



Blogs de prueba de software a seguir en 2018

por Alex Jones · Ene. 07, 18 · DevOps Zone

¿Estás planeando extraer algunos microservicios de tu monolito? Lea esta guía gratuita para conocer las mejores prácticas antes de comenzar.

1. Guru99



Guru99 es uno de los mejores recursos de aprendizaje que debes seguir para mejorar tus habilidades. Tiene un gran depósito de tutoriales y conocimientos relacionados con la administración de pruebas, pruebas de rendimiento, pruebas empresariales, pruebas móviles, pruebas de seguridad y muchos otros temas. Marque esta página web ahora para subir de nivel sus habilidades. ¡Aprender nuevos cursos sobre la marcha nunca ha sido tan fácil!

2. Ayuda de prueba de software

Software Testing Help

Software Testing Help es uno de los sitios web más populares que se enfoca en temas de Pruebas de Software y Aseguramiento de la Calidad, encabezado por Vijay. Este blog es el mejor lugar para aprender y dominar tus habilidades de prueba. Cubre diversos temas sobre pruebas de automatización, pruebas manuales y pruebas web, plantillas de prueba, garantía

de calidad, certificaciones de prueba, libros, orientación profesional y tutoriales de prueba. También lo ayuda a estar actualizado con las tendencias, los últimos acontecimientos y las noticias.

3. Blog de Sticky Minds



Sticky Minds es una comunidad en colaboración con la comunidad de Techwell. Con Sticky Minds, la comunidad de usuarios puede ayudarlo a mantenerse al día sobre las tendencias de prueba, conferencias y capacitación a través de los artículos y eventos actualizados semanalmente. Y si tiene problemas con las pruebas, la comunidad interactiva de probadores lo ayudará a obtener la respuesta en la sección Preguntas y Respuestas. Regístrese en Sticky Minds ahora para interactuar con la comunidad y ampliar su conocimiento.

4. Clase de prueba de software



Software Testing Class es un sitio web completo para personas que prueban software para comenzar con útiles tutoriales, artículos, materiales de capacitación, cursos ... relacionados con diversos temas y categorías. Aquí puede encontrar casi todo lo que necesita sobre las pruebas de software.

5. Abode QA



Con la misión: traer más tutorial de calidad en línea. **AbodeQA** es el lugar donde puede encontrar un enorme repositorio de tutoriales, blogs, artículos sobre diversos temas como pruebas manuales, selenium-web-driver, tutoriales de Java, Katalon y más, y son totalmente gratuitos. Todo el contenido proviene de apasionados Profesionales del Software y, si lo desea, puede convertirse en el autor de este sitio para contribuir y compartir sus conocimientos.

6. Ejecutar Automatización



Execute Automation es el mejor sitio web, que proporciona conocimiento útil sobre herramientas y conceptos de pruebas de automatización. Las características que hacen que **Execute Automation** sea diferente de otros sitios web son simplicidad y conocimiento conciso sobre herramientas y procesos de prueba de automatización.

<http://executeautomation.com/blog/>

7. Aprender-Automatización

Learn-Automation es una comunidad técnica con **10000 miembros** y más de **20000 miembros** en las redes sociales.

Learn-Automation está dirigido por **Mukesh Otmani** en línea desde 2013 y es uno de los blogs populares en lo que respecta a los temas de **Selenium** o **Automatización**. El objetivo principal de hacer esto es llegar a su audiencia y compartir su talento, y lo más importante devolver algo a la **comunidad de automatización**.

<http://learn-automation.com/>

8. Joe Colantonio





Si busca un sitio web para descubrir más sobre: Reseñas de libros técnicos; Información sobre nuevas herramientas de prueba de código abierto; Crear y ejecutar pruebas de rendimiento exitosas; Tutoriales en video de automatización de pruebas; Consejos y trucos para usar herramientas de automatización de prueba como Selenium, soapUI y herramientas HP, incluidas UFT / QTP, ALM, LoadRunner y HP Diagnostics; Actualizaciones de HP y alertas de parches; Descubra cómo evitar las pruebas descartadas y los tutoriales de Automatización de prueba **100% gratuitos** y mucho más, ... Entonces este es el lugar correcto para usted.

<https://www.joecolantonio.com/>

9. Material de prueba de software



Como su nombre lo sugiere, el **material de prueba de software** es el sitio web al que todos los probadores de software profesionales pueden acceder a los materiales de prueba. Proporciona artículos útiles de última generación, mejores prácticas y otra información sobre las pruebas de software que son fáciles de entender para los profesionales de pruebas y personas no técnicas que desean elegir una carrera en Pruebas de software. Ayuda especialmente a los novatos que buscan un buen material de prueba de software. En este sitio, puede encontrar las publicaciones relacionadas con Manual, Automatización, Pruebas de rendimiento y también cubrimos Consejos de entrevista. Este es el sitio web que debe marcar en **2018** .

<http://www.softwaretestingmaterial.com/>

10. ToolsQA





ToolsQA es un blog de pruebas de automatización que contiene varios tutoriales sobre Selenium IDE, Katalon Studio, Cucumber, TestNG, Mobile Application Testing, Java, Appium, pruebas manuales de software, y más. Además de proporcionar tutoriales, también tiene una sección de Preguntas y respuestas para que los usuarios compartan los problemas y las dificultades en las pruebas de automatización y obtengan las respuestas de otros usuarios.

Este blog también ofrece artículos sobre pruebas de automatización, tutoriales de videos pagados y gratuitos y capacitación en línea, ... Este es el mejor sitio de aprendizaje paso a paso que debe seguir para acortar su tiempo de aprendizaje y evitar problemas que no necesariamente necesita para cara.

11. Software Test Pro



Software Test Professional sirve a la comunidad global de pruebas de calidad y control de calidad, brindando a más de 50,000 profesionales información, educación, capacitación y oportunidades de contactos profesionales.

Este sitio web ofrece valiosos libros blancos, conferencias, ofertas de capacitación y otros eventos de redes enfocados en las necesidades de los probadores de software y profesionales de control de calidad.

Software Test Pro se mantiene al tanto de las tendencias que configuran el futuro de la industria con excelentes recursos para los miembros. Visite este sitio web y explore nuevos conocimientos en la industria de pruebas de software.

12. Software Testing Club



Software Testing Club es una comunidad global y profesional para probadores de software. Con más de

11.400 usuarios y casi 500 temas, es la plataforma líder que contiene una amplia gama de categorías, como Cool Projects: para que las personas de la comunidad compartan herramientas increíbles y el proyecto en el que han estado trabajando o Eventos: un lugar para hablar todos esos eventos de prueba y mucho más.

Y si está buscando un sitio web para mantenerse actualizado con las últimas noticias sobre pruebas de software, **Software Testing Club** es el lugar adecuado para usted.

13. El ojo de prueba



The Test Eye está dirigido por tres expertos profesionales, Martin Janson, Henrik Emilsson y Rikard Edgren. Son antiguos colegas que comenzaron a trabajar con pruebas de software a fines de los 90, por lo que son expertos en este campo.

El blog se centra en temas sobre ideas de prueba, pruebas con guiones, software y pruebas ágiles, ideas de prueba, técnicas de prueba y mucho más. Además, la sección de recomendaciones te dará consejos sobre habilidades, personas, máquinas, ideas y documentación. Pero como se actualiza una vez al mes, no tiene que visitarla regularmente.

14. Trucos de prueba de software



Software Testing Tricks es el blog de prueba de software para probadores nuevos y experimentados, y este sitio está dedicado a ayudar a los evaluadores a aprender las habilidades de las pruebas, compartir sus

experiencias. Este blog es propiedad de Debasis Pradhan, un Tester con más de 25 años de experiencia. Software Testing Tricks proporciona consejos y trucos para los evaluadores de software e incluye una amplia gama de temas como herramientas de automatización, preguntas de entrevistas, tutoriales, ... Este es uno de los blogs útiles que debe visitar con regularidad.

15. Blog de James Bach



El **blog de James Bach**, también conocido como **Satisfice**, es un sitio web fundado y operado por un experto en el campo de las pruebas de software, James Marcus Bach. Satisfice se dedica a la enseñanza y la consultoría en pruebas de software y aseguramiento de la calidad. Este blog cubre muchos temas como pruebas ágiles, exploratorias, automatización y mucho más. Además, los artículos en este blog son muy interesantes y ricos en contenido. Ve a este blog y mejora tus habilidades.

16. Prueba de control de calidad de software

The logo for SoftwareQA Test.com is displayed in white text on a dark blue rectangular background.

© 1996-2017 by [Rick Hower](#)

A diferencia de otros sitios web, **Software QA Test** no es el lugar donde puede encontrar artículos informativos y debates sobre Pruebas de software. En su lugar, encontrará las preguntas más frecuentes sobre control de calidad de software y pruebas, las respuestas a cada pregunta. Además, en las secciones de **Recursos y Herramientas**, encontrará una gran cantidad de recursos sobre QA de Software y Pruebas y la información detallada sobre las herramientas. Y si está buscando trabajo, también hay una lista de bolsas de trabajo en la sección de empleo. Marque esta página web ahora para explorar secciones más útiles.

17. Revista de pruebas de

software

Software Testing Magazine

Software Testing Magazine es un sitio web gratuito dedicado a presentar artículos, tutoriales, publicaciones de blog, reseñas de libros, herramientas, noticias y videos y otros recursos sobre pruebas unitarias, pruebas de integración, pruebas funcionales o de aceptación, pruebas de carga o rendimiento en proyectos de desarrollo de software.

El blog se actualiza semanalmente e incluye muchas secciones, como herramientas, conocimientos, noticias, eventos, ... para que pueda recibir actualizaciones, no solo las últimas noticias e información sobre las herramientas y las tendencias de prueba, sino también las próximas pruebas de software. eventos en todo el mundo.

Esta es solo mi lista de favoritos, así que si me perdí tus blogs favoritos, puedes dejar un comentario y compartir con nosotros tus favoritos.

Mida el impacto de los lanzamientos de características y de su programa DevOps de manera más amplia con la experimentación de pila completa. Aprenda cómo con este artículo gratuito de CITO Research, proporcionado por Split Software.

Temas: PRUEBAS DE SOFTWARE, DEVOPS, QA

Las opiniones expresadas por los contribuidores de DZone son suyas.

Obtenga lo mejor de DevOps en su bandeja de

entrada.

Manténgase actualizado con el boletín DevOps quincenal de DZone. VER UN EJEMPLO

SUSCRIBIR

Recursos para socios de DevOps

Elegir entre las herramientas de administración de registros

Scalyr



Continúe su transformación digital con un Blueprint para entrega continua.

Atomic



Redefina los procesos de liberación de aplicaciones y unifique su estrategia de DevOps.

Atomic



Cómo extraer un servicio con estado de un monolito

División



Benefits of REST APIs With HTTP/2

by Guy Levin MVB · Jan 09, 18 · Integration Zone

Modernize your application architectures with microservices and APIs with best practices from this free virtual summit series. Brought to you in partnership with CA Technologies.

HTTP/1.x vs HTTP/2

First, let's look at some of the high-level differences:

HTTP/2 Is Binary, Instead of Textual

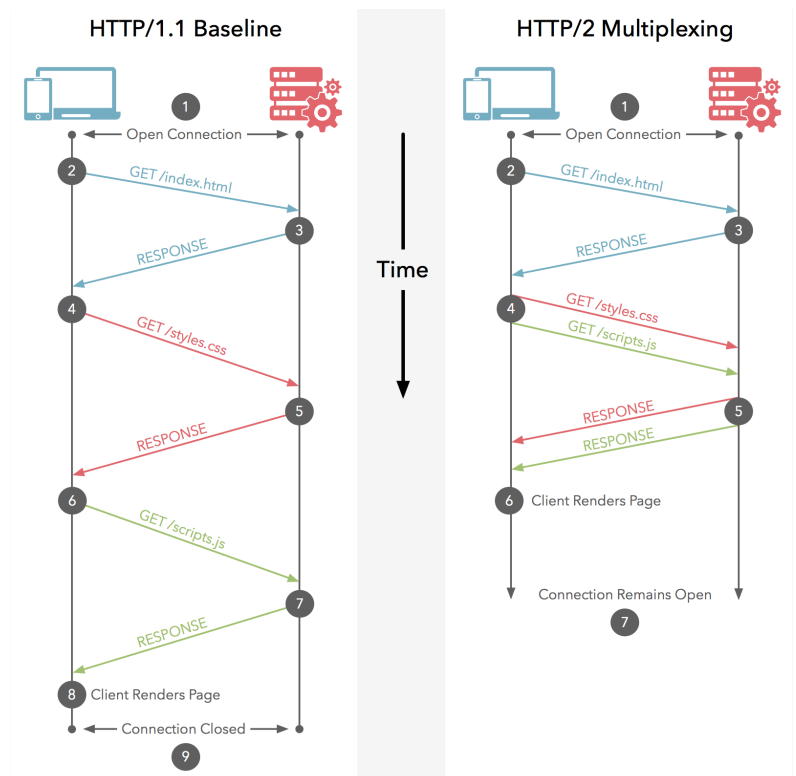
Binary protocols are more efficient to parse, more compact "on the wire," and, most importantly, they are much less error-prone compared to textual protocols like HTTP/1.x, because they often have a number of affordances to "help" with things like whitespace handling, capitalization, line endings, blank lines, and so on.

For example, HTTP/1.1 defines four different ways to parse a message; in HTTP/2, there's just one code path.

HTTP/2 Is Fully Multiplexed, Instead of Ordered and Blocking

HTTP/1.x has a problem called "head-of-line blocking," where effectively only one request can be outstanding on a connection at a time.

HTTP/1.1 tried to fix this with pipelining, but it didn't completely address the problem (a large or slow response can still block others behind it). Additionally, pipelining has been found very difficult to deploy, because many intermediaries and servers don't process it correctly.



This forces clients to use a number of heuristics (often guessing) to determine what requests to put on which connection to the origin and when; since it's common for a page to load 10 times (or more) the number of available connections, this can severely impact performance, often resulting in a "waterfall" of blocked requests.

Multiplexing addresses these problems by allowing multiple request and response messages to be in flight at the same time; it's even possible to intermingle parts of one message with another on the wire.

This, in turn, allows a client to use just one connection

This, in turn, allows a client to use just one connection per origin to load a page.

HTTP/2 Can Use One Connection for Parallelism

With HTTP/1, browsers open between four and eight connections per origin. Since many sites use multiple origins, this could mean that a single page load opens more than thirty connections.

One application opening so many connections simultaneously breaks a lot of the assumptions that TCP was built upon; since each connection will start a flood of data in the response, there's a real risk that buffers in the intervening network will overflow, causing a congestion event and retransmits.

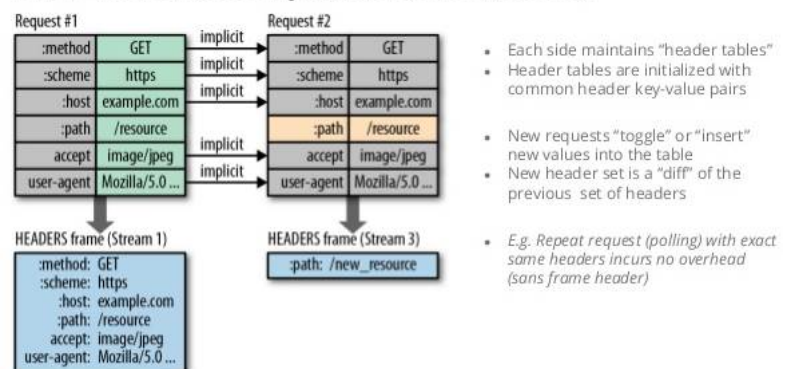
**You can see a demo of how
HTTP/2 is working here:
<https://http2.akamai.com/demo>**

Additionally, using so many connections unfairly monopolizes network resources, "stealing" them from other, better-behaved applications (e.g., VoIP).

HTTP/2 Uses Header Compression to Reduce Overhead

If you assume that a page has about 80 assets (which is conservative in today's Web), and each request has 1400 bytes of headers (again, not uncommon, thanks to Cookies, Referer, etc.), it takes at least 7-8 round trips to get the headers out "on the wire." That's not counting response time - that's just to get them out of the client.

HTTP 2.0 header compression (in a nutshell)



This is because of TCP's Slow Start mechanism. which

paces packets out on new connections based on how many packets have been acknowledged - effectively limiting the number of packets that can be sent for the first few round trips.

In comparison, even a mild compression on headers allows those requests to get onto the wire within one roundtrip - perhaps even one packet.

This overhead is considerable, especially when you consider the impact upon mobile clients, which typically see round-trip latency of several hundred milliseconds, even under good conditions.

HTTP/2 Allows Servers to "Push" Responses Proactively Into Client Caches

When a browser requests a page, the server sends the HTML in the response and then needs to wait for the browser to parse the HTML and issue requests for all of the embedded assets before it can start sending the JavaScript, images, and CSS.

Server Push potentially allows the server to avoid this round trip of delay by "pushing" the responses it thinks the client will need into its cache.

However, pushing responses is not "magical" - if used incorrectly, it can harm performance. For now, many are still will continue to work with Webhooks.

How Does This Affect the Existing REST APIs Built on HTTP/1.1?

The main semantic of HTTP has been retained in HTTP/2. This means that it still has HTTP methods such as GET , POST , HTTP headers , and URIs to identify resources.

What has changed in HTTP/2 with respect to HTTP/1.1 is the way the HTTP semantic (e.g. "I want to PUT resource /foo on host domain.com") is transported over the wire. This means that **REST APIs built on HTTP/1.1 will continue to work transparently as before, with no changes to be made to applications.**

The web container that runs the applications will take care of translating the new wire format into the usual

care of translating the new wire format into the usual HTTP semantic on behalf of the applications, and the application just sees the higher level HTTP semantic, no matter if it was transported via HTTP/1.1 or HTTP/2 over the wire.

Because the HTTP/2 wire format is more efficient (in particular due to multiplexing and compression), REST APIs on top of HTTP/2 will also benefit from this.

The other major improvement present in HTTP/2, HTTP/2 Push, targets the efficient downloading of correlated resources, and, as it's probably not useful in most REST API use cases, perhaps only the Object Storage like services can benefit from this (like Amazon S3).

A typical requirement of HTTP/2 is to be deployed over TLS. This requires deployers to move from HTTP to HTTPS which means buying SSL certificates from a trusted authority, etc.

HTTP/2 Benefits Explained

Among the key improvements brought by HTTP/2 are multiplexed streams, header compression, server push, and a binary protocol instead of textual one. These and other positive changes allowed to achieve good web pages loading results, including those having lots of additional files attached to them (e.g. styles, scripts, images, fonts, etc.).

HTTP/2

1. One TCP connection

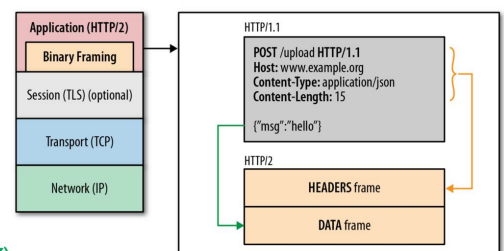
2. Request → Stream

- Streams are multiplexed
- Streams are prioritized

3. Binary framing layer

- Prioritization
- Flow control
- Server push

4. Header compression (HPACK)



HTTP/2, the new version of the HTTP protocol, provides also a lot of new features for server-to-server communication:

Bidirectional Communication Using Push Requests

HTTP/2's "server push" allows a server to proactively send things to the client's cache for future use.

This helps avoid a round trip between fetching HTML and linked stylesheets and CSS, for example; the server can start sending these things right away, without waiting for the client to request them.

It's also useful for proactively updating or invalidating the client's cache, something that people have asked for.

Of course, in some situations, the client doesn't want something pushed to it - usually because it already has a copy, or knows it won't use it. In these cases, it can just say "no" with RST_STREAM.

Multiplexing Within a Single TCP Connection

HTTP/2 uses multiplexing to allow many messages to be interleaved together on a connection at the same time so that one large response (or one that takes a long time for the server to think about) doesn't block others.

Furthermore, it adds header compression, so that the normal request and response headers don't dominate your bandwidth - even if what you're requesting is very small. That's a huge win on mobile, where getting big request headers can easily blow out the load time of a page with a lot of resources by several round trips.

Long Running Connections

HTTP/2 is designed to use fewer connections so servers and networks will enjoy less load. This is especially important when the network is getting congested because HTTP/1's use of multiple connections for parallelism adds to the problem.

For example, if your phone opens up six TCP connections to each server to download a page's resources (remembering that most pages use multiple servers these days), it can very easily overload the mobile network's buffers, causing them to drop packets, triggering retransmits, and making the problem even worse.

HTTP/2 allows the use of a single connection per host and encourages sites to consolidate their content on one host where possible.

Stateful Connections

If your HTTP/1 client sends a request and then finds out it doesn't need the response, it needs to close the

connection if it wants to save bandwidth; there's no safe way to recover it.

HTTP/2 adds the RST_STREAM frame to allow a client to change its mind; if the browser navigates away from a page, or the user cancels a download, it can avoid having to open a new connection without wasting all of that bandwidth.

Again, this is about improving perceived performance and network friendliness; by allowing clients to keep the connection alive in this common scenario, extra roundtrips and resource consumption are avoided.

But...

As always, not everything is about benefits, there are some questionable downsides:

Using Binary Instead of Text

This is also a good and a not so good feature.

One of the nice things about HTTP/1 is the ability to open up telnet, type in a request (if the server doesn't time out!) and then look at the response. This won't be practical in HTTP/2 because it's a binary protocol. Why?

Consider how can we store short int 30000 (0x7530), both as text and as binary:

– One option is to store the number in text form as chars '3', '0', '0', '0', '0'

- Using ASCII chars, we need 5 bytes to store this number

Byte #	0	1	2	3	4	
	'3'	'0'	'0'	'0'	'0'	5 bytes long

- The other option is to store the number in binary, which would take as few as 2 bytes

Byte #	0	1	
	0x30	0x75	2 bytes long

As you can see, instead of using 5 bytes we are using 2 bytes. It is more than 50% size reduction.

While binary protocols have lower overhead to parse, as well as a slightly lighter network footprint, the real reason for this big change is that binary protocols are simpler and therefore less error-prone.

That's because textual protocols have to cover issues like how to delimit strings (counted? double-newline?), how

to handle whitespace, extra characters, and so on. This leads to a lot of implementation complexity; in HTTP/1, there are no fewer than three ways to tell when a message ends, along with a complex set of rules to determine which method is in use.

HTTP/1's textual nature has also been the source of a number of security issues; because different implementations make different decisions about how to parse a message, malicious parties can wiggle their way in (e.g., with the response splitting attack).

One more reason to move away from text is that anything that looks remotely like HTTP/1 will be processed as HTTP/1, and when you add fundamental features like multiplexing (where associating content with the wrong message can have disastrous results), you need to make a clean break.

Of course, all of this is small solace for the poor ops person who just wants to debug the protocol. That means that we'll need new tools and plenty of them to address this shortcoming; to start, Wireshark already has a plug-in.

More Encryption

HTTP/2 doesn't require you to use TLS (the standard form of SSL, the Web's encryption layer), but its higher performance makes using encryption easier since it reduces the impact on how fast your site seems. This means that you will probably need to buy SSL certificates, renew them, etc. This is not a small amount money to spend when you are working with many microservices using REST APIs.

In fact, many people believe that the only safe way to deploy the new protocol on the "open" Internet is to use encryption; Firefox and Chrome have said that they'll only support HTTP/2 using TLS.

They have two reasons for this. One is that deploying a new version of HTTP across the Internet is hard, because a lot of "middleboxes," like proxies and firewalls, assume that HTTP/1 won't ever change, and they can introduce interoperability and even security problems if they try to interpret an HTTP/2 connection.

The other is that the Web is an increasingly dangerous place, and using more encryption is one way to mitigate

place, and using more encryption is one way to mitigate a number of threats. By using HTTP/2 as a carrot for sites to use TLS, they're hoping that the overall security of the web will improve.

Summary

The real benefit to your existing REST APIs will be if most of your microservices that are probably REST based are working server to server communication. In today's microservices architecture, when many microservices are talking between themselves in many ways but still using REST, HTTP/2 can increase the speed of your workflows.


HTTP/2 does not define a JavaScript API nor does it help you build your REST APIs much more easily. For now, JavaScript clients running in a web browser can make only limited use of the new capabilities. However, for server-to-server communication, HTTP/2 provides a lot of ways to go beyond existing REST APIs.

Furthermore, the downside of HTTP/2's network friendliness is that it makes TCP congestion control more noticeable; now that browsers only use one connection per host, the initial window and packet losses are a lot more apparent.

Just as HTTP has undergone a period of scrutiny, experimentation, and evolution, it's becoming apparent that the community's attention is turning to TCP and its impact upon performance; there's already been early discussion about tweaking and even replacing TCP in the IETF.

The Integration Zone is proudly sponsored by CA Technologies. Learn from expert microservices and API presentations at the Modernizing Application Architectures Virtual Summit Series.

Topics: INTEGRATION, REST APIS, HTTP 2, API DEVELOPMENT

Published at DZone with permission of Guy Levin, DZone MVB. [See the original article here.](#) 
Opinions expressed by DZone contributors are their own.

