

FEATURES:
- ✓ BPEL
- ✓ SERVICE ORCHESTRATION
- ☐ FAILOVER HANDLING
- ✓ FEDERATION
- ✓ WEB DEPLOY CONSOLE
- ✓ SERVICE MOCKING

OPEN SOURCE
No

ADAPTERS URL
N/A

TWITTER @oracle

WEBSITE oracle.com

MESSAGE QUEUE

RabbitMQ Pivotal

Pivotal RabbitMQ is an open source asynchronous message server designed for the cloud, with simple adoption and deployment, several language clients, and a small footprint.

TYPE
Standalone software

JMS
Version 1.1

SPECIALTY
Broker (central node)

- ✓ **TLS ON ALL MESSAGES**

OPEN SOURCE
Yes

FEATURES:
- ✓ JOURNALING
- ✓ TRACING SUPPORT
- ✓ DISK WRITE SUPPORT
- ☐ ALL MSGS IN MEMORY
- ✓ ALL TASKS ASYNC
- ✓ THREAD-TO-THREAD

TWITTER @Pivotal

WEBSITE pivotal.io

API MANAGEMENT

SAP API Management SAP

SAP's API Management provides a comprehensive infrastructure for API creation, security, and management as well as back-end services for developing client apps.

- ✓ API GATEWAY
- ✓ API PUBLISHER
- ☐ API STORE
- ✓ API DEV PORTAL

FEATURES:

- ✓ WEB CONSOLE
- ✓ PARALLEL VERSION
- ✓ 3RD PARTY GATEWAYS
- ✓ EXTERNAL KEY SYSTEMS
- ✓ PROXY-BASED FORWARDING

OPEN SOURCE
No

HOSTING OPTIONS
On-premise

TWITTER @SAP

WEBSITE sap.com

INT. FRAMEWORK

Spring Integration Pivotal

Spring Integration is a lightweight integration framework that provides an alternative to ESBs or integration suites that integrates both systems and components.

CURRENT VERSION
4

UPDATES
Semi-annually

DSLS
XML Java Groovy Scala

LANGUAGE ECOSYSTEM
Java

OPEN SOURCE
Yes

STRENGTHS

- Canonical implementation of Enterprise Integration Patterns
- Spring XD is an available option to unify Spring Integration, Batch, and Data
- Spring XD extends Spring Integration by assembling component modules into a stream
- Provides control over message flow using MessageChannels as first class elements

DEPLOYMENT OPTIONS
Web container; JEE container; Spring container; Spring XD container

COMPONENTS: http://docs.spring.io/spring-integration/docs/

ESB

Talend ESB Talend

Talend's ESB product simplifies the connection, mediation, and management of services and applications.

DEPENDENCIES
Java

DRAG-N-DROP

- ✓ ROUTE CREATION
- ✓ WEB SERVICES
- ☐ DATA SERVICES

RUNTIME MODE
Standalone; On app server

OPEN SOURCE
Yes

FEATURES:

- ✓ BPEL
- ✓ SERVICE ORCHESTRATION
- ✓ FAILOVER HANDLING
- ☐ FEDERATION
- ☐ WEB DEPLOY CONSOLE
- ☐ SERVICE MOCKING

ADAPTERS URL
talendforge.org/components/

TWITTER @talend

WEBSITE talend.com/products/esb

MESSAGE QUEUE

TIBCO EMS TIBCO

TIBCO EMS is standards-based messaging middleware that simplifies and accelerates integration and management of data distribution.

TYPE
Library + bindings

JMS
Version 1.1

SPECIALTY
Low-latency P2P

- ✓ TLS ON ALL MESSAGES

OPEN SOURCE
No

FEATURES:

- ✓ JOURNALING
- ✓ TRACING SUPPORT
- ✓ DISK WRITE SUPPORT
- ✓ ALL MSGS IN MEMORY
- ☐ ALL TASKS ASYNC
- ✓ THREAD-TO-THREAD

TWITTER @tibco

WEBSITE tibco.com

ESB

UltraESB AdroitLogic

UltraESB uses non-blocking IO with ZeroCopy/DMA support of the hardware and OS, and a RAM disk instead of heap memory to reduce garbage collection overhead.

DEPENDENCIES
Java 1.6

DRAG-N-DROP
- ☐ ROUTE CREATION
- ☐ WEB SERVICES
- ☐ DATA SERVICES

RUNTIME MODE
Standalone; On app server

FEATURES:
- ☐ BPEL
- ☑ SERVICE ORCHESTRATION
- ☑ FAILOVER HANDLING
- ☑ FEDERATION
- ☑ WEB DEPLOY CONSOLE
- ☑ SERVICE MOCKING

OPEN SOURCE
Yes

ADAPTERS URL
docs.adroitlogic.org

TWITTER @AdroitLogic

WEBSITE adroitlogic.org

API MANAGEMENT, SUITE

WebMethods SoftwareAG

SoftwareAG provides an end-to-end integrated API management toolchain, and supports the API management lifecycle with additional SoftwareAG products.

- ☑ API GATEWAY
- ☑ API PUBLISHER
- ☑ API STORE
- ☑ API DEV PORTAL

FEATURES:
- ☑ WEB CONSOLE
- ☑ PARALLEL VERSION
- ☑ 3RD PARTY GATEWAYS
- ☑ EXTERNAL KEY SYSTEMS
- ☑ PROXY-BASED FORWARDING

OPEN SOURCE
No

HOSTING OPTIONS
On-premise

TWITTER @SAG_API_Mgt

WEBSITE softwareag.com

ESB RESEARCH PARTNER

WSO2 ESB WSO2

WSO2 ESB operates at high performance and has a wide range of Integration capabilities and connectors, including support for EIPs.

DEPENDENCIES
Java

DRAG-N-DROP
- ☑ ROUTE CREATION
- ☑ WEB SERVICES
- ☑ DATA SERVICES

RUNTIME MODE
Standalone

FEATURES:
- ☑ BPEL
- ☑ SERVICE ORCHESTRATION
- ☑ FAILOVER HANDLING
- ☑ FEDERATION
- ☑ WEB DEPLOY CONSOLE
- ☑ SERVICE MOCKING

OPEN SOURCE
Yes

ADAPTERS URL
storepreview.wso2.com/store/

TWITTER @wso2

WEBSITE wso2.com

MESSAGE QUEUE

ZeroMQ iMatix

ZeroMQ is a lightweight toolkit for building distributed systems without needing a central broker, and is optimized for high speed and high volume.

TYPE
Library + bindings

JMS
Non-Java interfaces supported

SPECIALTY
Low-latency P2P

- ☐ TLS ON ALL MESSAGES

FEATURES:
- ☐ JOURNALING
- ☐ TRACING SUPPORT
- ☐ DISK WRITE SUPPORT
- ☑ ALL MSGS IN MEMORY
- ☑ ALL TASKS ASYNC
- ☑ THREAD-TO-THREAD

OPEN SOURCE
Yes

TWITTER @hintjens

WEBSITE zeromq.org

GLOSSARY OF TERMS

A

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.

B

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

D

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).

E

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.

H

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.

I

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.

J

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.

M

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 business feature. The services are independently deployable and communicate with each other through a well- defined interface.

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.

O

OAUTH: A common open standard for authorization.

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

R

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 application programming interface that is said to meet the principles of representational state transfer.

S

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.

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.

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

W

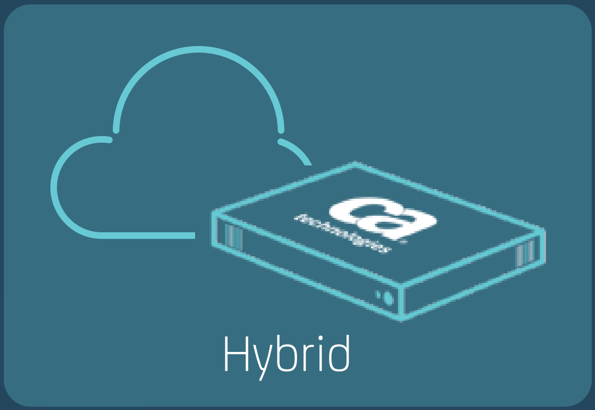
WEB SERVICE: A function that can be accessed over the web in a standardized way using APIs that are accessed via HTTP and executed on a remote system.

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 SECURITY (WSS): A specification that determines how security measures are implemented in web services and SOAP-based messages.

Monetize the Value of Data with CA API Management

Connect every mobile app, cloud service, developer and device



Start your free trial today: http://api.co/DzoneAPI

ca technologies®