

API Management Trend Report

Seamless Experiences of Connected Systems

BROUGHT TO YOU IN PARTNERSHIP WITH



POSTMAN

Table of Contents

3 Highlights & Introduction

BY KARA PHELPS

5 Developer Experience for APIs

BY LORNA JANE MITCHELL

10 Accelerating Your Enterprise API Program

BY MATT MCLARTY

13 How APIs Can Extend Your Business

BY AYSEGUL YONET

16 Key Research Findings

BY JORDAN BAKER

20 Diving Deeper Into API Management

21 Executive Insights on the State of API Management

BY TOM SMITH

BUSINESS & PRODUCT

Jesse Davis EVP, Technology

Kellet Atkinson Media Product Manager

PRODUCTION

Billy Davis Production Coordinator

Naomi Kromer Sr. Campaign Specialist

Michaela Licari Campaign Specialist

SALES

Kendra Williams Sr. Dir. of Media Sales

Tevano Green Sr. Account Executive

Brett Sayre Account Executive

Alex Crafts Key Account Manager

EDITORIAL

Mike Gates Senior Editor

Kara Phelps Editorial Project Manager

Tom Smith Research Analyst

Jordan Baker Publications Associate

Sarah Sinning Staff Writer

Andre Lee-Moye Content Coordinator

Lauren Ferrell Content Coordinator

Lindsay Smith Content Coordinator

Charleigh Smith Content Coordinator

Peter Connolly Content Coordinator

MARKETING

Susan Wall CMO

Aaron Tull Dir. of Demand Generation

Waynette Tubbs Dir. of Marketing Comm.

Colin Bish Member Marketing Spec.

Suha Shim Acquisition Marketing Mgr.

Cathy Traugot Content Marketing Mgr.

Melissa Habit Content Marketing Mgr.

Kate Lloyd Customer Marketing Mgr.

Lisa Cronk Graphic Design Specialist

To sponsor a Trend Report:

Call: (919) 678-0300

Email: sales@devada.com

Highlights & Introduction

By Kara Phelps, Editorial Project Manager at DZone

"APIs are the synapses that allow our interconnected world to communicate," our director of engineering, Rick Severson, wrote to introduce another DZone publication last year. Application Programming Interfaces (also known as APIs) continue to run the modern world as we know it. They're a key driver of innovation, essential to the growth of existing applications, and one of the main reasons why software can complete the tasks we ask it to do — even though customers don't always realize it.

You could say that APIs, and those who work with them, are the unsung heroes of software development. It's crucial that APIs stay updated, secure, and operational — and if you want to drive adoption, developers also need clear documentation and good SDKs. In this Trend Report, we'll explore the world of API management and make predictions about where it's heading next.

In "Developer Experience for APIs," Lorna Jane Mitchell explores what takes for your API to delight developers. Organizations that provide thorough documentation in multiple formats, for example, will be more likely to offer the kind of support a developer needs to learn. For more tips, you can read the full article later in this Trend Report.

Aysegul Yonet's article, "How APIs Can Extend Your Business," delves into the different levels of scope that APIs provide and what each scope can bring to an organization as it scales. In this thorough refresher on the concept of APIs, she calls them "the next step in the evolution of software development."

In "Accelerating Your Enterprise API Program," Matt McLarty offers practical solutions to the challenges of API-driven digital transformation. You'll need to strategize around more than just API strategy — your organization should be seeing a cultural shift, and the entire API ecosystem should have buy-in. Through the lens of API management, this article examines the ways that large organizations purposefully adopt and standardize new technologies.

TREND PREDICTIONS

- ▶ Microservices-based architecture will continue to grow in popularity, increasing the necessity of building a scalable way to send messages among applications. These types of communications will play a critical role in API programs at the organizational level.
- ▶ Across all types of integrated systems, JSON will lead the pack in terms of adoption among data serialization and interchange formats.
- ▶ The need for open-source integration frameworks will help expand the broader interest in open-source solutions.

DZone Research

In the Key Research Findings, we're also sharing a few insights from our survey of DZone readers involved in API management. Our results draw from 889 responses, gathered over the course of two weeks earlier this summer.

Sixty-seven percent of survey respondents reported that their organizations currently use a microservices-based architecture. 71 percent of respondents also rated scalability as the single most important quality for software architecture to have. Microservices are becoming crucial to enterprise software development, and we predict that microservices-based architecture will continue to grow in popularity. This will increase the necessity of building a scalable way to send messages among applications — one of the most common use cases for APIs.

67% of survey respondents reported that their organizations currently use a microservices-based architecture.

Among respondents, JSON is by far the most popular data serialization and interchange format. 96% of respondents use JSON, and 72% reported that they enjoy working with it. Given that data, we can extrapolate that JSON will keep dominating in the near future.

Finally, we noticed that most of the technologies, tools, and frameworks that respondents said they plan to use within the next 12 months are open-source solutions. Open-source integration frameworks are less risky to implement than custom-built solutions, as they've already proven effective, and development teams often prefer having an opportunity to examine the code before signing off. Considering that API programs are often critical to broader business goals, we're predicting that the need for open-source integration frameworks will help expand the wider interest in open-source solutions.

Thank you for downloading this Trend Report. We hope you find all the information you need. 

Developer Experience for APIs

By Lorna Jane Mitchell, Developer Advocate at Nexmo

We're all familiar with user experience: putting the user front-and-center of the products we create. For APIs, that user is a developer, and so we talk about developer experience. As with the user experience, we start from the point of view of some of the individuals we're hoping to serve and we consider their stories.

A delightful developer experience is about more than just shipping a working API and perhaps some reference documentation, although both of those things are important. Developers expect APIs to be more than reliable — they must be consistent throughout so that developers can quickly find their way to what they need. The documentation aspect of the developer experience is vitally important; many purchasing decisions will be made on the strength of the supporting materials available. There also needs to be an understanding of the developers' context, the technologies they are familiar with, and the tools they typically use.

API Design and Standards

Whatever style and standards you pick for your API, choose once and then *stick to them*. Keep the same field called the same thing throughout, even if it could be abbreviated in some contexts. Format all the dates in the same way, with days, times, and time zones all included, and keep the same field the same length at all times. Consistency is what makes developers feel at home in your API. They can only get familiar with the API if it feels familiar right across it (and if you have multiple APIs, ideally they should also share some characteristics).

When picking standards, the best are the ones that already exist. Try [OAuth2](#) for authenticating users with tokens and scopes — or [JWT](#) (JSON Web Tokens) if you expect client-side usage. Try [HAL](#) as a data structure for links and related resources. How about [RFC 7807](#) (catchy name, I know) for your error messages? There are lots of ways to make your developers' lives easier, and using existing standards will help them settle into your API more quickly.

It's really important that the standards for your API are written down somewhere. This document serves as a reference manual for everyone who might add a feature to your API, and helps spell out the conventions and keep everything consistent. Sometimes I feel silly stating obvious things, such as "if there are multiple results, these will be in an array." But at other times, I refer back and can more quickly ship my feature because the information I need is clear and available.

TREND PREDICTIONS

- ▶ The developer-specific user experience of an API program will have a direct effect on its future success or failure.
- ▶ With well-designed APIs, solid supporting documentation, and good support for developers' existing tools, more organizations will begin to delight their technical users.

Consistency in design is key — and it takes time and diligence to get it right. Take a moment to check every API feature against both the API standards document and perhaps a couple of real, live developers before you ship it.

Documentation and Developer Portals

A [Developer Portal](#) is so much more than a place to keep the API reference documentation. The documentation definitely needs to be included, but it is not enough by itself. I like to say that every API is someone's first API, so the documentation needs to reflect that. Include overview information and a "Getting Started" walkthrough for developers who are just cutting their teeth. I like to include a "Quick Start" article as well for those who are familiar with APIs and looking to get going fast.

Having tutorials covering the use cases that you had in mind when you designed the API is incredibly helpful; even developers with other goals will be able to read these examples and pick and choose the pieces they need for their own projects. Creating these tutorials also allows you to try out your API in a more real-world setting than thinking about a single endpoint at a time as you do when creating an API. Sometimes, seeing the API calls in sequence can inform future changes or improvements to the API.

Depending on the target users for the API, your developer portal may include code samples in the languages or chosen technologies for those developer communities. On [developer.nexmo.com](#), if you look at the code snippets for any of our products, you'll see that we have code snippets in four or five programming languages for most of the examples. We get good feedback from our developer communities about this feature.

A detailed walkthrough of every step required to achieve a common use case can be incredibly valuable, especially for newcomers to the API.

Developers aren't a particularly homogenous group, and they learn in different ways. To enable all your API consumers to get up to speed quickly, it's good to provide a few different types of documentation. Some developers find a high-level overview guide really useful as a way of finding their way around and perhaps learning some of the vocabulary that your product uses. We already touched on the tutorials, but a detailed walkthrough of every step required to achieve a common use case can be incredibly valuable, especially for newcomers to the API. It's also helpful to include shorter, focused snippets for the impatient and potentially more experienced developer. Don't be afraid to mix the types or link extensively between them; this can help to re-orient a lost developer and lead them to the information they need.

With all documentation, maintainability can be a problem and this is especially true when the product is iterating quickly — which happens more often than perhaps we'd like. Keeping the documentation up-to-date with the API can be a puzzle, and I've seen a few different solutions. The most effective approach I've seen is to use an API description format such as OpenAPI, and then generate the API reference documentation from that.

API Descriptions Improve Developer Experience

API Description languages, such as [OpenAPI Specification](#) or its predecessor, Swagger, are machine-readable and comprehensive descriptions of every aspect of an API. The authentication; URL endpoints; every field, value, and possible response ... they are all documented in the OpenAPI specification document, and this file can be used to create other outputs.

The most obvious and possibly most useful is that API reference documentation can be generated from an OpenAPI specification. Try a tool like [Stoplight](#) or [ReDoc](#) and you'll have nice-looking documentation that includes all the comments,

examples, and other details that were included in the spec. When the API changes, you can update the spec and then recreate the documentation — very handy.

It's also possible to create other outputs from the API spec. For example, you can generate a basic SDK from the OpenAPI specification — in any programming language. Crucially, your developers can also do this for themselves. Documentation that was created from OpenAPI will typically include a link to download the original spec file so that developers can use this themselves.

I do see developers generating SDKs, although we are definitely in the early days for those tools. By far the most popular use case, however, is to import the OpenAPI spec into an HTTP client such as [Postman](#). Postman is a very popular HTTP client that many

Importing the OpenAPI spec file into Postman gives [developers] a ready-to-use collection of API examples in a tool they already know and love.

developers will be familiar with; they probably have a few sets of API calls stored in the tool that they can quickly use. (*Editor's note: Postman is a sponsor of this Trend Report. The author is not affiliated with Postman.*) Importing the OpenAPI spec file into Postman gives them a ready-to-use collection of API examples in a tool they already know and love. With the API calls at their fingertips, we find this sort of tool improves the speed of developer onboarding significantly, and also helps them work out any problems they do encounter.

Ingredients for a Good API Developer Experience

With a well-designed API, solid supporting documentation, and good support for developers' existing tools, you will delight the technical consumers of your API every time. Hopefully I've given you some ideas for your next API project, and I can't wait to see what you build. 🎲

Faster, Better, More Resilient APIs

By **Abhinav Asthana**, CEO and Co-Founder at Postman

In an API-first world, APIs are the building blocks of effective software. As software growth continues, APIs evolve even faster to connect services. Innovation in APIs is driving the future of software development, and collaboration is driving innovation.

APIs are rarely developed in isolation anymore. New strides in collaborative practices and technology are inspiring more imaginative, flexible, and capable APIs. Speed to production and functionality are improving — and collaborative platforms for API development are driving that shift.

Faster

Not long ago, the standard API lifecycle required teams to wait between each step. Development would pass something off to QA, pause while QA tested, QA would pass the baton back, and so on. This “hurry up and wait” mentality delayed progress and prolonged time to production.

Today, collaborative technology has completely restructured the lifecycle. Now, rather than a process with starts and stops, the lifecycle reflects multiple and frequent connections among developers, QA, and other stakeholders. API design can be separated from backend development — **enabling efforts to run in parallel**, along with testing and documentation.

Better


When kicking off API design, it is nearly impossible for a single developer to anticipate all of the possible issues, potential blockers, and necessary functionalities of an API. To find solutions, overcome obstacles, and avoid wasted effort, teams need tools that **ensure transparency and maintain a single source of truth**.

Transparency through the API lifecycle relies on access control, open communication, prompt feedback, and optimal workflows, all of which are supported by a collaboration platform. And, transparency doesn't end at the endpoint — it must extend to the users consuming the API via accurate, up-to-date documentation, complete with robust use cases and examples.

Resilient

Whether you're creating an API or using one, things will change. Resiliency relies on transparency. To keep up, developers must maintain several versions of an API simultaneously — ensuring that consumers that depend on prior versions can maintain their systems. In addition, APIs need to have built-in tests to prevent breakage. Setting up monitors, automated tests, and consumer-driven contract tests can **make the difference between a failure and a successful API**.

Bringing It All Together

Collaboration will continue to drive innovation — and platforms that provide holistic views of API development will be the foundation on which collaborative practices are built, ultimately inspiring more creative and capable software. 



POSTMAN

API DEVELOPMENT. SUPERCHARGED.

Accelerating Your Enterprise API Program

By **Matt McLarty**, Global Leader of API Strategy at MuleSoft

The business landscape has changed. We have shifted from the industrial economy of the twentieth-century to the digital economy of the twenty-first. Some of the biggest brands in the world — companies like Blockbuster, Kodak, and a long list of retailers — failed to anticipate the new digital reality, and have been replaced in the global consciousness by brands that only emerged at the turn of the millennium: Netflix, Facebook, Amazon, and others. Does this mean that every enterprise from the previous business era is doomed?

The answer to that question, fortunately, is no. Organizations of every size in every industry have recognized that they need to adapt to digital techniques in order to thrive. One of the most effective tools in digital transformation is the API. As the web native giants have shown, APIs provide a bridge to many digital opportunities. Externally, Netflix used APIs to establish a ubiquitous presence on digital platforms. Internally, Amazon used APIs to front all of their business capabilities, leading to unprecedented agility and scalability, and ultimately to a multi-billion-dollar business line in AWS. Established enterprises should adopt a strategy that clarifies why and how they can use APIs in their digital transformation.

TREND PREDICTIONS

- ▶ APIs are essential to the future of digital transformation.
- ▶ The organizations poised to benefit the most from APIs are the ones that adopt the organizational principles and practices of API pioneers.
- ▶ APIs are digital products. Thoughtfully conceived and executed API programs will become even more key to corporate strategy.

Your API Strategy Shouldn't Just Be an API Strategy

I help organizations around the world on this path in my role as Global Leader of API Strategy at MuleSoft and want to share the most successful approaches. First of all, an API strategy should not be narrowly scoped to just the API-related activities in an organization. To gain full advantage of the business opportunities, an API strategy should be considered part of a company's overall digital strategy. Secondly, the organizations that benefit most from APIs are the ones that adopt the organizational principles and practices of API pioneers, not just their technologies. Thirdly, leading API companies are purposeful about the technologies they use, and careful about how new tools are introduced and standardized. Lastly, and perhaps most importantly, the companies that get the most out of APIs spend a great deal of time cultivating and engaging an ecosystem of developers, partners, stakeholders, and platforms for their APIs.

These four areas — establishing a strategy, aligning the organization and culture, evaluating and building technologies, and engaging with the ecosystem — serve as a useful blueprint for any company's API strategy. By assessing the needs in each area, an organization can formulate a balanced strategy that matches its goals and priorities. If you know why you want to use APIs, how you are going to deliver them, and who needs them, you are much more likely to pick the right APIs to produce at the right time.

Four Pillars of an API Program

Even with the balanced perspective on API strategy that these areas provide, it is hard for large enterprises to determine where their implementation should start. Digital transformation requires a shift in the business paradigm of people, processes, technologies, and culture. To mobilize these changes using APIs as the catalyst, I advocate a programmatic approach.

An API program is a cross-enterprise change initiative that leverages the benefits of APIs in support of an organization's digital strategy. Most effectively driven by a combined team of business and technology leaders, an API program provides greater alignment, visibility, and communication of an organization's digital strategy with its API-enabled efforts. In addition to getting the enterprise's coordinated API-related activities in motion, a well-executed API program can provide stronger governance, lead to economies of scale, and provide an organizational conduit to external entities in the digital ecosystem. The four focus areas of the API strategy blueprint can be used as pillars in an enterprise API program.



Establish an API-Enabled Digital Strategy

The first step in executing an API program is to set its direction. Although much of the change needed to succeed in the digital economy must happen within an organization, the direction is often best guided from without. Specifically, customer needs should be the fundamental driver of the digital strategy. In his extensive study of companies that survive disruption — in both *The Innovator's Dilemma* and *The Innovator's Solution* — Clayton Christensen outlined the “jobs-to-be-done” framework that allows organizations to reimagine their own business strategies to match their

customers' needs. Blockbuster was susceptible to disruption from Netflix since they could not switch their thinking from video rentals to the broader category of home entertainment. Netflix, on the other hand, amplified their growth by maintaining a focus on their customers' home entertainment demands, while switching their supply side from physical media to online streaming.

It is not easy for an established organization to shift from the inside-out view of driving revenue through existing products and optimizing current processes to an outside-in perspective that aims to extend and optimize customer experience, while taking advantage of the digital supply chain's immediacy. APIs help enterprises overcome these mental barriers by providing a means to unbundle their business capabilities, then to identify gaps and opportunities. At its outset, an API program should include an assessment of current customer experiences and associated business capabilities. From there, organizations can build a roadmap of APIs that can be leveraged to extend the value proposition offered to customers or to unbundle business capabilities that can be used to enable new customer experiences.



Align Your Organization and Culture to the Strategy

With the customer context established, the API program should address the organizational dimension. That means designing a digital product organization with the agility to respond to volatile market dynamics. Given the prominence of software solutions in all digital products, services, and experiences, software delivery must be a core competency in any digital enterprise. The traditional enterprise model of having a separate IT team that takes orders from the business does not work in this context.

The divide between business and IT creates too many delays and too much distance between customer inputs and IT's technological understanding. Instead, organizations adopting APIs should have business domain experts and IT experts come together in small, product-oriented teams who maintain close ties to end customers.

An organization's common practices and patterns are also determinants of its ability to successfully launch and operate API-enabled digital products. These cultural elements can inhibit or empower the product team's ability to meet customer needs in a timely manner. An enterprise API program should establish and communicate a common cause and set of accompanying principles that can guide the organization's members without a high degree of coordination. Well-defined principles can help organizations shift their mental models from fear of change and fear of failure to an experimental culture of continuous delivery through small, low risk changes, the standard for the most successful API-enabled companies.



Evaluate and Build the Technologies You Need

A classic API program mistake is starting with the technology and trying to work backward to the business strategy. Instead, the business strategy and organizational approach should drive technology decisions. An enterprise's API program should assess the technological capabilities required to support its roadmap of APIs, from initial API products to the unfolding landscape of API management needs. Business priorities and organizational dynamics can also surface which technologies need to be standardized in order to facilitate interoperability, and which ones can be left to the choice of product teams.

Of particular concern are the technologies needed to ensure the safety of the enterprise and its customers. Security and privacy have not been the hallmark of many web native companies. As established companies enter the digital economy by opening up their data and services through APIs, it's important that they do not make the same mistakes as those pioneering startups. The API program needs to keep security as a top concern for its duration.



Engage the API Ecosystem

Enterprises can adjust their perspective from inside-out to outside-in by considering the ecosystem of developers, partners, platforms, tools, and other resources needed to connect customers to their core business capabilities via APIs. Treating APIs as digital products through design thinking and a well-defined business model is a recommended approach that helps flesh out each API's sustaining ecosystem elements.

The API's two key constituencies beyond customers — namely developers and partners — warrant special attention. An API is worthless without developers building the applications and systems of engagement that provide customer experiences. An API program should establish the API design principles and tooling that drive developer productivity. A [developer program](#) is a good way to develop a thriving developer community. Partners can be essential in delivering digital products to market. Imagine ride-sharing businesses Lyft and Uber trying to get started without the geolocation services of the Google Maps API, or Instagram driving and scaling its growth without Apple's App Store and AWS's underlying cloud service APIs. The API program should explore [patterns of API collaboration](#) early to identify capabilities that could be better delivered by third parties, or opportunities to power other companies' customer experiences.

Getting in Motion

APIs are useful tools for accelerating an organization's digital transformation. A strategic approach to API adoption must consider a wide range of organizational dimensions: from business models to cultural norms, from external entities to internal technologies. An enterprise API program clarifies the priorities and approaches in all of these areas. Although this holistic approach casts a wide net, thoughtful execution will ensure no fundamental aspect gets overlooked. The good news is that companies, such as health sciences enterprise [IQVIA](#), are already on the path to API-enabled digital transformation. With the API strategy blueprint as its guide, your organization can follow this path, too. 📦

How APIs Can Extend Your Business

By Aysegul Yonet, Senior Azure Cloud Developer Advocate at Microsoft

Today's applications are connected to 17 different third-party APIs (Application Programming Interfaces) on average. APIs are enabling companies to create complex applications faster than ever. An API simplifies programming by abstracting the underlying implementation and only exposing objects or actions the developer needs.

APIs are becoming more and more ubiquitous every day, powering websites, mobile applications, smart devices, and all kinds of other software products. There are many reasons for the popularity of APIs.

APIs are the next step in the evolution of the modularity of programming.

The first programs were written as a sequence of instructions that the computer had to execute from the beginning to the end. Later on, subroutines came along so you could group a set of instructions that are frequently called together and call that subroutine instead of repeating all those instructions. At this point, you generally had a good idea about how the subroutine worked, but did not want to repeat it. After subroutines, modules came along, which acted as a set of subroutines that you could import into your program. This opened up the possibility of sharing code between programmers who did not know each other. The internet supercharged this capability by providing unlimited access to software libraries that are shared publicly.

APIs are the next step in the evolution of software development. With APIs, we do not have to know anything about the code that is behind the functionality that the API provides. All we need to know is how to call the API, the effect of the call, and how to interpret the response we get. We do not need to know the implementation details or the programming language in which it is implemented.

The modularity that APIs provide has deep implications. We will examine these implications in three different scopes: private APIs, semi-private APIs, and public APIs. We will make a case for the importance of APIs in all three of these scopes.

Private APIs

By private APIs, we are referring to APIs that are developed and consumed in a closed ecosystem, such as a company. As companies grow, communication between different parts of the company becomes more difficult. Two different teams might need the same functionality without knowing about each other. A basic example can be getting a list of employees, their job titles, and other employee information. Let's assume this data exists in a database. These two teams can separately write software to connect to this database, combine data from multiple tables, clean it, and transform it into a good format. If a third team comes along with the same needs, they will have to write the same piece of software all over again, thus multiplying the effort and increasing the chance of errors and maintenance costs of the same piece of functionality.

TREND PREDICTIONS

- ▶ APIs represent the next step in the evolution of programming, shifting from code to implementing modular functions.
- ▶ The modularity of APIs will allow more and more people from diverse backgrounds and areas of expertise to create applications for their industries, broadening the possibilities of software development itself.

Another way to approach the same problem is by creating an API. When the first team needs the list of employees, they can expose the functionality in an API and, when another team needs it, they can simply call this API without having to know anything about the data's source or how the data was transformed. This avoids multiplying the engineering efforts, creating multiple sources of truth, and lowers maintenance costs, as there is only one codebase to maintain.

A similar solution is pulling common functionality into a library where other teams can import and reuse it. There are a few disadvantages to this approach: it requires downloading extra code, keeping track of updates in the library, and making sure the used version is up to date in case there are bug fixes. It also requires the library to be implemented in different languages. A well-defined API does not require downloading packages and updating versions for changes in the implementation that does not affect the use of the API. It is also independent from other implementation details, such as the programming language that powers the API. Any program that can make a call to the API takes advantage of it, regardless of the language in which it is written.

Semi-Private APIs

A variety of applications need the same data or the same functionality. Weather data or stock market data are basic examples of data that might be required by different applications. Sending a text message or performing a financial transaction are examples of functionality required by many companies.

Any program that can make a call to the API takes advantage of it, regardless of the language in which it is written.

We are referring to APIs that are accessible only to paid customers as semi-private APIs in this section. Building and exposing APIs that can be used by different parties is now a viable business plan for an increasing number of companies. This is possible due to the wide reach of the internet and the ability to find customers for more and more niche needs. For the API companies, this business model allows them to provide a software product that does not require a UI or UX design, but only a well-designed API. Therefore, companies can go to market for niche needs and with a small number of employees. For companies that pay for these APIs, the cost makes sense, as they are saving on paying for the human expertise, maintenance, and computational power that they would need to replicate the same functionality.

The financial markets have recognized the importance of API companies. In 2018, approximately \$1 billion was invested in API startups, double the amount from the previous year ([source](#)). There are estimates about the current size of the API economy ([source](#) and [source](#)) putting it between \$1 trillion to \$2 trillion. There have been numerous acquisitions of API companies such as SendGrid, MuleSoft, and Apigee.

Public APIs

Since there are so many public APIs, there is now a marketplace for sharing APIs called [RapidAPI](#). There are over 10,000 public APIs listed on RapidAPI. Although some of these APIs are for non-series purposes, for others, there is a good business case for companies to provide a public API.

One important case is a lack of human resources. If you are a public-facing startup with limited resources, it is not possible to build all the products that your users might demand. By exposing the APIs that you are using inside the company publicly, you can enable third-parties to build new features and applications for your customers. A good example is the social network Reddit. Reddit did not have official mobile clients for years, but independent developers have built more than one client for both iOS and Android



**1 Billion
Dollars**

**2018 Investment
in API Startups**

platforms, which has helped to increase the popularity of the website.

Another important use case is when Public APIs are used for the advancement of society. Public APIs provided by the government or civic organizations can be used to provide benefits to citizens, shed light on inefficiencies, and increase internal communication between government agencies. By their nature, government agencies and civic organizations have scarce resources. By investing in making their data available through public APIs, they can enable volunteers and other socially responsible organizations to use that data to benefit everyone. A high-profile example of such an effort is the [Open311](#) project.

When Would You Have an SDK Along With an API?

A software development kit (SDK) is a set of tools, utility functions, documentation, and code samples used to develop an application for a specific device or operating system. A Windows, Android, or iOS SDK includes components and functionalities that make up the building blocks of an app for that platform. SDKs allow rapid prototyping and a unified way to interact with platform APIs. SDKs are programming language-specific and create a file format that matches the platform for which you are building. When the operating system has an update, the accompanying SDK would have an updated version as well. Therefore, SDKs need to be developed for a specific operating system version and have to be maintained over time.


A REST API endpoint allows developers to use functionality with any programming language they want and not worry about the changes to the implementation of the functionality. If we can create an API endpoint and allow all of the languages, why would we ever want to create an SDK that is language-specific and maintain it? There are some cases where an SDK is beneficial when interacting with an API. By having components that make it easy to interact with an underlying API, SDKs make the development of an application fast and easy. For example, if you have a complex way of interacting with a platform's API, an SDK can abstract the interaction to a method you can easily call in your code.

Another good use case of an SDK is if you have complex data that needs to be sent to an API endpoint. An SDK can aggregate the data and reduce the cost of calling the endpoint. Similarly, SDK functions can format data in the way the REST API needs to receive it and perform error handling for you.

Finally, your users' privacy and data security might be a reason why you would want to support a REST API with an accompanying SDK. A great example is the Azure Spatial Anchors (ASA) SDK. Azure Spatial Anchors is a managed cloud service and developer platform that enables multi-user, spatially aware mixed reality experiences across HoloLens, iOS, and Android devices. A spatial anchor is a common frame of reference between the digital and physical world. A spatial anchor refers to a specific place in your environment, to which you can attach an object or piece of information. You can share the spatial anchor's location information by saving it in Azure Spatial Anchor Cloud Service, then send it between devices to create shared experiences.

There are two reasons why you'd use ASA SDK instead of only using an Azure API endpoint. The first one is efficiency; the data needs to be sent to create a spatial awareness of the user's environment. By aggregating the user's environment data into point clouds, ASA SDK reduces the data needed by the Azure API endpoints. Also, spatial awareness is created by using frames from a camera feed to map the user's environment. Sending the images of a user's environment would be a huge security issue. Instead, ASA SDK takes the camera frames and turns them into cloud points on the device. No camera frame leaves your device to be shared with the service endpoints. Additionally, ASA SDK translates the underlying platform with an Augmented Reality API, which is used for creating anchors, and translates them into a format that the ASA API endpoints expect. This translation utility allows developers to not worry about the platform-specific implementation details and easily share anchors between different platforms. Security and the performance of ASA services would not be possible without an SDK, while sharing between platforms would require a lot of implementation on the developer's side without it.

Conclusion

APIs allow a vast number of developers and creators to innovate using technologies with which they are unfamiliar. By making complex technologies easily consumable by many, APIs are opening the way for people with other areas of expertise to create the best applications for their field's needs. 

Key Research Findings

By **Jordan Baker**, Publications Associate at DZone

Demographics

For this Trend Report, we surveyed our members to learn the percentage of developers and the organizations who use APIs, how they plan their architecture strategies, and the integration approach they use, in general. Before we dive in, let's examine the demographics of those surveyed.

- ▶ Most respondents work for very large organizations.
 - 33% work for organizations sized <100.
 - 23% work for organizations sized 10,000+.
 - 20% work for organizations sized 1,000-9,999.
 - 14% work for organizations sized 100-499.
 - 9% work for organizations sized 500-999.
- ▶ Respondents' organizations work on three main types of applications.
 - 80% develop web applications.
 - 64% develop enterprise business applications.
 - 37% develop native mobile applications.
- ▶ Respondents' organizations tend to use four main programming language ecosystems.
 - 84% use Java.
 - 71% use client-side JavaScript.
 - 39% use Python.
 - 39% use Node.js (server-side JavaScript).
- ▶ A majority of respondents have over a decade's worth of experience.
 - 71% have worked in IT for 10 or more years.
 - 14% have worked in IT for 5-9 years.
 - 11% have worked in IT for 2-4 years.

TREND PREDICTIONS

- ▶ Simplicity will become an even more important factor in developers' individual use of APIs.
- ▶ The need to allow customers to integrate with their software will continue to drive organizational adoption of APIs.
- ▶ As the need to continuously deliver software grows in importance, pain points around documentation will worsen.

- Respondents reported filling three main roles within their organizations.
 - 35% work as developers.
 - 26% are architects.
 - 21% work as developer team leads.

Architectural Patterns and Practices

At the Developer Level

To understand how individual developers see and approach integration, we asked several questions about how they deal with issues that arise in this field. To begin, we asked respondents to delineate the most important factors to consider when approaching integration issues. Four main factors were reported by over 50% of survey takers: synchronous/asynchronous, data format, coupling, and simplicity.

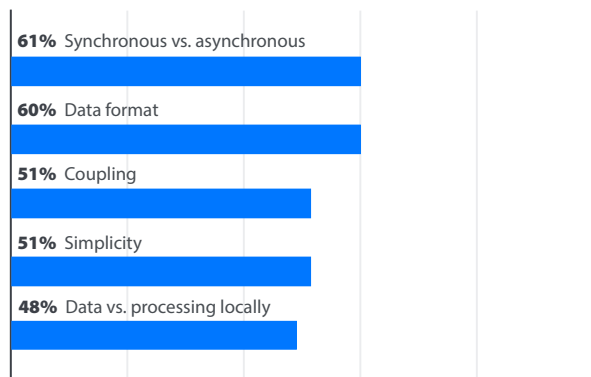
When we compare this data to the statistics presented in the demographics section above regarding the type of applications under development at respondents' organizations, we see some interesting variance in these percentages.

As we can see in Figure 2, while the general order of importance of these factors remained the same across different development efforts (with the exception of coupling in web application development), the percent of respondents reporting these factors as important varied depending on the type of application under development.

Furthermore, all four integration factors presented here proved more important to developers of web apps, enterprise business apps, and native mobile apps than among the general survey population.

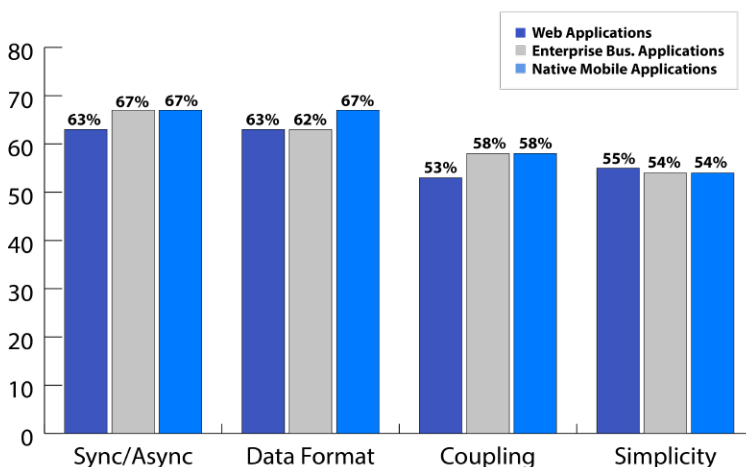
We also questioned respondents on the architectural qualities they deem most important. Again, there was no decisive winner here, as 71% of respondents reported scalability as an important architectural quality, followed by performance (67%), agility (66%), and loose coupling (66%).

Figure 1: Most Important Factors to Consider When Approaching Integration Problems.



Respondents could choose multiple answers.

Figure 2: Important Factors by Application

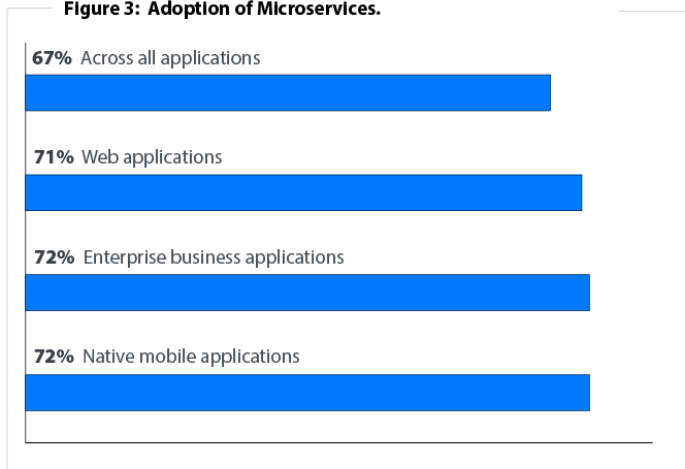


At the Organizational Level

To understand integration practices at the organizational level, we'll focus on how respondents told us their organizations approach the problems presented by integration. We started by simply asking respondents the purpose for their organization's use of integration. We received four main answers to this question: sending messages among applications (71%), sharing a common database among different applications (52%), transferring files among different applications (42%), and having one application invoke procedures in another (40%).

The importance of sending messages among applications makes sense when we consider another major current architectural movement: the development and use of microservices. In fact, 67% of respondents reported that their organizations currently use microservices and this is even higher among web (71%), enterprise business (72%), and native mobile applications (72%).

Figure 3: Adoption of Microservices.



Respondents could choose multiple answers.

This increasing adoption of microservices-based architecture could also explain why respondents, as outlined in the previous section, reported scalability (71%) and agility (66%) as two of the most important architectural qualities.

Interestingly, when we asked survey takers about their organization's biggest integration difficulties, two of the four main problems related more to quality software standards than technological barriers. Unclear requirements (51%) was the most popular response, with poor documentation (50%) as a close second. Some of the technological issues organizations tend to struggle with are different standards/interpretations between systems (reported by 50% of respondents) and propagating changes to integrated systems (reported by 42% of respondents).

APIs and RESTful Web Services

At the Developer Level

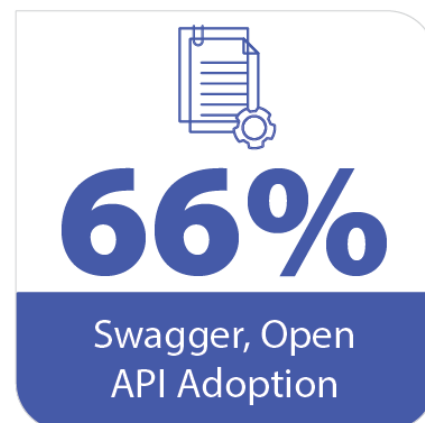
To get at how developers view APIs and RESTful web services, we asked our respondents what, in their opinion, are the most important technical aspects of building and using APIs. According to respondents, the factors developers deem important for both building and using APIs are more or less identical, save for the data around documentation. When building APIs, 62% of respondents told us documentation is important. When using APIs, 71% of respondents told us documentation is important. This difference is interesting and could point to a lack of documentation efforts in new development projects.

We also found that two main tools are used by those specifically designing and documenting for RESTful APIs. Swagger/OpenAPI was the undoubted winner, with a 66% adoption rating among respondents. Postman also proved rather popular, coming in as the runner-up behind Swagger/OpenAPI with an adoption rating of 45% among our survey population.

At the Organizational Level

When it comes to why organizations adopt the use of APIs, it typically revolves around allowing their clients to use the software they've purchased. In fact, 66% of respondents told us their organization implements APIs to "allow customers to integrate their software with ours." When we compare this to the data on application types currently under development at respondents' organizations, we find that allowing customers to integrate with their software was the number one driver of API adoption for organizations, regardless of the kind of application they are developing.

Some other important factors for API adoption were the need for mobile support (38%) and the desire to encourage users to "develop new solutions using our software" (35%). As the push for open source continues to gain steam, one should expect to see this rationale become more and more of an engine for implementing APIs at the organizational level.

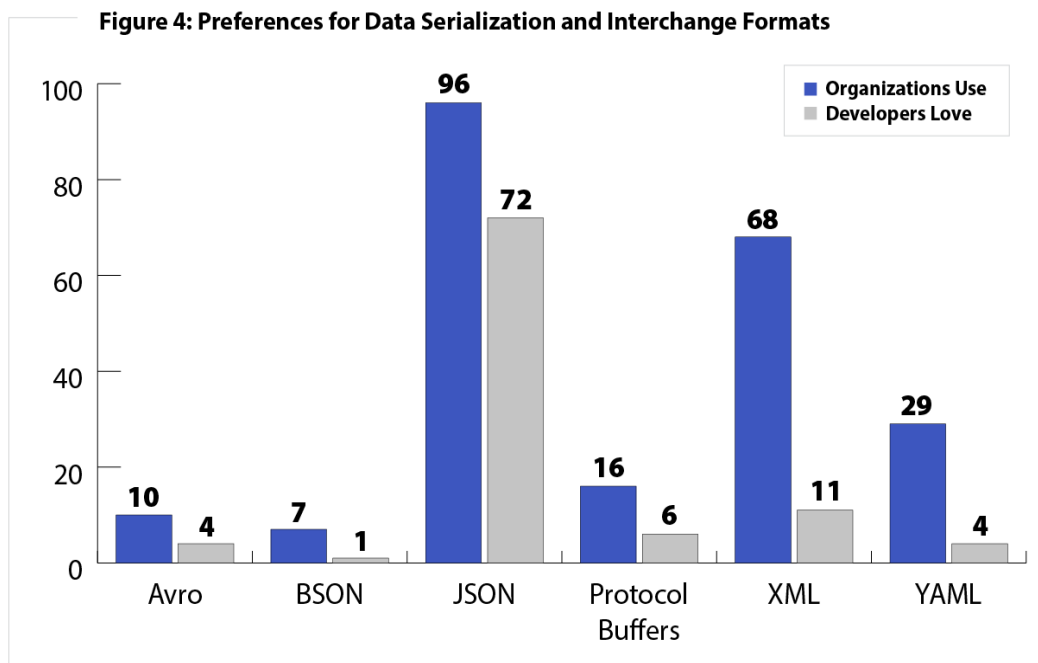


In addition to traditional APIs, we also asked respondents how often their organizations build and use REST APIs. 48% told us that they use REST whenever they can, and another 33% said they use REST when they can, with only a few exceptions. Putting those together, it seems safe to reason that 81% of respondents' organizations use REST APIs rather often. Also, we found that only 3% of respondents work for organizations who actively avoid REST APIs.

Tools and Technologies

At the Developer Level

We asked respondents to tell us which data serialization and interchange formats they use at work and which they actually enjoy using. Unsurprisingly, a disconnect appeared. At work, 96% of respondents use JSON, while 72% enjoy working with JSON.



For both use at work and enjoyment, JSON proved to be the most popular answer. For the second most popular answer, XML, there was a rather large gap that appeared between the stats for those who use it at work and those who enjoy using it. 68% of respondents report to use XML at work, but only 11% told us they enjoy working in this format. A similar, though not quite as large, difference appeared around the use of YAML. 29% said they use YAML at work, though only 4% claim to enjoy working with this data format.

Respondents reported four main types of systems they work to integrate: BI/analytics (47%); financial (43%); mobile (37%); CRM (37%). Interestingly, the adoption rates around the data serialization and exchange formats respondents claim to use for all these integration projects line up with the adoption rates among the general survey population outlined in the previous paragraph.

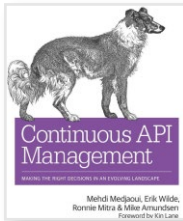
As we can see, the popularity of using XML varies based on the systems being integrated, but the general pattern (JSON as the most popular option, followed by XML and YAML) remains intact.

At the Organizational Level

While most respondents reported that their organizations were not planning on adopting new integration technologies, tools, or frameworks over the next 12 months, of those who did, the tools on their radar tended to be open source solutions. For integration frameworks, suites, or ESBs, 27% reported their organization plans on using Spring Integration, and 20% plan on using Apache Camel. For messaging tools, 47% said their organization plans on adopting Apache Kafka, while 31% are looking at Rabbit MQ. When we asked if respondents' organizations are planning on adopting an API management tool/framework over the next 12 months, 39% reported they will use Swagger, and 11% reported they will use Apigee. 🎲

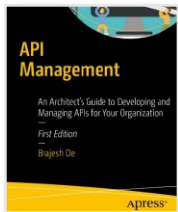
Diving Deeper Into API Management

Books

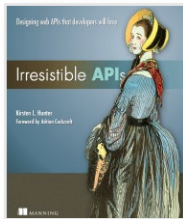


Continuous API Management

Learn what the “just-right” amount of planning looks like in preparation for an API release, with maturity models for each phase of your API landscape.



API Management The success of an organization’s API program depends on multiple factors. This book shows how to achieve effective API management at an enterprise scale.



Irresistible APIs This one is for the whole team. Enjoy this down-to-earth exploration of web APIs for all tech levels.

Refcards

API Integration Best Practices This Refcard serves as a starting point for best practices when approaching integration.

Hybrid Integration Best Practices In this Refcard, we’ll look at typical components of a HIP, as well as hybrid integration best practices, a roadmap for organizational buy-in, and more.

Java API Best Practices In this Refcard, you’ll learn how to make well-documented, consistent, evolvable APIs to help your users make the most of your Java applications.

Zones

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 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.

Microservices The Microservices Zone will take you through breaking down the monolith step-by-step and designing microservices architectures from scratch. It covers everything from scalability to patterns and anti-patterns. It digs deeper than just containers to give you practical applications and business use cases.

Podcasts

APIs You Won’t Hate Inspired by a blog, a book, and a Slack channel with the same name, this podcast takes an irreverent look at building and using APIs.

Integration Podcast An SAP integration expert discusses new and changing technologies with other industry pros.

AWS Podcast AWS is a huge part of the API world — this is a good way to keep up with the latest enhancements and updates.

Executive Insights on the State of API Management

By Tom Smith, Research Analyst at DZone

To gather insights on the current and future state of API management, we asked IT professionals from 18 companies to share their thoughts. Here's what they told us:

1. Companies primarily use APIs to access data, to develop internal and external applications, and to aid in microservices development. With more data and data sources, companies can create discrete applications and data sets, and expose them as a series of API-enabled services. APIs are also being used for both internal and external development, as companies begin to realize the value of reusing code — APIs power many companies' product ecosystems.

Legacy enterprises are also breaking monolithic applications into microservices, and APIs are the smallest "unit of compute" shared across monoliths, microservices, and serverless.

2. The most important elements of managing APIs are their protection, analytics, lifecycle management, and ease of use. It's important to enforce security policy management while also providing transparency. Rate limiting can control how many requests you have to handle. It can also keep malicious actors from damaging APIs or preventing their access by legitimate users.

Performance, reliability, and analytics are all important, as is runtime, security policy management, telemetry, monitoring, and consistent performance. API lifecycle management consists of defining, publishing, securing, routing, mediating traffic, monitoring, and analyzing performance.

It is essential that your API program provides a seamless and frictionless user experience. Every time you require users to make a change, they'll ask themselves why they are working with you versus a vendor that gets things right from the beginning.

3. APIs have made application development easier and more creative; they have facilitated the decomposition of monolithic apps into microservices and led to the development of more powerful mobile apps. Developers can now focus on the core capabilities of the product they are building, leveraging best-in-breed building blocks for other key (but not core) features. APIs reduce the complexity of the development process and enable developers to get closer to reusable code and components, enabling them to take off-the-shelf feature capabilities to compose new use cases.

TREND PREDICTIONS

- ▶ Companies primarily use APIs to access data, to develop internal and external applications, and to aid in microservices development.
- ▶ The most important elements of managing APIs are their protection, ease of use, analytics, and lifecycle management.
- ▶ Authentication is the most frequently mentioned way of securing APIs, followed by rate-limiting and more broad-reaching solutions.

The maturity of APIs and mobile development go hand-in-hand — behind every mobile app is an API. APIs externalize redundant or complex parts of applications. The massive shifts we're seeing on the infrastructure side, from Docker to Kubernetes, would not be possible without APIs, either. Microservices are designed to fit and mirror good RESTful API design.

4. Authentication is the most frequently mentioned way of securing APIs, followed by rate-limiting and more broad-reaching solutions. Several contributors suggested focusing on authentication and authorization. They recommended using OAuth and OAuth2 to communicate and secure communications between APIs.

Some developers use one-time secure token management and certification-based authentication. Others use rate-limiting API calls to mitigate distributed denial of service (DDoS) attacks. To secure the APIs themselves, you should consider applying a rate-limiting policy that sets a threshold on the number of requests the API gateway accepts each second.

Take a prescriptive approach to securing your API program. Think about how application identities are connected to user identities. Consider API security in its broadest sense — beyond authentication — to mitigate intrusion attempts. A multi-layered approach will include a web app firewall in a separate layer with Apache Mod security.

5. Consistent with expectations, API applications have a broad range of industry use cases. Those mentioned more than once are integration of business services and partners in industries like professionals services and transportation. Orchestrating real-time integration between endpoints to composing entire business solutions is one common use case.

6. Perhaps a function of the variety of use cases, the primary issues affecting APIs seem to be disaggregated. Contributors mentioned architecture, communications, complexity, and tools/standards, as well as several other topics.

Orchestrating real-time integration between endpoints to composing entire business solutions is one common use case.

7. Concerns were equally wide-ranging, though none were strikingly problematic. They included a lack of adoption of design-first methodology and the challenges of version management.

8. The future of API management is in the careful identification and management of the full API lifecycle, as well as in the continued adoption of API-driven microservices. Respondents saw improvements in the lifecycle of APIs as organizations start treating APIs as products with product managers guiding the lifecycle. Ultimately, many believe an integrated approach will make it easier for developers to design, implement, deploy, and manage API programs automatically.

Modernizing applications based on a microservices-based architecture is central to digital transformation initiatives. The greatest opportunity lies in API management of microservices — solutions with small footprints that are flexible, portable, and can operate in any infrastructure. There is a huge trend toward service mesh. It should be possible to take API management and bring it to every microservice. Ensure the ability for applications and microservices to talk to each other.

9. Developers are the consumers of APIs, so developers need to keep themselves — and other developers — in mind when building and managing APIs. What do you want to accomplish with the API? Why would someone use the API? What are the use cases? Achieve a balance between efficiency and human readability. Think in terms of a broad range of consumers as well as reusability. Follow best practices for development, including excellent documentation, understandable error messages, and predictable and consistent output and performance. Ensure consistency throughout the API ecosystem so consumers will feel like they're using a reliable collection of interfaces.

Consider the effect of microservices on API management. The majority of enterprises expect microservices to be the default

application architecture in five years. Build security into the architecture from the beginning, of course, and look for repeatable processes that can be automated.

Our thanks go out to these executives for sharing their insights with us:

- [Benoit Perrot](#), Director of Engineering, [Algolia](#)
- [Paulo Michels](#), EVP Engineering and Co-founder, [ArcTouch](#)
- [Mike Schuricht](#), VP Product Management, [Bitglass](#)
- [Ryan Breen](#), Director of API Management, [Cimpress](#)
- [Jorge Rodriguez](#), SVP Product Development, [Cleo](#)
- [Nick Chernets](#), Founder and CEO of [DataForSEO](#)
- [Amrit Jassal](#), CTO and Founder, [Egnyte](#)
- [Valery Novikov](#), Co-founder and CTO, [FI.SPAN](#)
- [Brian Platz](#), Co-CEO of [Fluree](#)
- [Manoj Chaudhary](#), CTO and SVP of Engineering, [Jitterbit](#)
- [Derek Smith](#), CTO and Co-founder, [Naveego](#)
- [Rob Whiteley](#), CMO, [NGINX](#)
- [Karthik Krishnaswamy](#), Senior Project Marketing Manager, [NGINX](#)
- [Mark Cheshire](#), Director Product Management, [Red Hat](#)
- [Cyril Nicodème](#), Founder, [Reflectiv](#)
- [Chetan Conikee](#), Founder and CTO, [ShiftLeft.io](#)
- [Idit Levine](#), CEO, [Solo.io](#)
- [Marc MacLeod](#), Founder and CEO, [Stoplight](#)
- [Rob Zazueta](#), Director of Digital Strategy, [TIBCO](#) 



INTRODUCING THE

Integration Zone

As software connects the world in more and more complex ways, integration makes it all possible by facilitating communication among applications. Learn more about this necessity for modern software development.

Keep a pulse on the industry with topics such as:

Enterprise apps and integration protocols
API management

Communication architectures
ESBs and SOAs

Visit the Zone



TUTORIALS



CASE STUDIES



BEST PRACTICES



CODE SNIPPETS