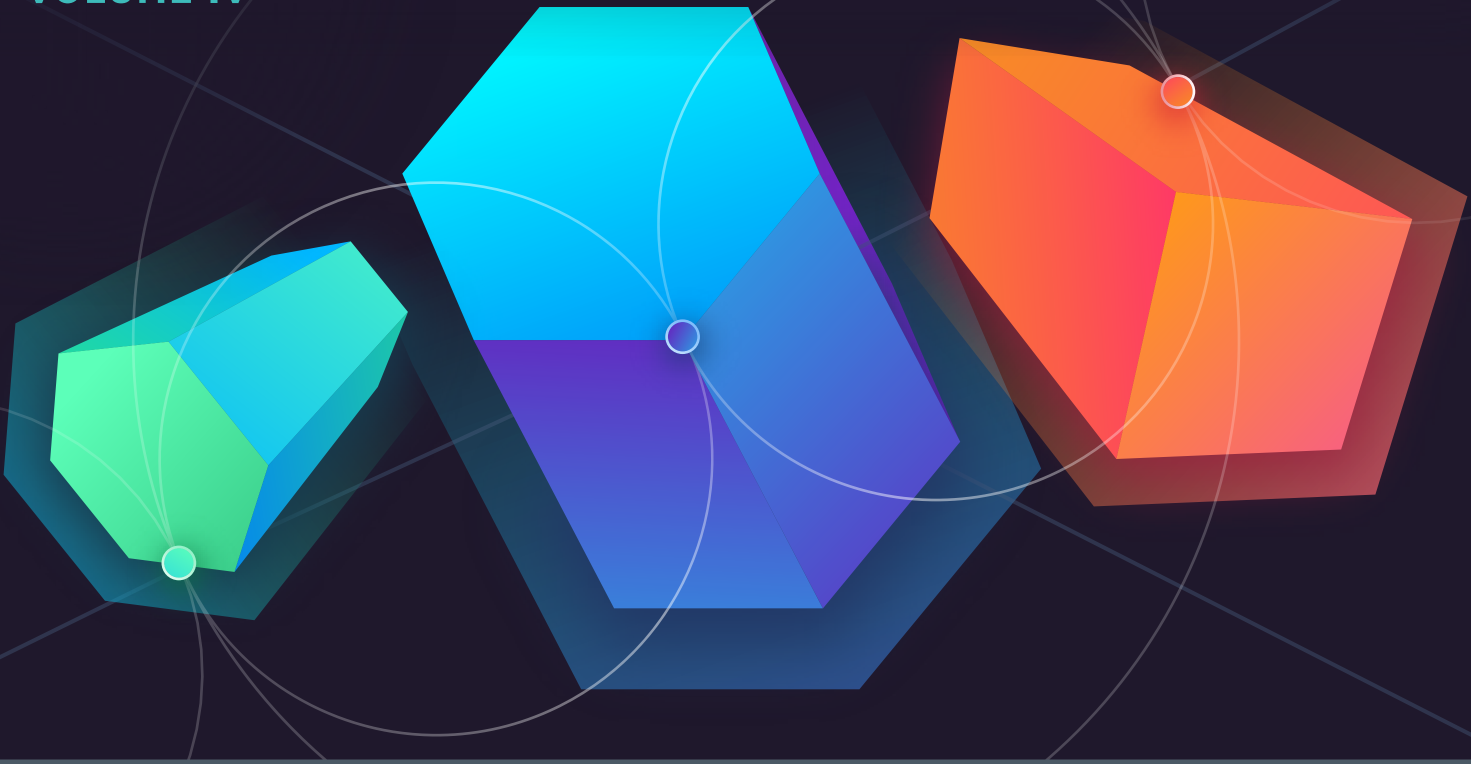
DZone

THE DZONE GUIDE TO

Integration

API Design and Management

VOLUME IV



BROUGHT TO YOU IN PARTNERSHIP WITH

cloud elements

DEAR READER,

Not so long ago I was working on a huge ESB-like project that had only one major responsibility – send a bunch of SOAP requests to other systems and translate their giant XML responses into one huge, more-or-less consistent XML of our own. I never actually checked this myself, but I once heard a manager saying that the project had approximately 2 million lines of code. It often took hours to build and weeks to test it properly. We were supposed to be the masters of integrating systems, yet it was the "integration" part of the problem that was killing us. In the integration jargon, we were a smart pipe; too smart.

Fortunately, times have become much more RESTful since then. The heavy SOAP + XML fad has been washed away by a simpler approach based on HTTP verbs and friendly formats like JSON. I think it's fair to say that REST is past its infancy, as it's being adopted even by the largest and most conservative enterprises, with all their special needs and limitations. On the other side, we've got the continuous rise of microservices architectures, which dramatically and quickly have increased the need for effective integration. These two extremely powerful forces are making us look at the topic of APIs and integration from a completely different angle.

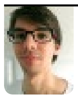
When building and integrating software on such a large scale, simply knowing how to represent your data in a resource-like form and manipulate it using HTTP forms is not going to be enough. Unless you take on the continuous process of API optimization, things can easily get out of hand. We're not talking about toy examples any more. We're talking real business.

Speaking of real business, nowadays more and more developers are taking advantage of Domain-Driven Design practices and patterns. It should be no surprise that this way of tackling the inherent complexity of our most crucial systems is also spilling into APIs that we create. Heck, what would be the advantage of building a powerful, business-aligned service if we surrounded it with a crappy interface that nobody wants to use?

That said, business alignment is only one of the steps towards enterprise-scale integration via APIs. Obviously, there are also other, more technical concerns that require our attention. I will just mention two that you will come across when reading this guide: security and tooling. These are things that are very easy to ignore in the initial stages of APIs adoption and often as hard to introduce later on a company-wide scale.

As you get more experience with APIs and deal with the issues mentioned above, you will find yourself in a position in which you want to level up the game even more. On this route, you will have to face the hard truth that API Design does not equal REST – there's much more to it. At a very deep level, as almost anything in software, API design is not just about technology – it's about people. If there's one thing that you should take out of this guide, let it be that.

Enjoy your read!



BY GRZEGORZ ZIEMOŃSKI

DZONE ZONE LEADER AND JAVA ENGINEER AT ZOOPLUS AG

TABLE OF CONTENTS

PRODUCTION

Chris Smith
DIRECTOR OF PRODUCTION

Andre Powell
SR. PRODUCTION COORDINATOR

G. Ryan Spain
PRODUCTION PUBLICATIONS EDITOR

Ashley Slate
DESIGN DIRECTOR

MARKETING

Kellet Atkinson
DIRECTOR OF MARKETING

Lauren Curatola
MARKETING SPECIALIST

Kristen Pagàn
MARKETING SPECIALIST

Natalie Iannello
MARKETING SPECIALIST

Stewart Cox
PRODUCT MARKETER

Miranda Casey
MARKETING SPECIALIST

Julian Morris
MARKETING SPECIALIST

BUSINESS

Rick Ross
CEO

Matt Schmidt
PRESIDENT

Jesse Davis
EVP

Gordon Cervenka
COO

SALES

Matt O'Brian
DIRECTOR OF BUSINESS DEV.

Alex Crafts
DIRECTOR OF MAJOR ACCOUNTS

Jim Howard
SR ACCOUNT EXECUTIVE

Jim Dyer
ACCOUNT EXECUTIVE

Andrew Barker
ACCOUNT EXECUTIVE

Brian Anderson
ACCOUNT EXECUTIVE

Chris Brumfield
SALES MANAGER

Ana Jones
ACCOUNT MANAGER

Tom Martin
ACCOUNT MANAGER

EDITORIAL

Caitlin Candelmo
DIRECTOR OF CONTENT AND COMMUNITY

Matt Werner
PUBLICATIONS COORDINATOR

Michael Tharrington
CONTENT AND COMMUNITY MANAGER

Mike Gates
SR. CONTENT COORDINATOR

Sarah Davis
CONTENT COORDINATOR

Tom Smith
RESEARCH ANALYST

Jordan Baker
CONTENT COORDINATOR

Anne Marie Glen
CONTENT COORDINATOR

Want your solution to be featured in coming guides?
Please contact research@dzone.com for submission information.

Like to contribute content to coming guides?
Please contact research@dzone.com for consideration.

Interested in becoming a dzone research partner?
Please contact sales@dzone.com for information.

Special thanks to our topic experts, Zone Leaders, trusted DZone Most Valuable Bloggers, and dedicated users for all their help and feedback in making this guide a great success.

Executive Summary

BY MATT WERNER
PUBLICATIONS COORDINATOR, DZONE

Enterprise integration has been a consistent concern for developers everywhere, regardless of industry or application. Nearly every website or application relies on APIs, message queues, or ESBs to provide value to their users. However, just because the field has been present for so long, and so many developers are used to their preferred processes does not mean the industry isn't ripe for major changes. Microservice architectures continue to grow in popularity along with DevOps and container technologies. Tools like Kafka are seeing a meteoric rise as well. How do these trends impact developers working with the latest and greatest in web standards and tools? What about those who work with legacy systems? What kinds of systems are being integrated now?

To find out, DZone surveyed almost 450 developers and tech professionals experienced with integrating systems to share their thoughts on popular tools, architectural styles, communication, and more. We've also provided four articles and a checklist from notable members of the DZone community to explore APIs, design and documentation, tooling, and best practices.

MESSAGING WITH KAFKA

DATA Adoption of Apache Kafka grew from 5% of organizations in 2014 to 13% in 2015, 20% in 2016, and 31% in 2017, making it the second most-popular message queue solution, tied with RabbitMQ, and behind ActiveMQ at 38%.

IMPLICATIONS The growth in Kafka's adoption among enterprises is the most significant trend in the Integration space. At the rate Kafka is growing, it is on track to surpass ActiveMQ to become the most popular measuring tool. This change could happen as early as next year. The main driver of this growth is likely tied to the increased need to harness and effectively utilize big data generated by applications and users.

RECOMMENDATIONS Apache Kafka is a messaging platform for handling real-time data streams, written in Scala and Java. It has become increasingly important for enterprises who need to handle the volume and velocity of incoming data from and between applications. Currently, the major challenge is finding a way to effectively monitor Kafka and Apache Zookeeper, a distributed key-value store used by Kafka, in real-time. There are open source and commercial solutions to solve this challenge. Developers should become more acquainted not just with using Kafka, but also working with other pieces of software that use it. Those who are pursuing microservice architectures should also consider Kafka as a way for microservices to communicate with each other. For more information on trends and the most popular tools outside of Kafka, you can refer to our checklist by Pitor Minkowski on page 15.

NOT SO MICRO ANYMORE

DATA 74% of respondents used microservices in 2017 in some capacity, compared to 67% in 2016. For developers intending to use them, Microservices were the second most used architectural patterns at 53%, tied with MVC-style architectures, with event-driven architectures used the most at 62%.

IMPLICATIONS The use of microservices is rapidly growing as more literature and software emerges to help developers understand, develop, and use them, though the difficulty in setting up microservices and dealing with legacy systems has not allowed them to overtake event-driven architectures yet.

RECOMMENDATIONS Microservice architectures separate components of software into modular pieces, which communicate with each other in order to perform a task. One of the main benefits to using microservices is the ability to deploy changes to only one component of an application, rather than doing a build of an entire application and deploying that. Developers and organizations who are interested in pursuing DevOps methodologies should consider building their applications as microservices to improve the speed at which they can make changes to their software. Despite the hype around microservices, they may not be necessary for your application or your business. Regardless of the application you're currently working on, you should at least become acquainted and familiar with using microservices as more organizations make the transition. Reza Rahman spoke about this in-depth in "Separating Microservices Hype and Reality for Pragmatic Java Developers" in the DZone Guide to Java.

REST IS BEST

DATA 68% of survey respondents currently use REST whenever possible, compared to 62% in 2016.

IMPLICATIONS Systems are becoming more complex, and a major integration challenge has been the need to deal with standards interpretation between systems. REST seems to increasingly be seen as the standard of choice for developers whenever possible. Most respondents who do not use REST whenever possible do not do so due to clients using SOAP or because they have to deal with legacy systems.

RECOMMENDATIONS At this point, REST is by far the leader in setting interoperable web standards. Investing in RESTful APIs is a wise investment as this standard increases in popularity. However, as RESTful APIs are adopted, security is a major step in the design process that should not be ignored. You can read some tips from DZone's John Vester on page 12. It's also worth keeping in mind that REST is not the only thing to account for when designing your APIs, as API Evangelist Kin Lane discusses in his article "API Design Does Not Equal REST" on page 16.

Key Research Findings

BY G. RYAN SPAIN
PRODUCTION COORDINATOR, DZONE

We received 440 complete responses and 625 partial responses to DZone's 2017 Integration survey. Demographics for complete survey responses include:

- 33% of respondents identify as developers or engineers; 26% identify as architects; and 22% identify as developer team leads.
- The average respondent has 14.2 years of experience as an IT professional. 62% of respondents have 10 years of experience or more; 17% have 20 years or more.
- 38% of respondents work at companies headquartered in Europe; 30% work in companies headquartered in North America.
- 23% of respondents work at organizations with more than 10,000 employees; 23% work at organizations between 1,000 and 10,000 employees; and 21% work at organizations between 100 and 1,000 employees.
- 85% develop web applications or services; 53% develop enterprise business apps; and 30% develop native mobile applications.

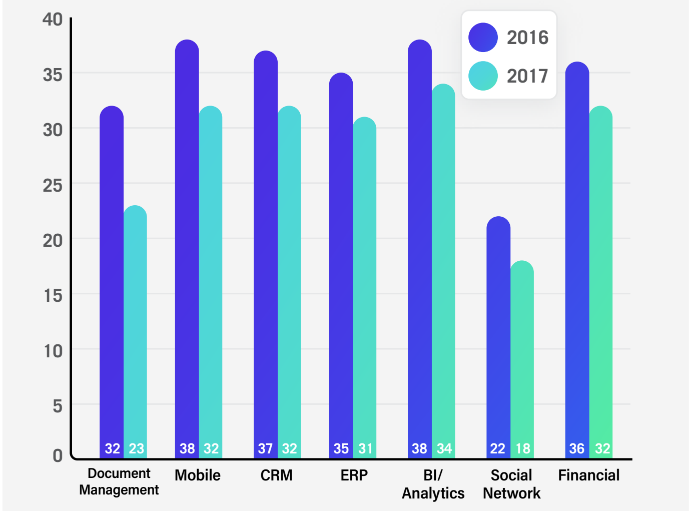
INTEGRATION DECLINATION

Our integration survey asked respondents whether they integrate 16 discrete types of systems. This year's survey saw a decline in responses for 15 of them (IoT was the only system type that showed an increase, though this increase was within the margin of error for the question*). Significant changes (those outside of the margin of error) were seen in the following integrated systems: document management, a 9% decrease; mobile, a 6% decrease; CRM, a 5% decrease; ERP, a 4% decrease; BI/Analytics, a 4% decrease; social networks, a 4% decrease; and financial, a 4% decrease. As far as tools and technologies used for integration, this year's results showed little variance from the results in 2016, something we noticed in our findings last year as well. Still, there were enough shifts in tool usage and integration practices to show that, while integration development is stable enough to avoid many dramatic shake-ups, it still has room for evolution and maturation.

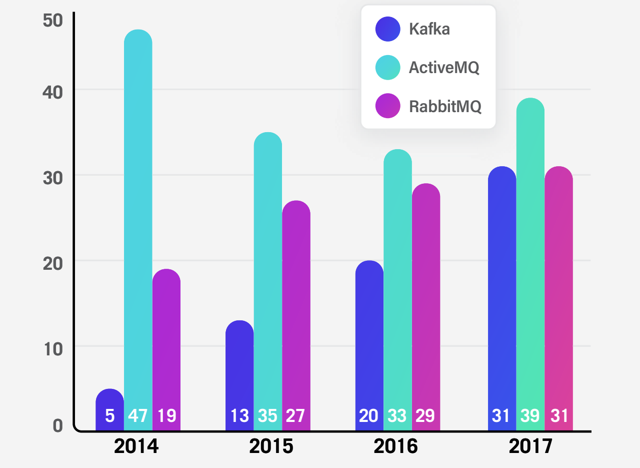
KAFKA'S METAMORPHOSIS

Among mostly ripples of change in the integration technology waters, Kafka has been making waves. Graduating from the Apache Incubator in late 2012,

What types of systems do you integrate?



Does your organization use any of the following messaging tools?



Kafka adoption has grown tremendously in a short time, and is now one of the dominant messaging tools in the integration landscape. In 2014, 5% of our integration survey respondents said their organization used Kafka. This increased in 2015 to 13%, and increased again in 2016 to 20.3%. This year, 31% of respondents said their organization uses Kafka, tying it with RabbitMQ, which has seen much slower adoption growth, for second place among messaging tools. The current adoption pace for Kafka indicates the possibility that in the near future, Kafka will be the foremost messaging tool for integration projects, and Kafka and Zookeeper performance monitoring solutions will become increasingly vital.

YEAH, YAML!

Little has changed from last year regarding the two kings of data serialization, JSON and XML. These two formats are nearly universally used, with 97% of respondents saying they use JSON (and 90% saying they enjoy it), and with 93% of respondents saying they use XML (though only 47% enjoy it). YAML, however, has seen steady growth over the last 2 years. Respondents who said they use YAML grew from 34% in 2015 to 39% in 2016, and this year 46% of respondents say they use this format for data serialization and interchange. While YAML is unlikely to ever overtake JSON or XML for use in RESTful API data interchange due to things like speed and interoperability, its steady growth shows experimentation and utilization of different data formats for different purposes.

MICRO-MANIA

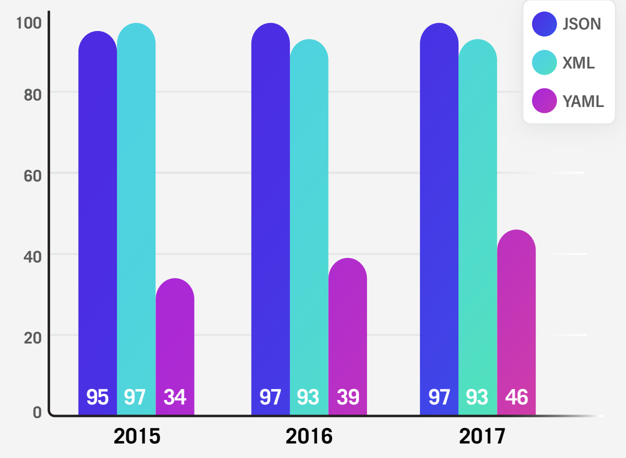
Adoption of microservices architectures has also been on the rise, and this architectural style is becoming commonplace in software development. Microservices use grew from 67% in 2016 to 74% in 2017, and while it remains in fourth place out of six architectural styles our survey asks about, microservices has become tied for second place among intentionally used architectural styles; after event-driven architectures, which 62% of respondents said they use intentionally, microservices and MVC-style architectures had 53% responses for intentional use, followed closely by n-tier/layered architectures at 52%. Advantages of microservices over more monolithic architectures, such as fault isolation, scalability, and deployment speed, along with the rise of microservices-friendly technologies like containers, continue to drive microservices popularity.

GETTING SOME REST

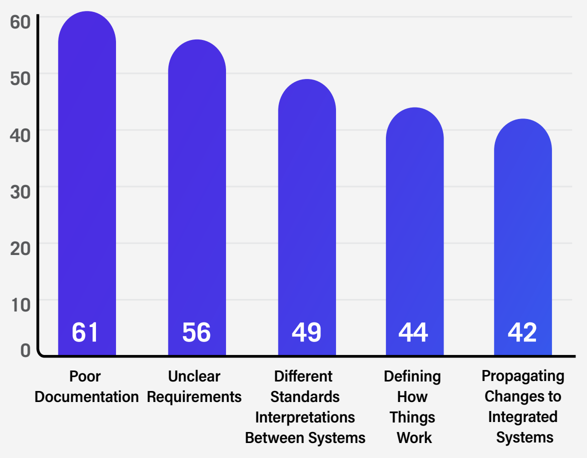
Use of REST practices is becoming more prevalent as API standards become more important. 68% of respondents in this year's survey said they use REST whenever they can, compared to 62% in 2016 and 60% in 2015. Considering that the third most common integration challenge respondents had was dealing with different standards interpretations among systems, it makes sense that more developers are trying to introduce more standardization in the way their APIs communicate. In fact, the responses from those who said they do not use REST for specific reasons mostly focused on RESTful APIs being difficult or impossible to communicate with legacy or SOAP-based systems. The rapid increase in the number of interactions between applications and systems puts more and more pressure to create the kind of loosely coupled architectures REST APIs can provide.

**Margin of error calculated with 95% confidence interval*

Which of the following data serialization & interchange formats do you use?



What are your organization's biggest integration difficulties?



Documentation-Driven API Design

BY **GUY LEVIN**
CTO, **RESTCASE**

QUICK VIEW

01 Documentation-Driven API Design is a cycle of writing docs, writing tests according to the docs, writing code to pass the tests, refactoring, and repeating.

02 Documentation changes are cheap; code changes are expensive.

03 If a feature is not documented, then it does not exist, and if a feature is documented incorrectly, then it is broken. Think of planning your documentation first as a way of creating more time to work on the main project.

Many companies and developers think documentation is something to be done as an afterthought, a nightmare for any developer who has been assigned the difficult task of having to go through each of the application's features and write about them and how they are used.

But think, what if you could speed up your API development by focusing on the documentation first? Documentation changes are cheap, code changes are expensive.

The philosophy behind Documentation-Driven API Design or Development is simple: from the perspective of a user, if a feature is not documented, then it does not exist, and if a feature is documented incorrectly, then it is broken.

API DESIGN – DOCUMENTATION FIRST

Designing an API is a special case because you're often designing for unknowns. You might not know:

- What types of clients will consume the API, or their individual preferences.
- The flow that any given consumer will take through your API endpoints.
- Which information will be most valuable to the consumer.

With all of these unknowns, the best thing you can do for yourself and your consumers is to write your documentation first, as visibly as possible. The goal is to gather insight and recommendations from your consumers early and throughout the design process, turning your unknowns into knowns.

By designing your API through documentation, you can easily get feedback and iterate your design before any development happens. Changes are easier and faster to make in documentation than they are in code. Documentation can become an invaluable resource as you want to increase your company's Velocity.

The documentation gives the client team and test team something to work from before the server team has even implemented the documented endpoints. This prevents rework and parallelizes the efforts of the server and client teams. Coders and testers could start working at the same time to implement and test such an API.

When writing or updating documentation is a required first step of your development process, you ensure your documentation is always consistent with your code.

6 GUIDELINES FOR WORKING DOCUMENTATION-DRIVEN API DESIGN

1. DOCUMENT YOUR API OR FEATURE FIRST

You can do this as you design an API, or later if you'd like to rework an existing API. Wherever the documentation becomes complicated or difficult to write, revisit the design. This process works because it is easier to spot complexity in the documentation than in the code. Figure out how you are going to describe the feature to users; if it is not documented, it does not exist.

DO DOCUMENTATION REVIEWS
Whenever possible, documentation should be reviewed by users before any development begins. This also gives you the chance to get your ideas peer-reviewed, since it helps users to understand what you are trying to do.

3. WORK IN PARALLEL
Once documentation has been written, development should commence, preferably test-driven development. Both developers and testers can start working on the implementation, one focusing on writing code and the other on writing automatic tests.

4. TESTING
Unit tests should be written to test the features as described in the documentation. If the functionality is ever out of alignment with the documentation, tests should fail.

5. CHANGES
When a feature is being modified, it should be modified documentation-first; when documentation is modified, so should the tests change. Along with the tests being changed, the coding will also have to be modified accordingly.

6. VERSIONING
Documentation and software should both be versioned, and the versions should match so someone working with old versions of software will be able to find the proper documentation.

API DOCUMENTATION DESIGN FUNDAMENTALS
Concise and clear documentation, which allows your API consumers to adopt it into their application quickly, is no longer optional for organizations that want to drive adoption of their APIs. According to SmartBear's 2016 State of API Report, 75% of organizations that develop APIs now have a formal documentation process. 46% say it is a high priority for their organization. A survey by ProgrammableWeb found that API consumers considered complete and accurate documentation as the biggest factor affecting their API decision-making, even outweighing price and API performance.

Good documentation accelerates development and consumption, and reduces the money and time that would otherwise be spent answering support calls. It is important to remember that documentation matters for internal API users as well. Do not assume that your internal stakeholders will have intimate knowledge of how to work with the API.

What should go into your API documentation? These are three features that should be ever-present in documentation:

1. There should be consistency, meaning that an end user should be able to know what your documentation has in the offing. The terminology used should also be in line with your language.

2. The documentation should be a complete scope of the features of your entire project. Private methods used should also be put down for your developers to utilize. The public features should be made explicitly available for your end users.

3. It should also be up to date. This means that the documentation should maintain recent versions of the code used for the project.

There are, of course, no standards or hard-and-fast rules on what API documentation should have. As a general rule, whatever resonates with the developer community and makes it easy for them to understand an API is a good starting point.

The philosophy behind Documentation-Driven API Design or Development is simple: from the perspective of a user, if a feature is not documented, then it does not exist, and if a feature is documented incorrectly, then it is broken.

Some general sections that are backed up by examples from established providers in the API economy:

- A list of the resources with an explanation of the purpose of each in the context of the product or service being offered via the API.

- Examples of API calls in a variety of languages and tools (curl, Postman collections, etc.). The examples are probably the most important section of the API document, as these are going to be used by your clients.

- Guides that detail the workflows implicit in using the API, i.e. the sequence of API calls that do something meaningful in the context of the product or service the API offers.

- An overview of the design principles adopted by the API provider and what that means for aspects adherence to REST (especially hypermedia), HTTP codes, etc.

- Information on authentication, including schemes that may be implemented, such as OAuth or OpenID Connect.
- General information on error handling with information on the HTTP return codes that will be returned.

OVERVIEW

- Why the platform should be used
- Architecture

GETTING STARTED

- Tutorials
- Walk-throughs

SAMPLE CODE

- Clarity
- Simplicity

REFERENCES

- Description
- Parameters
- Remarks

OPEN API SPECIFICATION, A.K.A. SWAGGER

Some API documentation formats have the added benefit of being machine-readable. This opens the door to a multitude of additional tools that can help during the entire lifecycle of your API:

- Create a mock server to help during the initial API development for both testers and client/server-side teams, and to test your API before deployment to ensure that the API and the documentation match.
- Create interactive documentation that allows developers to perform demo requests to your API. These interactivity features will serve as an invaluable tool for both the developer and the debugger.

Among all the API documentation formats, the most popular and used by many big companies is the OAS (Open API Specification), or Swagger. It gives you the ability to design your API in a way that can be easily consumed by humans as well as machines.

The Open API/Swagger specification is a machine- and human-readable description format that defines the API's contract. This contract defines what data is exposed by the resources and services, and how the client should call these resources. While this idea seems simple, it has powerful implications for the multi-platform API economy, where services are built in many languages and consumed by clients across different devices.

Today, thousands of API teams around the world use the Open API Specification and Swagger tooling to generate documentation for their internal and public-facing APIs.

There are many practical uses for implementing the OAS into your API workflow starting from code generation (try out ApiBldr, a Visual Open API Builder that helps you build Open API Specification for your API. It's also able to generate client and server SDKs from the OAS) and ending with generation of beautiful interactive documentation that can easily be shared with the API's end users.

Concise and clear documentation, which allows your API consumers to adopt it into their application quickly, is no longer optional for organizations that want to drive adoption of their APIs.

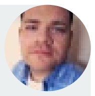
CONCLUSION

API documentation, as detailed above, is no easy task. Organizations not only need to work on technical writing, but also must make sure the documentation is secure and easy to work with.

Everyone agrees that documentation is an absolute must if you want to guarantee that your API is well understood by potential consumers and business partners. While some people believe that starting an API project with initial documentation is a good idea, most people struggle to actually write something.

After you finish your documentation, you can quickly generate the code for your API using the many tools out there, like ApiBldr, for example.

Guy Levin is the CTO of RestCase, a startup company that develops a REST API Development Platform to help software companies and developers speed up development of RESTful services, minimizing time-to-market and raising quality. Guy is an architect focusing on distributed and cloud systems, a full stack software engineer, and a DBA. He has over 20 years of experience in software development in a variety of sectors, including medical and healthcare, cyber security, and fintech.



Diving Deeper

INTO INTEGRATION

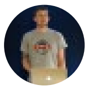
TOP #INTEGRATION TWITTER ACCOUNTS

@samnewman

@apireport

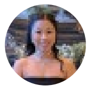
@launchany

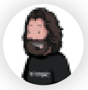
@crichardson


@dunglas

@izuzak

@mattbiehl


@kathykam


@kinlane

@rjrodger

INTEGRATION-RELATED ZONES

Integration dzone.com/integration
The Integration Zone focuses on communication architectures, message brokers, enterprise applications, ESBs, integration protocols, web services, service-oriented architecture (SOA), message-oriented middleware (MOM), and API management.

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

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

TOP INTEGRATION REFCARDZ

Continuous Integration
This newly updated Refcard explains patterns and anti-patterns and the benefits of CI. Using specific examples from the Git Command Line Interface and Git Plugin hooks for Jenkins, it also walks through build management, build configuration, testing, and code quality.

RESTful API Lifecycle Management
In this Refcard, familiarize yourself with the benefits of a managed API lifecycle and walk through specific examples of using RAML to design your API.

Getting Started With Microservices
In this updated Refcard, you will learn why microservices are becoming the cornerstone of modern architecture, how to begin refactoring your monolithic application, and common patterns to help you get started.

TOP INTEGRATION WEBSITES

The Largest API Directory on the Web
programmableweb.com/apis/directory

microservices.io
microservices.io

Enterprise Integration Patterns
enterpriseintegrationpatterns.com

TOP INTEGRATION BOOKS

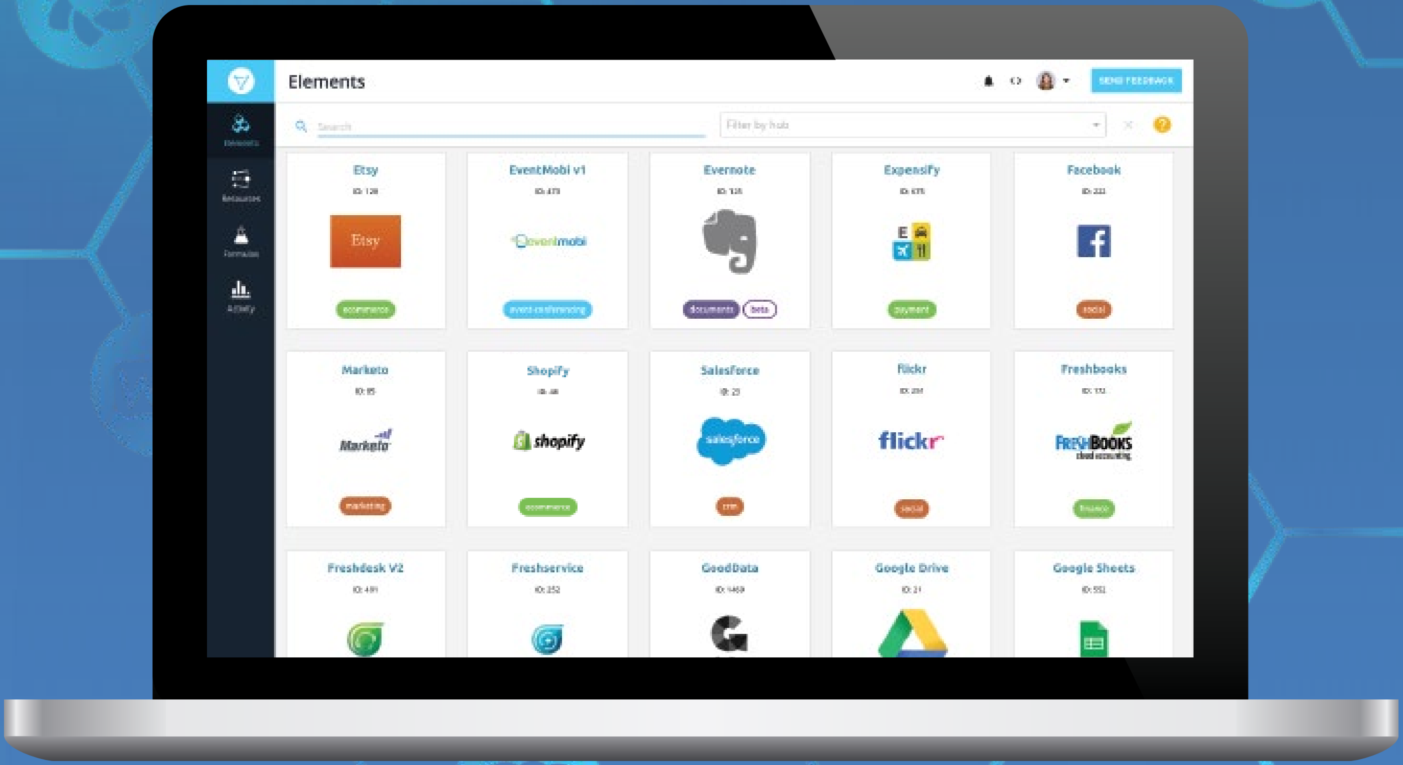
Building Microservices: Designing Fine-Grained Systems
Author Sam Newman provides you with a firm grounding in the concepts while diving into current solutions for modeling, integrating, testing, deploying, and monitoring your own autonomous services.

Undisturbed REST: A Guide to Designing the Perfect API
After reading Undisturbed REST, you'll have a strong understanding of APIs, best practices, and available tooling for designing, prototyping, sharing, documenting, and generating tooling (such as SDKs) around your API.

The Tao of Microservices
This book teaches you the path to understanding how to apply microservices architecture with your own real-world projects. This high-level book offers you a conceptual view of microservice architectures, along with core concepts and their application.

cloud elements

Outthink. Outbuild. Out Integrate Them All.



Create the ultimate integration experience without your customers even knowing we're there.

Go beyond connecting. INTEGRATE.

LEARN MORE

Turning Products into Platforms: Integration as a Differentiator

Today's most successful software companies have made the leap from selling products to selling a platform. The appeal of such a move is understandable, since products generate only a single revenue stream, while platforms - which connect different groups of users and services - can generate several, and ensure your product can participate in (or enable) a broader app ecosystem.

Your product's APIs provide a first step towards building a platform, by allowing developers and partners to easily do business with you. But a platform strategy requires more than an API. You must shift the burden of integration away from customers and developers, and provide embedded integration within your product. Stop forcing your customers to solve integration problems for themselves.

However, it's important to realize that building integrations into your product isn't easy. And in many cases, technology solutions offer only basic connectivity between apps with simple pre-built recipes for data movement, not robust and flexible integrations that most customers will demand. You need to consider the data your customers need to interact with. In marketing and CRM apps for example, most users have custom data, unique to their business, and this must be factored in when offering integration solutions between your platform and these other apps.

A platform strategy requires more than an API. You must shift the burden of integration away from customers and developers.

In short, to become a platform you must make it not just possible, but simple to integrate with the other products and services in their ecosystem seamlessly. Integration options must be provided as an embedded part of your product experience, creating differentiation and happier, stickier customer relationships.

WRITTEN BY ROSS GARRETT
HEAD OF PRODUCT MARKETING, **CLOUD ELEMENTS**

PARTNER SPOTLIGHT

API Integration Platform By Cloud Elements

Cloud Elements is purpose built to connect SaaS and the Digital Enterprise with the applications used by your customers and partners.

CATEGORY	NEW RELEASES	OPEN SOURCE
API Integration	Every two weeks	Yes

CASE STUDY
Influitive, the leading Advocate Marketing platform, had a mission to take the burden of building integrations for their product off their customers' shoulders. However, their team was frustrated when they realized the amount of time required to scope, plan, develop, test, deploy, support, and maintain each of the integrations on their roadmap. Aaron Rothschild, Director of Product Management at Influitive, discovered Cloud Elements and realized the work to build these integrations had already been done. "With Cloud Elements, we were moving 10x faster and released three more integrations in less than two weeks," shares Rothschild. On top of a faster time-to-market, Influitive was able to avoid many of the risks of building and maintaining the code themselves.

STRENGTHS

- Our 125+ pre-built **Elements** offer more functionality and usability than other connectors on the market.
- Our **Hubs** are designed to provide "one-to-many" integration across categories of API providers.
- Our **Common Object Model** allows you to focus on the data you care about, instead of the unique interfaces of your apps and services.
- Our entire platform is **API-Enabled**, making it possible to fully embed integration logic directly into your app.

NOTABLE CUSTOMERS

- Sage
- Concur
- GoToWebinar
- Sprinklr
- Avalara
- Act-On
- Inspirato with American Express
- Nintex
- Totaljobs Group

WEBSITE www.cloud-elements.com **TWITTER** @cloudelements **BLOG** blog.cloud-elements.com

RESTful API Security

QUICK VIEW

01 Security should always be a high priority when designing a RESTful API — an identity provider and API key functionality are recommended.

02 Non-production APIs should be treated as production APIs to minimize risk

03 Input validation should be included in the RESTful API

04 Use of HTTP response codes are recommended

BY **JOHN VESTER**
SR. ARCHITECT, **CLEANSLATE TECHNOLOGY GROUP**

API DESIGN – DOCUMENTATION FIRST

According to TechTarget, a RESTful API is "an application program interface (API) that uses HTTP requests to GET, PUT, POST, and DELETE data. A RESTful API — also referred to as a RESTful web service — is based on representational state transfer (REST) technology, an architectural style and approach to communications often used in web services development."

With the explosive growth of RESTful APIs, the security layer is often the one that is most overlooked in the architectural design of the API.

WHY DOES SECURITY MATTER?

There are three core reasons why security should be a serious consideration when designing and deploying a RESTful API.

DATA PROTECTION

A RESTful API is the way in which a given service can present value to the world. As a result, protection of the data provided via RESTful endpoints should always be a high priority.

DOS ATTACKS

Denial of Service (DOS) attacks can render a RESTful API into a non-functional state if the right security measures are not taken. Consider the case where an entry-level RESTful API is provided for public consumption. While it's often a great marketing idea to generate usage of your API, it can also lead to catastrophic results when a consumer opts to perform DOS attacks on the API.

ANTI-FARMING

Today, there are several marketing-heavy websites that offer consumers the best deal on everything from flights to vehicles and even groceries. In many of these cases, the aggregated service is taking advantage of other APIs to obtain the information they want you to utilize. When this happens, the RESTful API is being farmed out for the benefit of another entity.

SECURITY ADOPTION MEASURES

Below are some key concepts that should be part of any RESTful API design.

SESSION MANAGEMENT AND AUTHENTICATION

Aside from use of TLS/HTTPS, the most important level of RESTful API security is centered around session management and authentication. For this article, the focus is going to be on API keys, OpenID Connect/OAuth2/SAML, and session state considerations.

API KEYS

The concept of API keys provides the consumer of the endpoint(s) with a unique string (a key) that is part of their HTTP requests. While not a complete security solution, the use of API keys yields a higher level of knowing who is using the API when compared to anonymous usage.

API keys can also be used to provide additional services. As an example, the service may opt to have Silver, Gold, and Platinum levels for the RESTful API. At the Silver level, end-users are granted free access, but only to a limited set of endpoints. Gold and Platinum would require some level of payment, with the Platinum level providing more benefits over the Gold option.

The preferred manner to use API keys is to include them in the request header. As an example, the API key of 67a73dde1bdd1d90210ba is set to the X-API-KEY key/value pair in the HTTP header when calling the widgets end-point:

```
curl -H “X-API-KEY: 67a73dde1bdd1d90210ba”
https://www.example.com/v1/widgets
```

Another common use of an API key is to include the key in the URI itself:

```
https://www.example.com/v1/widgets?api_key
=67a73dde1bdd1d90210ba
```

The problem with this approach is that the API key is revealed in both the browser history and in logs at the API server — exposing the unique key to all who have access to these elements.

OPENID CONNECT, OAUTH2, AND SAML

OpenID Connect, OAuth2, and SAML use HTTP protocols as a transport and are used for security purposes. Authentication provides the verification of the individual, while authorization performs the task of granting or revoking access.

From an authentication perspective, the following options exist:

- **OpenID Connect**: Can leverage an existing identity provider (like Google or Facebook) to obtain a token used to validate the end-user's credentials.
- **OAuth2**: Can perform pseudo-authentication through the authorization process through delegation (explained below).
- **SAML**: Uses assertions, protocols, bindings, and profiles to manage authentication, including an identity provider, but is not well-suited for mobile applications.

To provide authorization services, the following strategies are employed:

- **OAuth2**: Provides secure delegated access, allowing actions to be taken on behalf of the user by allowing tokens to be issued by a third-party identity provider. Since OAuth2 must know about the delegated user, authentication is achieved in a pseudo fashion (as noted above).
- **SAML**: Performs assertions to trusted services, including a token to provide the authorization.

SESSION STATE CONSIDERATIONS

RESTful API endpoints should always maintain a stateless session state, meaning everything about the session must be held at the client. Each request from the client must contain all the necessary information for the server to understand the request. In order to streamline the process, the API token along with a session token can be included to provide a “session blob” as part of each transaction.

ACCESS CONTROL

As noted above, authorization for RESTful services can introduce security into provided end-points so that there are limits for those who can issue a HTTP DELETE request to the API.

In the simple Java example below, only members of the Admin and Manager roles (i.e. groups) can perform the DELETE request to delete all users, but all users can perform the GET request to obtain a list of users:

```
@Path(“/”)
@PermitAll
public class userExample {
   @DELETE
   @Path(“users”)
   @Produces(“text/plain”)
   @RolesAllowed({“Admin”, “Manager”})
   public String deleteAllUsers() {
      return userService.deleteAllUsers();
   }

   @GET
   @Path(“users”)
   @Produces(“text/plain”)
   public String getAllUsers() {
      return userService.getAllUsers();
   }
}
```

RATE LIMITS

As noted above, an API key is a valuable strategy to provide a level of identity to consumers of a RESTful API. In addition to providing tiered services, another benefit of using API key is the ability to throttle usage of the API.

API management solutions like TIBCO Mashery, Mulesoft, and Dell Boomi allow the ability to throttle API requests, leveraging API key for this functionality. As a result, a consumer who attempts to perform a DOS attack (intentionally or unintentionally) will reach a set threshold at which time all future requests will be rejected.

RESTful API endpoints should always maintain a stateless session state, meaning everything about the session must be held at the client.

INPUT VALIDATION AND HTTP RETURN CODES

Input validation should always be a consideration when securing RESTful APIs. While the client calling the end-point may be handling validation, the assumption cannot always be relied upon with RESTful APIs.

As an example, if the end-user is attempting to POST data a collection of JSON data related to an address, the service within the RESTful end-point should validate the data and use HTTP return codes to reflect the correct status. In the simplified Java example, Jersey and Jackson are used to call a very basic addressService to both validate and persist the address:

```
public class addressExample {
   @POST
   @Path("/address")
   @Consumes("application/json")
   @Produces("application/json")
   public Address createAddress(Address newAddress) {
      if (!addressService.isValidAddress(newAddress))
{
              return Response
             .status(Response.Status.BAD_REQUEST)
             .entity("Invalid address provided.")
             .type(MediaType.APPLICATION_JSON)
             .build();
      } else {
              String jsonAddress = addressService.
convertAddress(newAddress);
              return Response
                      .status(Response.Status.CREATED)
                      .entity(jsonAddress)
                      .build();
      }
   }
}
```

In the example above, the newAddress object (marshalled from JSON to the Address Java object) is validated using the isValidAddress() method. If the address is not valid, an HTTP 401 (Bad Request) code is returned to the user with the text "Invalid address provided." If the address is considered valid, the convertAddress() will perform the necessary actions, then return a String-based JSON rending of the Address object to be included back to the user, along with a HTTP 201 (Created) return code.

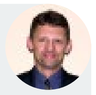
The risks associated of not protecting sensitive data, allowing DOS attacks, and not considering farming usage of RESTful APIs could easily put a business at a disadvantage, if not damage the business permanently.

CONCLUSION

Protecting RESTful APIs should always be at the forefront of the API design effort. The risks associated not protecting sensitive data, allowing DOS attacks, and not considering farming usage of RESTful APIs could easily put a business at a disadvantage, if not damage the business permanently.

Authorization and authentication can provide the necessary security to RESTful APIs, with implementation of an API key strategy adding significant value — with a low cost to implement.

Input validation should always be a part of the RESTful API since there is no guarantee the consumer of the API will be performing any necessary validation. Communication back to the calling client should implement HTTP return codes beyond simply success (200) and error (404).



john Vester is an Information Technology professional with 25+ years expertise in application development, project management, system administration and team management. Currently focusing on enterprise architecture/application design/continuous delivery utilizing object-oriented programming languages and frameworks. Prior expertise building Java-based APIs against React and Angular client frameworks. CRM design, customization and integration with Salesforce. Additional experience using both C# (.NET Framework) and J2EE (including Spring MVC, JBoss Seam, Struts Tiles, JBoss Hibernate, Spring JDBC).

New Trends and The Most Popular Integration Tools

BY PIOTR MINKOWSKI

IT ARCHITECT AT P4

Today's IT world is changing quickly. New popular trends in software development, frameworks, and tools are arising almost all the time. When developing applications, we always face the choice of the right architecture to meet our needs. Our applications are not always separate, independent entities. Usually they are exchanging data and messages with each other using different communication patterns. In this article, I will try to show the latest trends in the IT world and to analyze the most common uses of popular technologies.

1. Hybrid Integration Platforms

Hybrid integration platforms combine on-premises solutions like ESBs or ETL and cloud platforms like iPaaS/iSaaS or API Management. The hybrid approach to integration supports the full spectrum of application, data, and process integration. Organizations must deal with varying requirements, different partners, and different levels of technical knowledge. The answer to these challenges cannot only be one particular integration technology, although Integration Platforms-as-a-Service are becoming a mainstream way of connecting mobile, SaaS, IoT, Big Data, and on-premise Line of Business (LOB) systems. Hybrid integration allows on-premise applications to integrate with cloud based applications.

2. Serverless

A solution that does not require developers to manage the application and its configuration. In return, it hosts small and agile snippets of code that do this job for us. A whole backend can be implemented as a collection of functions running in the cloud. It is closely related to the concept of Function as a Service (FaaS). Currently, most of the major cloud providers have a suite of cloud computing services like AWS Lambda, Google Cloud Functions, or Azure Functions. The main advantages of this approach are the ability to scale according to demand and to only pay for resources used.

3. Microservices

Has anyone heard of microservices yet? This is still a hot trend in the IT world. This is a model of single-purpose, independently deployable services, usually separated as a group from the monolithic apps. Just like serverless apps, they are based on the concept of the division of the source code into smaller pieces. But the difference is in size: serverless is often referred to as nanoservices in that it can be shared in the on-premise architecture with such tools like Docker, Kubernetes, Pivotal Cloud Foundry, or Spring Cloud.

4. Reactive Services

Reactive programming is an asynchronous programming paradigm that focuses on building decoupled scalable and resilient services. The main assumptions of this approach to service creation are described in The Reactive Manifesto. They are the answer to the latency problem with synchronous microservices over REST protocol. Reactive APIs makes it easy to call other services asynchronously and in parallel.

Integration Frameworks

No matter which architecture you choose, you must think about mediation and orchestration layers. Here's a list of the most popular integration frameworks implementing EIP patterns:

1. Apache Camel

Camel offers many components for most of the popular technologies out there. It has DSL support for Java, Groovy, and Scala. It is used as a mediation layer in ESBs like Talend or JBoss Fuse.

2. Spring Integration

Spring Integration is based on the well-known Spring project and extends the programming model with integration support. You can use Spring features such as dependency injection, transactions, or security as you do in other Spring projects.

3. Mule ESB

It is a full ESB, including several additional features instead of just an integration framework. Mule ESB can be use as a lightweight integration framework, too, by not adding or using any additional features besides the EIP integration components.

Messaging Styles and Protocols

1. SOAP

SOAP is still very popular, especially in the complex business systems where architecture is based on ESBs, integration suites, or groups of large monolithic applications talking to each other. It always uses XML to encode calls, depends on WSDL, and does not have any standardized mechanism for dynamic discovery of services. Well-known frameworks for SOAP Web Services are Apache CXF and Axis. SOAP is also strongly supported by JavaEE solutions.

2. REST

REST can use various notations for call encoding. The most popular is JSON, but you can use other notations like Google's Protocol Buffers for performance reasons. It's most commonly used for small systems sharing APIs or a large system divided into many small microservices. If you are creating REST APIs, you should definitely look at the Jersey or Spring Web frameworks.

3. Websockets

According to some studies, Websockets is three times faster than HTTP. It is best for web applications where the client and server need to exchange data at high frequency and with low latency, such as in real-time chat applications.

4. Message Brokers

Based on the publish-subscribe model and epecially useful for event-based systems where one message needs to be delivered to several systems or you need to perform some background work with no need for real-time responses. Message brokers are great at ingesting and routing lots of data. The most popular message brokers are RabbitMQ, ActiveMQ, and Apache Kafka. Apache Kafka is designed for high volume publish-subscribe messages and streams, meant to be durable, fast, and scalable. RabbitMQ's strengths are consistency, security, and easy monitoring with web UI.

5. Reactive

Reactive is a natural fit for building resilient microservices, especially if you have a lot of calls between your microservices, and latency can be a problem. There are some interesting tools and framework like Akka, RxJava, Reactor, Vert.x, or Spring Framework 5.0.

API Design Does Not Equal REST

QUICK VIEW

01 API design does not equal REST

02 Web standards should drive API design

03 API design impacts the entire API lifecycle

04 Communication is essential to API design

05 API design is your business' API contract

BY **KIN LANE**
API EVANGELIST

In the early days of web APIs (2000-2010), the concept of API design was very much associated with Roy Fielding's dissertation on Architectural Styles and the Design of Network-based Software Architectures. If you were talking about API design, all conversations quickly became about using HTTP, crafting your URIs, and using your HTTP verbs — early RESTafarians made sure of this.

HYPERMEDIA DESIGN PATTERNS

Alongside this discussion that put HTTP at the center the API design conversation, a small band of API architects were pushing forward how we craft the requests and responses of our APIs, something that was modeled after the web and was called hypermedia. This new breed of API craftsperson focused on applying RESTful design principles while also deeply considering the structure and relationships between the resources they were making available and heavily considering how these resources would be consumed at the client level.

Even with these strong opinions regarding the technical details of an API and the careful crafting of your API requests and responses, the conversations were primarily within an exclusive group of developers. The wider business and enterprise developer groups were often unaware of the nuances of design and would need more documentation to be able to understand and put API design practices to work in their world. A new breed of API documentation tool called Swagger would soon emerge, which would help introduce developers to the world of API design through their need for up-to-date API documentation.

DEFINING COMMON DESIGN PATTERNS

In 2012, as the API space was picking up steam, the API definition format, formerly Swagger (now OpenAPI), was helping to standardize how we communicate around APIs. But another API service provider emerged with a new approach to defining APIs that was focused on design over just producing documentation. Apiary came onto the scene with their new way of defining APIs called API Blueprint, which emphasized design over mocking, testing, documentation, or even the deployment of your API. This new approach to designing APIs before you ever got dirty with the hard work of development would set into motion a new age of API design where it was becoming more than just REST or hypermedia but was also quickly becoming about communicating, collaborating, and sharing your API design throughout the API lifecycle.

Swagger and API Blueprint cracked open a new world of API design, enabling a much more standardized and shared conversation to occur amongst developers about what API design is beyond just REST and hypermedia. Honestly, I don't feel that most developers are even ready for hypermedia, and API definition formats are allowing for some critical web literacy to be taught to large groups of developers, preparing everyone for a hypermedia engineered future. We now had a common language for defining, collaborating, and sharing around common API design practices, setting the stage for growth beyond just the number of APIs, including what we expected of our APIs and ultimately the design constraints we were applying (or not).

SERVING THE FULL API LIFECYCLE

API definitions were being used to design APIs, potentially considering other existing API design patterns, all captured in a way that could be shared with other stakeholders, evolving the API design process into a group effort. The resulting API definition could then be used as a central truth to mock, deploy, manage, test, monitor, discover, and generate code. This new approach to developing software is not just driving web and mobile applications but also a growing number of devices across our personal and professional lives. API design was no longer about a single set of design patterns and could now encompass a wider practice and discipline, opening up the door to new ways of thinking.

RUNTIME DESIGN SOLUTIONS

Many APIs are just a façade for backend databases, so that as APIs are driving more web and mobile applications, it is natural for front-end developers to demand more access and control over backend resources. Modern approaches to designing web APIs do well when it comes to providing access to query, filter, and send instructions to backend databases, but a new way to access large, very complex data and content backends has emerged and seen adoption. GraphQL was developed by Facebook and is now used by GitHub and others, giving developers more control over what data they accessed, allowing them to design the exact API responses they needed for any application at runtime — offloading API design to front-end developers, sharing the load with backend developers.

DESIGNING FOR PERFORMANCE

Alongside GraphQL, as it was providing API solutions for big data problems, Google had been cooking up their own approach to delivering APIs, but this time with a focus on performance and the delivery of high-volume APIs. Google's approach to delivering high-volume, high-performance APIs using gRPC has been growing, allowing developers to design a new breed of APIs using a description format called Protobuf. At Google, RESTful APIs and gRPC live side by side, sharing a common API definition that set a coordinated set of API design goals, allowing for the deployment of two distinct design patterns to achieve shifting goals.

A DIVERSE API DESIGN TOOLBOX

The API design toolbox has expanded. RESTful approaches are still the preferred design pattern to follow across the majority of APIs we see out there. However, there is an increasing number of hypermedia APIs emerging, helping deliver more experience-based APIs that behave much like the web within client applications. GraphQL is growing amongst single-page application usage, as well as applications that service big data and complex content applications. Then adding to the professional API design toolset, gRPC is getting more attention within the data center and larger enterprises, and is driving a new generation of tooling like the Spanner database platform. A truly robust API design toolbox has emerged, helping API designers and architects deliver exactly the solution they need for a variety of use cases.

DESIGNING THE INTERNET OF THINGS

There are signs of leading API design patterns spreading beyond web and mobile and being applied to the growing world of the Internet of Things (IoT). We are beginning to see REST and hypermedia patterns applied to device APIs as part of the engineering of IoT platforms and services. API design has moved out of its dogmatic RESTful roots and is becoming more about finding exactly the design blueprint you need for the job. Mobile devices, equipped with API design patterns, has opened the door pretty wide when it comes to delivering and consuming digital resources made available on consumer, commercial, and industrial IoT implementations.

COMMUNICATION IS ESSENTIAL TO DESIGN

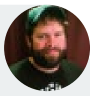
API design has proven to be much more than just the technical details, and one of the essential ingredients needed to set all this in motion was the ability to communicate around common API design patterns — if we can't share, communicate, and collaborate around a common set of design rules, the concept will never gain traction and see adoption. This is why API definitions like OpenAPI are playing an important role in the API design conversation because they allow us to communicate around a common set of design patterns. This communication is key to the growth in the number of APIs out there, as well as the number of developers who are aware and literate when it comes to common API design patterns.

API DESIGN IS THE API CONTRACT

In the last five years, API design has quickly become the contract that defines the technical, business, and political aspects of API operations, which in turn are driving web, mobile, and device applications. API design is quantified using API definition formats, which have evolved from XML, to JSON, to a more human-readable YAML — making API design a much more accessible discipline. Business stakeholders could now play a role in helping hammer out the details of the API contract, what data is available, and how it is accessed and made available within APIs.

API design is not just REST anymore. It is not just the domain of the developer. API design is about all stakeholders sitting at the table, having a conversation about what is needed to drive web, mobile, network, and device applications. In this environment, the API contract can be hammered out in plain English, defining the technical, business, and legal aspects of making data, content, and algorithms available for use across distributed applications. API design has become about following a common, well defined set of best practices, while being able to communicate, collaborate, share, and ultimately implement design strategies. Most importantly, API design has matured beyond a single approach, allowing for a robust toolbox to emerge, with a diverse set of blueprints to apply across almost any situation.

Kin Lane is the @APIEvangelist — paying attention to the technology, business, and politics of APIs. He has lived as a programmer, database administrator, architect, product developer, manager and executive, and has experience in business development, sales, and marketing. He spends his days studying the technology, business, and politics of APIs. He thrives on monitoring the API industry in real-time, as well as looking back at the history of APIs while keeping an eye on the future by tracking new emerging trends.



API Mediation: It's All About the Experience

BY **ROSS GARRETT**
HEAD OF PRODUCT MARKETING, **CLOUD ELEMENTS**

QUICK VIEW

01 Learn how Netflix realized a "one-size-fits-all" API strategy can be functionally flawed

02 The evolving integration landscape has caused developer expectations to evolve rapidly

03 Organizations must consider an integration strategy that embodies flexibility and agility

04 Personalize API experiences for different constituencies with a mediation layer

The world of application and data integration has undergone a sea change in recent years. Web APIs have turned the once debilitating task of integration into a competitive advantage — enabling new channels to market, providing a platform on which new web, mobile, and IoT applications can be built, and offering far greater engagement and interaction with customers, partners, or employees. But integration still isn't as easy as it could be — in fact, even with the broad landscape of tooling and best practices on hand, it can still be incredibly hard.

ONE SIZE DOES NOT FIT ALL

As API-centric integration evolves and matures, it has become very clear that not all API consumers are created equal: Data objects may need to be modified based on the device type; orchestration or composition may be needed depending on the sophistication of the client; security might need to be adapted to fit mobile, web, or IoT scenarios. The list goes on.

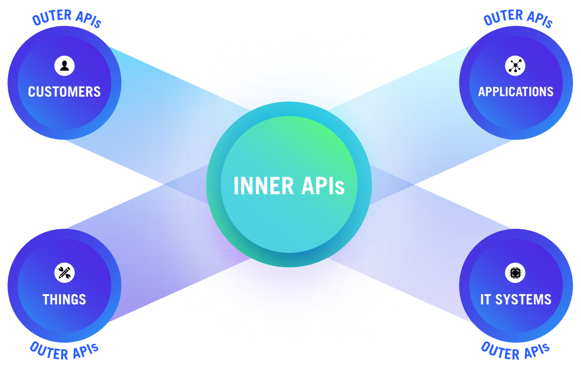
A great example of this diversity comes from Netflix, where their API development team learned quickly that a "one-size-fits-all" approach to API integration wasn't going to work. The team found that different client applications (such as a desktop browser, a mobile application, or a smart television) required different functionality as they accessed the Netflix API, thus forcing the API team to build an increasing number of features to address the demands. Once continuously updating this single interface became unmanageable, they introduced an API experience, or mediation, layer on top of their existing API platform that allowed each UI team and even partners to optimize the API experience for their specific app or device.

With each new innovation in connected devices, and each new user and application experience that accompanies these devices, organizations must consider an application architecture and integration strategy that embodies flexibility and agility. API mediation (in other words: API experience management) is becoming a more familiar concept as many organizations begin to evolve their first foray into the API Economy.

WHAT IS API MEDIATION?

API mediation is a solution for enriching or personalizing interactions between distributed application and service components. We like to call this API Experience Management. The key to the API experience model is placing a new mediation or abstraction layer between API consumers (apps, services, and "things") and API providers (services and applications). This layer wraps the backend API (the inner API) and exposes a personalized and managed API (the outer API) to each defined constituency of consumers, simplifying the developer experience while also ensuring loose coupling.

The new outer API may also aim to offer more advanced features to your developers, providing a façade on top of the inner API to enhance its functionality or simply keep pace with changes across API technologies and best practices.



Three functional areas you might want to consider adding as part of your API experience management include:

1. Eventing: There is increasing demand for event-driven integration patterns amongst web APIs, yet support amongst public and private APIs is still very limited. There is strong preference for WebHooks to handle this asynchronous eventing between application services, so offering this functionality can be a valuable addition for your developers. WebHooks are even gaining a place within API documentation languages such as OpenAPI Specification v3.0.

2. Bulk Data Operations: Bulk upload and download operations are useful for many data-centric services and applications — and where available developers are often keen to leverage this functionality. If your backend doesn't support bulk operations today, your API experience layer can be used to enable this.

3. API Discovery: To help relieve some of the pain of integration, API providers are increasingly offering a capability known as metadata discovery, so that data models and resource structures can be accessed and understood programmatically. Linked data can also enhance the usability and experience of your APIs, and again within OAS v3.0 there are new basic linking capabilities being described — a definite nod toward Hypermedia.

THE NEED FOR MEDIATION

Perhaps the simplest form of mediation is transforming legacy SOAP interfaces into more developer-friendly RESTful APIs. But in today's complex world of integration, this isn't enough to solve the challenges front-end developers are experiencing.

Rather than thoughts of divorce or separation that the term mediation may conjure up, this evolution is actually about providing a more positive experience for each of your API consumers. We've learned that quite often a one-size-fits-all approach to API design and exposure doesn't work, and different types of users, developers, and devices have different expectations and requirements when it comes to API consumption.

API EXPERIENCE MANAGEMENT IN PRACTICE

There are at least six fundamental reasons why your organization will derive value from an API experience layer in your integration architecture:

1. Integration requirements have changed: When you built out your APIs, they needed to serve a single use case, product, or business line, but now you have lots of constituencies — each with different use cases and integration requirements.

Example: I have created an API designed to allow B2B partners to resell my products and services via their channels. When a new product owner wants to build a mobile application, this API is likely not optimized for the mobile web.

2. Customer needs and expectations evolve: Your existing API layer has been in product for several years and needs to be enhanced with new functionality your developers are asking for.

Example: My API is a basic RESTful interface that customers today must poll the endpoint when checking for updated data. They would prefer a WebHooks implementation that alerts their app when new data is available.

3. Integration is about data, not interfaces: You need to synchronize data across a variety of services — even if they are in different domains. With many departments within an organization making their own purchasing decisions for the products they use, central control and data governance can be lost.

Example: If the sales organization selects Salesforce as their CRM platform and the support organization is using JIRA, are these two organizations and products sharing data effectively? Don't think about master data management, but rather a lightweight way to share and synchronize common data fields in each environment.

4. Reducing vendor lock-in: You are integrating to a particular service now, but in the future need to swap this for a new product or connect to multiple products.

Example: If I'm storing files in Box, and using Box's raw API to do that, and then I add in SharePoint or Google Drive down the road, or swap them out, I have broken everything else, unless I mediate those APIs.

5. Hiding backend complexity: You have objects or resources that exist in multiple underlying applications, databases, or other sources and want to provide consistent access to these as API resources to shield the consumer of the resource from complexity.

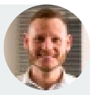
Example: Accessing a data object representing an "order" spans a number of systems — pulling data from stock management and eCommerce platforms. Developers should be abstracted from this complexity, only integrating with a single API that orchestrates the integration flows in the background to construct the data object.

6. Shifting the burden of integration: You are bringing a digital business application to market, and customers of this application will expect built-in integration to the SaaS apps they use within their organization rather than managing this burden themselves.

Example: You have built a new digital payments application and customers of this product need to be able to integrate this with their accounting applications (like NetSuite).

FINAL THOUGHTS

Quite often across each of the use cases or scenarios described above, when starting with a small number of integration points (one or two), there is no need to worry about API experience management, since each API consumer can navigate any limitations. But keep in mind that once you start integrating more and more services, products, or serve different types of API consumers and an increasing number of developers, you'll find that one-size-fits-all APIs become a barrier, rather than an enabler.


Ross Garrett is the Head of Product Marketing at Cloud Elements — responsible for market strategy, product positioning and evangelism. He is a well-known speaker at developer events and other industry conferences. Ross has over 10 years of product and marketing leadership experience in the integration space, most recently at Push Technology and previously with Axway, CA, and Layer 7.

Executive Insights on the State of Integration:

API Design and Management

BY **TOM SMITH**
RESEARCH ANALYST, **DZONE**

QUICK VIEW

01 The keys to successful integration are: 1) workflow; 2) flexibility; 3) data; and, 4) ease and reliability.

02 The most significant change to integration in the past year has been the accelerated adoption of APIs.

03 The future of integration is more systems being connected more seamlessly with artificial intelligence and machine learning.

To gather insights on the state of integration, API design, and API management, we spoke with 18 executives who are familiar with integration and APIs. Here's who we talked to:

MURALI PALANISAMY E.V.P., CHIEF PRODUCT OFFICER, APPVIEWX

KEVIN FEALEY DIRECTOR OF AUTOMATION & INTEGRATION SERVICES, ASPECT SECURITY

MAX MANCINI V.P. OF ECOSYSTEM, ATLASSIAN

SHAWN RYAN V.P. PRODUCT MARKETING, DIGITAL AS A SERVICE, AXWAY

PARTHIV PATEL CTO, FLOCK

ANWESA CHATTERJEE DIRECTOR OF PRODUCT MARKETING, INFORMATICA

SIMON PEEL CMO, JITTERBIT

KEOKI ANDRUS V.P. OF PRODUCTS, JIVE

STEVE BUNCH PRODUCT MANAGER APIs AND INTEGRATIONS, JIVE

RAJESH GANESAN DIRECTOR OF PRODUCT MANAGEMENT, MANAGEENGINE

BROOKS CRICHLOW V.P. PRODUCT MARKETING, MONGODB

DEREK SMITH CEO, NAVEEGO

GUILLAUME LO RE SENIOR SOFTWARE ENGINEER, NETVIBES

VIKAS ANAND V.P. PRODUCT MANAGEMENT & STRATEGY – INTEGRATION, ORACLE

KESHAV VASUDEVAN PRODUCT MARKETING MANAGER, SMARTBEAR

KEVIN BOHAN DIRECTOR OF PRODUCT MARKETING, TIBCO

PETE CHESTNA DIRECTOR OF DEVELOPER ENGAGEMENT, VERACODE

MILT REDER V.P. OF ENGINEERING, YET ANALYTICS

KEY FINDINGS

01 The keys to a successful integration strategy are: **1) workflow; 2) flexibility; 3) data; and, 4) ease/reliability**. Understand user workflow, who the users are, and what they're trying to accomplish. If you are successful, the technology will become part of the larger workflow. You also need to understand the possible use cases. A technological need likely requires a private API, while business drivers require public or partner-based APIs. Successful integrations think about the user experience (UX) first. They consider how end users want to interact with, and benefit from, the product.

Integrations need to be sufficiently flexible to handle new use cases and processes, as well as the ability to scale as data expands exponentially. Toolsets also need to be flexible and shouldn't be too restrictive when working with external data. Recognize that any given service may be superseded by another in the future – service providers may go out of business or better options may come about.

Have a sound strategy to expose and build on the data. Meet data where it lives in real-time. Take data from anywhere and enable business owners to get value from it.

Make it easy for developers to get started. API providers should provide resources to help developers effectively learn and begin using these processes. Provide SDKs for the most popular languages and ensure that documentation is up-to-date.

02 The most significant changes to integration in the past year have been the **accelerated adoption of APIs** and the importance of containers and microservices. Every service is an API-driven technology, and as such you must take an API-first approach. There's been tremendous proliferation of APIs, since software is not homogeneous.

There is a heightened focus on the containerization of applications. DevOps is being adopted and teams are leveraging Docker and AppFoundry. There is a need to support containerized development and microservices. With microservices hitting their stride, developers have the tools to quickly build integrations with any system imaginable and the challenges posed by legacy systems integration are being diminished.

03 Swagger is the most frequently mentioned technical solution with regards to the design and management of APIs. Developers like to make APIs self-documenting with Swagger. Swagger defines API structure and generates API documentation as part of the build process. Developers understand that well-maintained, easy-to-use API documentation is a top priority when building new apps. Swagger makes it easier for developers to build integrations.

04 Given the ubiquity of APIs, real-world problems being solved by APIs yields a wide and deep group of use cases. Companies are integrating CRMs to enable click-to-call and screen pop-ups on their websites. The Sacramento Kings have the most connected basketball arena in the NBA and have integrated 35 different applications with APIs including ticketing, ordering, CRM, concessions, loyalty, etc. Financial services are using integration to facilitate seamless and secure mobile banking applications. A container shipping company has generated a new revenue stream by opening their digital business platform to other businesses who are provisioning containers as they need them – for more than just shipping. LA Metro has a lot of information in their CRM, service cloud, card fulfillment, and ERP systems. All of these are integrated together, so customers can reload their transportation card via the website.

05 The most common issue you see affecting integration is the **failure to follow good API design practices**. APIs should be easy to adopt, flexible yet stable, backwards-compatible, easy to migrate, and secure. Look at the integration project strategically and build for the future.

1) Easy to adopt: APIs should be implemented using popular and standard technologies, rather than proprietary mechanisms. **2) Flexible, yet stable**: Frequently changing and deprecating an API is often a recipe for failed integrations. It makes sense to be flexible in terms of having more parameters or more calls than changing an existing API for every request. **3) Backwards compatibility**: Calling applications could stall and become useless if an API gets removed or discontinued often. Backward compatibility is needed for a while to allow all applications to move to more recent versions. **4) Migration**: It is critical to provide procedures and tools to make it easier to move to later versions of the API. **5) Secure:** Often overlooked, API security is a critical aspect that could make or break API adoption. It is vital for designers to keep security as an important factor right from the initial design, yet abstract the complexity from the API consumers. This includes choice of authentication mechanism, encryption of the communication channel, and ensuring authorization rules are applied and all input is validated on every API call.

06 The future of integration is using AI/ML to seamlessly connect more systems. Co-created integration is a mindset where two providers get together to provide seamless integration of their products. Serverless will hide the complexity of integrations, enabling system-to-system communication across multiple platforms. Users get a more robust experience as more services are integrated seamlessly.

We'll use AI/ML to predict future integration needs, which will produce more efficient connections. Systems will be more intelligent as they are ingesting data in real time.

07 There are a broad range of concerns around integration today. The most frequently mentioned were **security, tooling, and being business-centric**. The top concern was handling information security. Data must be encrypted in transit, and vendors must supply security around APIs to provide the best UX. Developers need to ensure their users can only access the data and services they are entitled to access.

We're at a point where we have a host of powerful and accessible tools to build the next generation of integrated systems, but we need to be mindful of the discipline it takes to use them well. Tools need to bring together capabilities without causing confusion with regards to what tool is used to solve a specific problem.

We need to be more business-centric than before and ensure we're meeting a consumer need. Identify integrations that create solutions to customers' problems.

08 The skills developers need to ensure their code and applications integrate well are **documentation, API design, and an API-first mentality**. Properly documenting the API for others ensures widespread adoption. Easy-to-follow tutorials for common tasks are helpful.

Developers should learn good API design, follow frameworks, understand use cases, and treat endpoints that address those needs. Think about integration in a sustainable fashion, build flexible APIs that will last 10 years. Lay out how you will expose the API. Think about the implications of what you're building. Create a contract first, then create a service to ensure people have the best experience, and don't forget to run tests before implementing the backend.

09 Additional considerations include: 1) **how to help legacy systems make the digital transformation**; 2) thinking about **the future of APIs** on several fronts; 3) **communicating the value of an API platform**.

Most companies are not digitally native. They are dealing with legacy systems. How do we modernize and **leverage traditional interfaces for the digital transformation**? How do we make innovation possible for these companies?

There's a lot of consolidation going on with API management. What's the future of the space, especially in the cloud? As developers, we need to **plan for change** by taking a critical look at the decisions we make around APIs, especially around API design.

The most elegantly architected, best integrated, and well-designed API framework will not attract developers without **a clear story about how developers can create value** (remove friction from workflows, make money, get new customers, etc.) by building an API platform that works in any environment for hybrid consumption of APIs.

The keys to success for APIs are to: 1) **stay abreast of technology**, invest in integration, and provide a level of investment to ensure customers can use the product they purchased for a reasonable amount of time going forward; and, 2) **use telemetry** to know what people are and aren't using so you can make informed business decisions.

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

Solutions Directory

This directory of APIs, ESBs, integration platforms, and frameworks provides comprehensive, factual comparisons of 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.

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
1060 Research	NetKernel	Integration Platform	N/A	1060research.com/products
Actian	Actian DataCloud	Integration Platform/PaaS	Demo Available by Request	actian.com/data-integration/datacloud-connector
Adaptris	Adaptris Interlok	Integration Platform/PaaS	N/A	adaptris.com/interlok.php
Adeptia	Adeptia Integration Suite	Integration Platform/PaaS	30 Days	adeptia.com/products/adeptia-integration-suite
AdroitLogic	UltraESB	ESB	Open Source	adroitlogic.org/products/ultraesb
Amazon	Amazon SQS	Message Queue	Free Tier Available	aws.amazon.com/sqs
Apache Software Foundation	ActiveMQ	Message Queue	Open Source	activemq.apache.org/download.html
Apache Software Foundation	Apache Camel	Integration Framework	Open Source	camel.apache.org
Apache Software Foundation	Qpid	Message Queue	Open Source	qpid.apache.org
Apache Software Foundation	ServiceMix	Integration Container/Platform	Open Source	servicemix.apache.org
Apache Software Foundation	Synapse	ESB	Open Source	synapse.apache.org
Apcera	NATS	Message Queue	Open Source	nats.io
Apigee	Apigee Edge	API Management	Available by Request	apigee.com/api-management/#/products
Aurea Software	CX Messenger	ESB	N/A	aurea.com/customer-experience-platform/cx-messenger
Axway	Axway API Management	API Management	N/A	axway.com/en/enterprise-solutions/api-management-solutions

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Azuqua	Azuqua Platform	Integration Platform/PaaS	14 days	azuqua.com
Bosch Software Innovations GmbH	ProSyst OSGi	API Management	N/A	bosch-si.com/iot-platform/iot-platform/gateway/software.html
BroadPeak	K3 Integration Platform	Integration Platform/PaaS	Demo Available by Request	broadpeakpartners.com/data-integration-platform
Built.io	Built.io Flow	Integration Platform/PaaS	14 days	built.io/products/flow
CA Technologies	CA API Management	API Management	Available by Request	ca.com/us/products/api-management.html
Cazoomi	Cazoomi	API Management	Free Tier Available	cazoomi.com
Cleo	Cleo Integration Suite	Integration Platform	Demo Available by Request	cleo.com/products/cleo-integration-suite
Cloud Elements	API Manager Platform	API Management	Available by Request	cloud-elements.com/platform
Dell Boomi	AtomSphere	Integration Platform/PaaS	30 Days	boomi.com/integration/integration-technology
DreamFactory Software Inc.	DreamFactory	API Management	Demo Available by Request	dreamfactory.com
Enduro/X	Enduro/X	Integration Platform	Open Source	endurox.org
Fanout, Inc.	Fanout Zurl	Integration Platform	Open Source	github.com/fanout/zurl
Firoano Software	Fiorano API Management	API Management	Demo Available by Request	fiorano.com/products/api-management.php
Firoano Software	Fiorano ESB	ESB	Open Source	fiorano.com/products/open-source-esb/fiorano-esb
Firoano Software	Fiorano Integration	Integration Platform	Demo Available by Request	fiorano.com/products/fiorano-integration-platform
Firoano Software	FioranoMQ	Message Queue	Free Solution	fiorano.com/products/Enterprise-Messaging/JMS/Java-Message-Service/FioranoMQ.php
Flowgear	Flowgear	Integration Platform/PaaS	14 days	flowgear.net
Fujitsu	RunMyProcess	Integration Platform/PaaS	Demo Available by Request	runmyprocess.com/platform
Fujitsu	Fujitsu Business Operations Platform	SOA Governance	N/A	fujitsu.com/global/products/software/middleware/application-infrastructure/interstage/solutions/bop
Hewlett Packard Enterprise Development LP	HP SOA Systinet	SOA Governance, API Management	N/A	saas.hpe.com/en-us/software/api-management-soa-governance
IBM	IBM API Management	API Management	Free Tier Available	ibm.com/software/products/eN/Api-connect

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
IBM	IBM Integration Bus	ESB	30 Days	ibm.com/software/products/en/integration-bus-advanced
IBM	IBM MQ Advanced	Message Queue	Free Developer Tier	ibm.com/software/products/en/mq-advanced
iMatix Corporation	ZeroMQ	Message Queue	Open Source	zeromq.org
Informatica	Informatica iPaaS	Integration Platform/PaaS	30 Days	informatica.com/products/integration-platform-as-a-service.html
Information Builders	iWay Service Manager	Integration Platform	Available by Request	informationbuilders.com/products/iway-service-manager
integrator.io	Celigo	Integration Platform/PaaS	Available by Request	celigo.com/ipaas-integration-platform
InterSystems Corporation	Ensemble	ESB	N/A	intersystems.com/our-products/ensemble/ensemble-overview
Iron.io	IronMQ	Message Queue	Free Solution	iron.io/platform/ironmq
Jitterbit	Jitterbit	Integration Platform/PaaS	Available by Request	jitterbit.com
JNBridge LLC	JNBridge Adapters	Integration Framework	30 Days	jnbridge.com
Liaison Technologies	Liaison Alloy	Integration Platform/PaaS	N/A	liaison.com/liaison-alloy-platform
Mertech Data Systems, Inc.	evolution	API Management	Available by Request	thriftly.io
Micro Focus	Artix	ESB	30 Days	microfocus.com/products/corba/artix
Microsoft	BizTalk Server	Integration Platform/PaaS	180 Days	microsoft.com/en-us/cloud-platform/biztalk
Microsoft	Microsoft Azure API Management	API Management	Free Tier Available	azure.microsoft.com/en-us/services/api-management
MID GmbH	Innovator Enterprise Modeling Suite	SOA Governance	60 Days	mid.de/en/business-activities/tools/innovator
MuleSoft	Mule ESB	ESB	Yes	mulesoft.com/platform/soa/mule-esb-open-source-esb
MuleSoft	Anypoint Platform	API Management	Available by Request	mulesoft.com/platform/enterprise-integration
nanoscale.io	nanoscale.io	Integration Platform/PaaS	Free Tier Available	nanoscale.io
NEC	WebOTX ESB	ESB	Open Source	jpn.nec.com/webotx/download/manual/92/serviceintegration/esb
Neuron ESB	Neuron ESB	ESB	30 Days	neuronesb.com
OpenESB	OpenESB	ESB	Open Source	open-esb.net
OpenText Corp.	GXS Enterprise Gateway	ESB	N/A	opentext.com/what-we-do/products/business-network/b2b-integration-services

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Oracle	Oracle Service Bus	ESB	Free Solution	oracle.com/technetwork/middleware/service-bus/overview
Oracle	Oracle SOA Suite	SOA Governance	30 Days	oracle.com/us/products/middleware/soa/suite/overview
OW2 Middleware Consortium	Petals ESB	ESB	Open Source	petals.ow2.org
Particular Software	NServiceBus	ESB	Open Source	particular.net/nservicebus
Pivotal Software, Inc.	RabbitMQ	Message Queue	Open Source	network.pivotal.io/products/pivotal-rabbitmq
Pivotal Software, Inc.	Spring Integration	Integration Framework	Open Source	projects.spring.io/spring-integration
PokitDot, Inc.	Pokitdot	Healthcare API Management	Free Developer Tier	pokitdok.com
Proden Technologies	accel<> DS Data Integrator	Data Integration	Free solution	prodentechnologies.net
Red Hat	3Scale API Management	API Management	90 days	3scale.net/api-management
Red Hat	Fabric8	Integration Platform/PaaS	Open Source	fabric8.io
Red Hat	HornetQ	Message Queue	Open Source	hornetq.jboss.org
Red Hat	JBoss Fuse	ESB	Open Source	redhat.com/en/technologies/jboss-middleware/fuse
Redis Labs	Redis	Database and Message Queue	Open Source	redis.io
Restlet, Inc.	Restlet Framework	API Management	Open Source	restlet.com/open-source
RoboMQ	RoboMQ	Integration Platform/PaaS	60 Days	robomq.io
Rocket Software, Inc.	Rocket Data	Data Integration Platform/PaaS	Demo Available by Request	rocketsoftware.com/products/rocket-data
Rocket Software, Inc.	LegaSuite Integration	Integration Platform/PaaS	Demo Available by Request	rocketsoftware.com/products/rocket-legasuite/rocket-api
Rogue Wave Software	Akana API Management	API Management	Demo Available by Request	roguewave.com/products/akana/solutions/api-management
Rogue Wave Software	Akana Software Suite	SOA Governance	Demo Available by Request	roguewave.com/products-services/akana/solutions/integrated-soa-governance
SAP	SAP API Business Hub	API Management	Available by Request	go.sap.com/product/technology-platform/api-management.html
SAP	SAP HANA	Integration Platform/PaaS	Available by Request	hcp.sap.com
SAP	HANA Cloud Integration	Integration Platform/PaaS	Available by Request	help.sap.com/cloudintegration

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
SAP	SAP NetWeaver	SOA Governance	Available by Request	go.sap.com/community/topic/netweaver.html
Scheer	E2E Bridge	Integration Platform/PaaS	60 Days	e2ebridge.com/en
SEEBURGER	Seeburger	Integration Platform/PaaS	N/A	seeburger.eu/business-integration-suite.html
Sikka Software Corporation	Sikka One API	API Management	Demo Available by Request	sikkasoft.com
SmartBear Software	SoapUI NG	API Testing	14 days	smartbear.com/product/ready-api/soapui-ng/overview
SmartBear Software	LoadUI	API Load Testing	14 days	smartbear.com/product/ready-api/loadui/overview
SmartBear Software	Swagger	Integration Framework	Open Source	swagger.io
SnapLogic, Inc.	Elastic Integration Platform	Integration Platform/PaaS	Demo Available by Request	snaplogic.com/enterprise-integration-cloud
Software AG	WebMethods	Integration Suite	Available by Request	softwareag.com/corporate/products/az/webmethods/default.asp
StormMQ	StormMQ	Message Queue	Open Source	github.com/stormmq
SwarmESB	SwarmESB	ESB	Open Source	github.com/salboaie/SwarmESB
Talend	Talend ESB	ESB	Available by Request	talend.com/products/application-integration
Talend	Talend Integration Cloud	Integration Platform/PaaS	30 Days	talend.com/products/integration-cloud
TIBCO Software Inc.	TIBCO Cloud Integration	Integration Platform/PaaS, API Management	30 Days	tibco.com/products/tibco-cloud-integration
TIBCO Software Inc.	Mashery	API Management	30 Days	mashery.com/api-management
Tyk.io	Tyk	API Management	Open Source	github.com/TykTechnologies/tyk
Vigience	Vigience Overcast	Salesforce Integration Platform	N/A	overcast-suite.com
WSO2	WSO2 Carbon	Integration Platform/PaaS	Free Solution	wso2.com/products/carbon
WSO2	WSO2 Enterprise Integrator	ESB	Free Solution	wso2.com/products/enterprise-service-bus
WSO2	WSO2 Message Brokers	Message Queue	Free Solution	wso2.com/products/message-broker
Zapier	Zapier	API Management	Free Tier Available	zapier.com

GLOSSARY

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

DENIAL OF SERVICE (DOS) ATTACK
An attack on a resource, caused when a perpetrator intends to make a resource unavailable by placing massive amounts of requests on the given resource.

DOCUMENTATION-DRIVEN DEVELOPMENT
A philosophy of software development in which documentation for a feature is written before the feature is created.

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

FARMING
The idea of providing services by leveraging the services of another resource or resources.

GRAPHQL
A query language and runtime for completing API queries with existing data.

HYPERTEXT TRANSFER PROTOCOL (HTTP)
A protocol used to exchange hypertext. The foundation of communication for websites and web-based applications.

INTEGRATION FRAMEWORK
A lightweight utility that provides libraries and standardized methods to coordinate messaging among different software.

IPAAS
A set of cloud-based software tools that govern the interactions between cloud and on-premises applications, processes, services, and data.

MESSAGE BROKER
Middleware that translates a message sent by one piece of software to be read by another piece of software.

MICROSERVICES
Small, lightweight services that each perform a single function according to a domain's bounded contexts. The services are independently deployable and loosely coupled.

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

REPRESENTATION STATE TRANSFER
A set of principles describing distributed, stateless architectures that use web protocols and client/server interactions built around the transfer of resources.

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

SERVICE DISCOVERY
The act of finding the network location of a service instance for further use.

SERVERLESS COMPUTING
A cloud computing model in which a provider manages the allocation of servers and resources

SERVICE ORIENTED ARCHITECTURE (SOA)
An application architecture built around the use of services that perform small functions.

SIMPLE OBJECT ACCESS PROTOCOL (SOAP)
A protocol that is used by web services to communicate with each other, commonly used with HTTP.

STATELESS SESSION STATE
A session which no information is maintained between the sender and receiver.

SWAGGER
A definition format used to describe and document RESTful APIs, to create a RESTful interface to develop and consume APIs.

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.

Take your development career to the next level.

From DevOps to Cloud Architecture, find great opportunities that match your technical skills and passions on DZone Jobs.

Start applying for free

THESE COMPANIES ARE NOW HIRING ON DZONE JOBS:

Microsoft cloudbees MuleSoft Couchbase THOMSON REUTERS

Is your company hiring developers?

Post your first job for free and start recruiting for the world's most experienced developer community with code '**HIREDEVS1**'.

Claim your free post