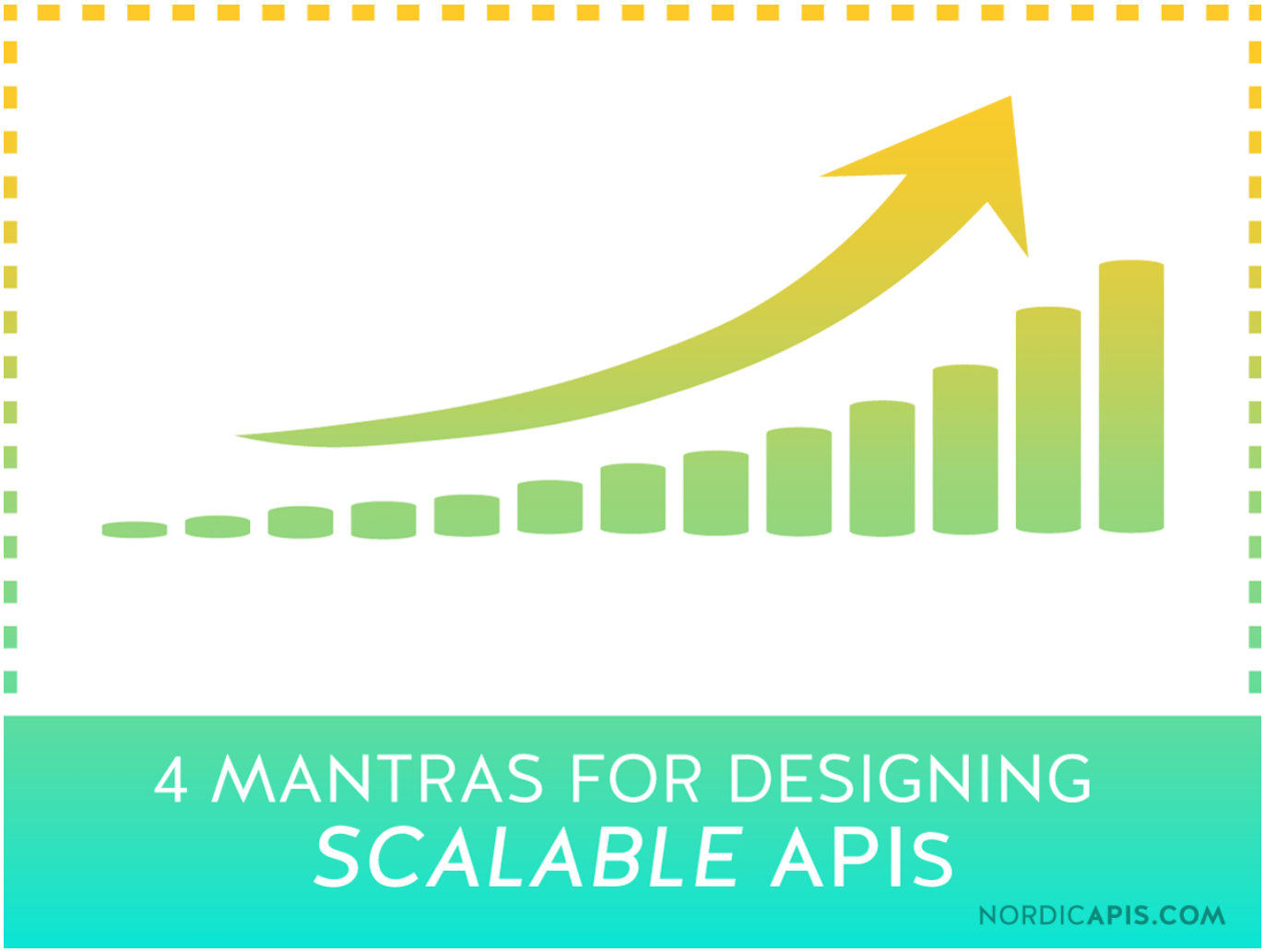
4 Mantras for Designing Scalable APIs (https://nordicapis.com/4-mantras-for-designing-scalable-apis/)

Kristopher Sandoval(https://nordicapis.com/author/sandovaleffect/)

June 20, 2017

(https://www.facebook.com/sharer.php?u=https%3A%2F%2Fnordicapis.com%2F4-mantras-for-designing-scalable-apis%2F) (https://twitter.com/intent/tweet?url=https%3A%2F%2Fnordicapis.com%2F4-mantras-for-designing-scalable-apis%2F&text=4+Mantras+for+Designing+Scalable+APIs) (https://www.linkedin.com/shareArticle?url=https%3A%2F%2Fnordicapis.com%2F4-mantras-for-designing-scalable-apis%2F&title=4+Mantras+for+Designing+Scalable+APIs)



Scalability might be a buzzword, but there's reason for that – it can define both early adoption and future success, as well as a variety of points and approaches along the lifespan of a product. That being said, it's also often poorly implemented.

Properly approaching scalability is important not only to nascent solutions in the industry, but to the industry itself. As applications grow larger and more demanding, processing ever larger amounts of data, scalability is going to become vastly more important than it already is.

Today, we're going to discuss scalability, and what it actually means. We'll discuss why scalability is so incredibly important for designing robust **APIs** and **microservices**, and consider the implications of proper scalability on the web industry as a whole. Finally, we'll provide **four mantras**; four basic and repeatable concepts that will help any provider adopt scalability and reap its rewards.

What Does *Scalable* Mean?

There's been a lot of lip service to the idea of scalability, and it often comes with a selling point. "Make your API scalable by tying into our simple API" or "you too can make your service scalable by licensing our endpoint collating system" are common sales pitches, but they're unfortunately misleading.

If you're tying into a magic service to make scalability happen, you're likely not being scalable – there's no silver bullet, and adopting a quick fix doesn't address **underlying architectural deficiencies**. Scalability itself should occur at the **base level** of the application, and is much more a design ethos rather than any sort of set architecture or solution.

Broadly speaking, scalability can be defined as thus:

- **Extensible**: Scalable software is extensible at the most basic level. The design itself does not limit functionality, and instead allows multiple avenues to tie into the underlying services and systems to enable extensions and other services. An example is Amazon's API Gateway.
- **Built into the architecture**: Scalability must be imbued into the foundational stages of the API itself. To put it simply, don't build then adopt scalability later on, ***build for scalability from the onset***. Third party solutions ***can*** be valuable for increasing scalability, especially when they assist a built-to-scale approach.
- **Implies demand balancing**: Just as important as extensibility is responding to demand. Scalability implies that the handling of traffic is done just as well with one hundred requests as with one million – it's in the name, "scale." The serious implication is that scalability naturally demands efficient performance, regardless of technique or methodology, at both extremely high traffic and extremely low traffic.

Four Mantras for Scalability

Now that we have a better understanding of what scalability means, here are four mantras to live by when designing scalable APIs.

1: Design for A Repeating Launch Day

The day a product launches is perhaps one of the most stressful days in the lifespan of development for a variety of reasons. One of these reasons is the fact that no matter how good your market planning, how effective your outreach, you simply cannot know the **requirements** your system might see.

When launching a service, what kind of traffic can we expect? Let's say we're launching a new social platform that ties into an API to join social profiles into a single "hub." So far, so good. We've designed this API to handle a certain volume of calls – for simplicity's sake, let's say 200k calls an hour is the maximum our server can handle.

The problem is that we don't know exactly what our system is going to have to deal with. Our hub could be featured in a Reddit post, shared on Facebook, mentioned in a YouTube video, and before you know it, our 200k an hour call rate has ballooned to several millions of calls.

Application

What does this mean for scalability? You need to **address the foundation of your API as if you are always on the verge launching**. Ensuring that you are using a proper **load balancer** is hugely important, and in many cases, can be the difference between success and failure. Ensuring that your API has **failover paths** and secondary functions can make a huge difference not only to your API's usability, but towards general user experience.

In fact, in some cases the server issues can be skipped entirely. If we're truly concerned with scalability and we anticipate that our service is going to have extreme fluctuations in traffic, we can decide to simply go for a serverless solution like Amazon Web Services. By tying into an extensible system and enabling our traffic to be off-loaded via hardcoded routes in the API architecture, we can dynamically scale to meet almost any demand – in essence, we can expect the unexpected.

Finally, one of the best ways you can plan for launch is to integrate analytics into your system. Launch day is a mystery, and there will never be a perfect prediction as to what it will mean for a product – that being said, you're essentially playing an information game. And as such, any edge you can give yourself is valuable. Being able to see trends developing in real time can help you develop in an agile way and address deficiencies as they arise organically, while predicting further failures down the road.

2: Anticipate Success

Speaking of "expecting the unexpected," you never truly know how successful you're actually going to be. This isn't a simple consideration of traffic, either – traffic might be high even if you're the second or third most popular choice.

The fact is that a service might easily be the number one service in a matter of days, regardless of what the expectation is. Instances of the "Reddit Hug of Death", which was previously called the "Slashdot Effect", can instantly inundate sites with dramatic amounts of traffic, causing them to crash and burn. It is this "crashing and burning" that can prematurely kill the cost benefit argument for a service, and can directly cause loss of retention.

Simply put, a provider cannot know how much traffic they can expect, and thus when planning in scalability terms, providers should **plan for the most extreme case possible** within their means in order to enable the handling of these extremes. Another way to frame this would be to *anticipate success*.

Application

An API provider that under-builds could be dramatically stunting potential growth. Ensuring that an API has the proper ability to route traffic is one thing, but under-building the underlying architecture to such a point that success is self-restricted is another thing entirely.

How can we stop this from being a problem? First and foremost, building out support for alternative data formats and adopting additional language implementations is incredibly helpful. Not everybody is going to a JSON fan, and not everybody is going to want to use Ruby. Easing implementations in both common and esoteric languages opens your system to further development and iteration to match a constantly evolving industry.

3: Non-Extensible is a Unitasker

Traffic management and scalable mindset is all for naught if the application itself is not **extensible**. While extensibility is indeed its own concept, with its own considerations and implications, whether or not a service is extensible can have a direct impact on whether or not it's scalable.

While it's a great practice to develop with scalability from the onset, there's no way that a provider can know literally every possible future use and application of their service.

Application

Thankfully, developers don't need to be clairvoyant. When a system is properly designed with scalability in mind, functions should be **interrelated** as much as they are interconnected and interfaceable. That sounds like a bunch of buzzwords, but we can see this exact type of solution in something like GraphQL.

In GraphQL, you get predictability by allowing users to define what they want, and get exactly that. While that's helpful, that's only the tip of the iceberg in terms of extensibility. Because GraphQL ties into the underlying structure of the API and its resources, many resources can be called in a variety of ways using a single call. What this means is that a system like GraphQL, compared to a traditional API, is better able to handle a variety of requests that the developer may never have anticipated.

Adopting extensibility adds **value** to your service. While many niche web services derive their power from specialization, making a truly extensible application, with the added benefit of being scalable to boot, is supremely valuable and grants amazing staying power.

4: Efficiency Is King

You cannot always decrease the complexity of the problem – your traffic will always be your traffic, and use cases will remain the same. What you can do, however, is simplify your API architecture and thus simplify the resultant solution applied to the problem. By increasing **efficiency**, you can drastically reduce the actual resources needed by an application.

The issue with scalability is that, in classic conception, an increase in power necessitates an increase in methodology. In the physical world, to carry more people, you need more buses, more taxis, etc. On the internet, in order to carry more data and do more calculations, you need more advanced protocols.

Application

A great example of this is the rejection of polling in favor of something like REST Hooks. According to REST Hooks developers Zapier, "On average, 98.5% of polls are wasted." All of this polling is simply wasted processing power, and can drastically contribute to increased API overhead, thereby limiting the processing available to the actual functionality of your API.

Why is Scalability Important?

Finally, it's important to contextualize exactly why scalability important. It sounds good in analogy form, but is this actually a significant enough issue to demand best practices? Well, consider this data supplied by HighScalability.com:

- Netflix has 100 million subscribers;
- 25% of US citizens won't subscribe to traditional cable, favoring streaming solutions;
- The largest Google Computer Engine job utilized 220,000 high-throughput cores;
- Sling TV has 1.3 million subscribers; and
- There are 10^5 data centers worldwide needed for cloud computing.

Why is this data important? All of these data points are from services that started out small, and grew **exponentially larger**. Netflix started as a relatively small service, and grew extremely fast. Google started essentially as a small experiment, and became one of the largest organizations in the world. Sling TV had a small subscriber base, and has considerably and consistently grown.

That the smallest, single-purpose solution is not going to stay that way forever. Designing for the traffic and function that you currently handle is fine, but intrinsically limits the future growth of the platform.

Conclusion: Scalability is Critical For API Platform Growth

Hopefully these mantras have helped establish a strong, long-term ethos for scalable API design. It should be apparent that scalability is all but a requirement for modern applications, especially in the current climate where an application can rush from relatively unknown to heavy use in a matter of hours. Failing to match modern standards means death, and even the best laid plans are essentially useless if the underlying system itself is not scalable.

This post is older than 4 years. External links has been removed

2017 (https://nordicapis.com/tag/2017/), Amazon (https://nordicapis.com/tag/amazon/), API design (https://nordicapis.com/tag/api-design/), APIs (https://nordicapis.com/tag/apis/), architecture (https://nordicapis.com/tag/architecture/), AWS (https://nordicapis.com/tag/aws/), demand (https://nordicapis.com/tag/demand/), Design (https://nordicapis.com/tag/design/), extensibility (https://nordicapis.com/tag/extensibility/), extensible (https://nordicapis.com/tag/extensible/), gateway (https://nordicapis.com/tag/gateway/), growth (https://nordicapis.com/tag/growth/), load balancer (https://nordicapis.com/tag/load-balancer/), load balancing (https://nordicapis.com/tag/load-balancing/), mantra (https://nordicapis.com/tag/mantra/), mantras (https://nordicapis.com/tag/mantras/), microservices (https://nordicapis.com/tag/microservices/), platfom (https://nordicapis.com/tag/platfom/), Platform Summit (https://nordicapis.com/tag/platform-summit/), platforms (https://nordicapis.com/tag/platforms/), scalability (https://nordicapis.com/tag/scalability/), scalable (https://nordicapis.com/tag/scalable/), scale (https://nordicapis.com/tag/scale/), serverless (https://nordicapis.com/tag/serverless/), traffic (https://nordicapis.com/tag/traffic/), web (https://nordicapis.com/tag/web/), web development (https://nordicapis.com/tag/web-development/)

Be the first to comment (https://nordicapis.com/4-mantras-for-designing-scalable-apis/#disqus_thread)

(https://nordicapis.com/author/sandovaleffect/)

Kristopher Sandoval

(https://nordicapis.com/author/sandovaleffect/)
Kristopher is a web developer and author who writes on security and business. He has been writing articles for Nordic APIs since 2015.

(https://www.linkedin.com/in/kristophersandoval/)

Best Practices for API Error... (https://nordicapis.com/best-practices-api-error-handling/)

Why Do FinTechs Want To Save... (https://nordicapis.com/fintechs-want-save-screen-scraping/)

0 Comments Nordic APIs Disqus' Privacy Policy Login

Recommend 2 Tweet Share Sort by Best

Start the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS

Name

Be the first to comment.

Subscribe Add Disqus to your siteAdd DisqusAdd Do Not Sell My Data

Latest Posts

Top OAuth API Vulnerabilities

 Gary Archer September 1, 2021

(https://nordicapis.com/top-oauth-api-vulnerabilities/)

The Top 4 Best Practices When Implementing API Security

 Crispen Maung August 31, 2021

(https://nordicapis.com/the-top-4-best-practices-when-implementing-api-security/)

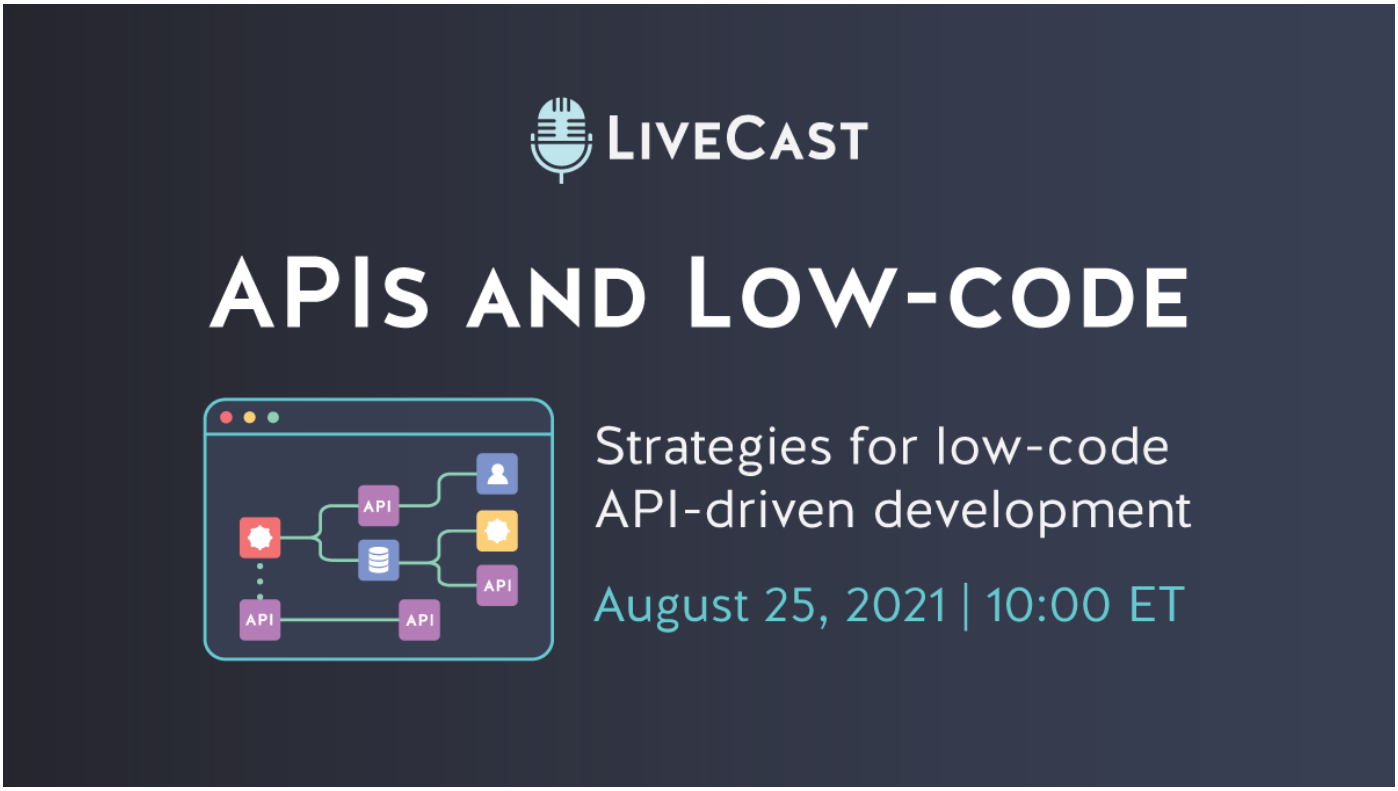
API Specifications Calm Chaos of Digital Transformation (Part 1)

 Frank Kilcommins August 26, 2021

(https://nordicapis.com/api-specifications-calm-chaos-of-digital-transformation-part-1/)

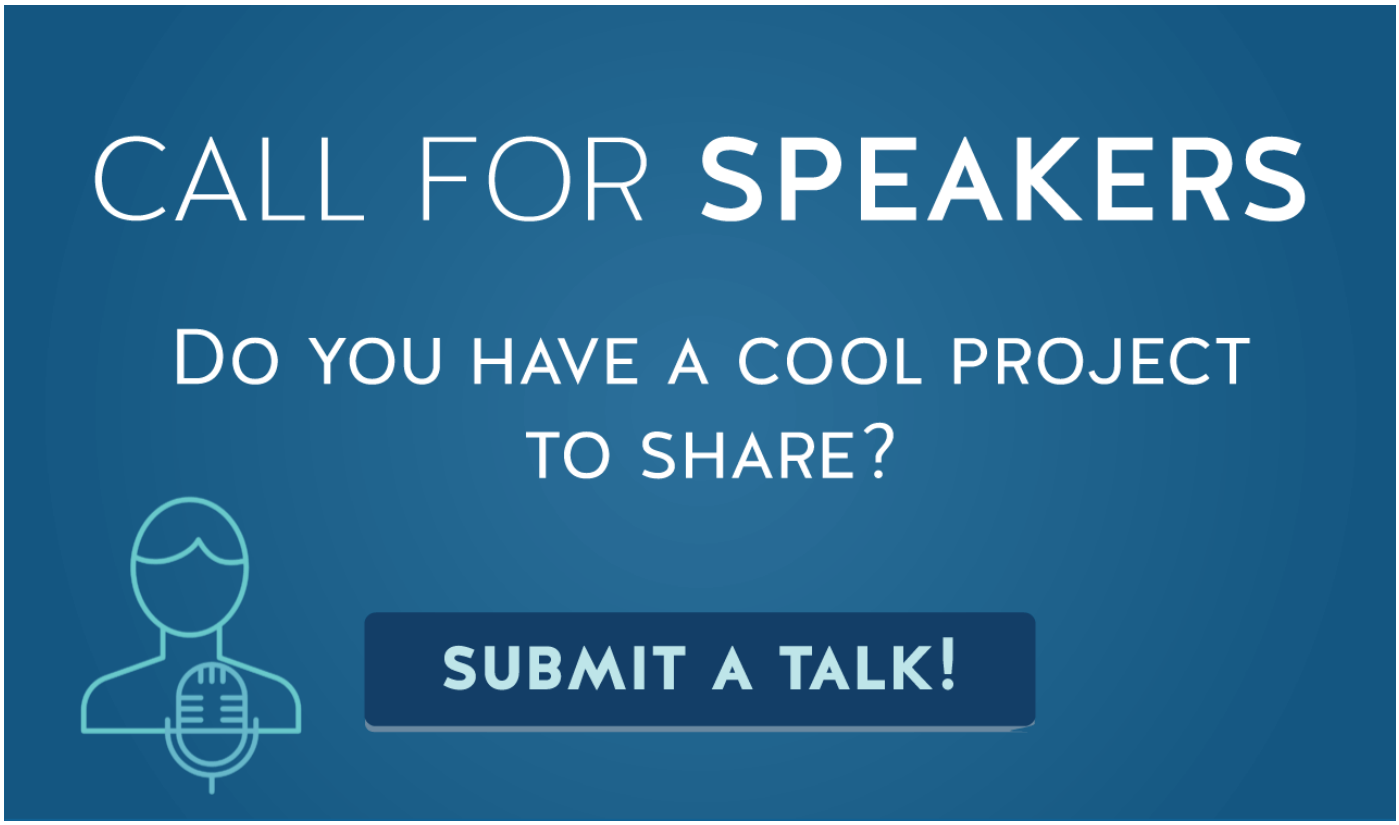
(https://nordicapis.com/best-public-api-of-2021/)

(https://nordicapis.com/events/evolving-hypermedia/)

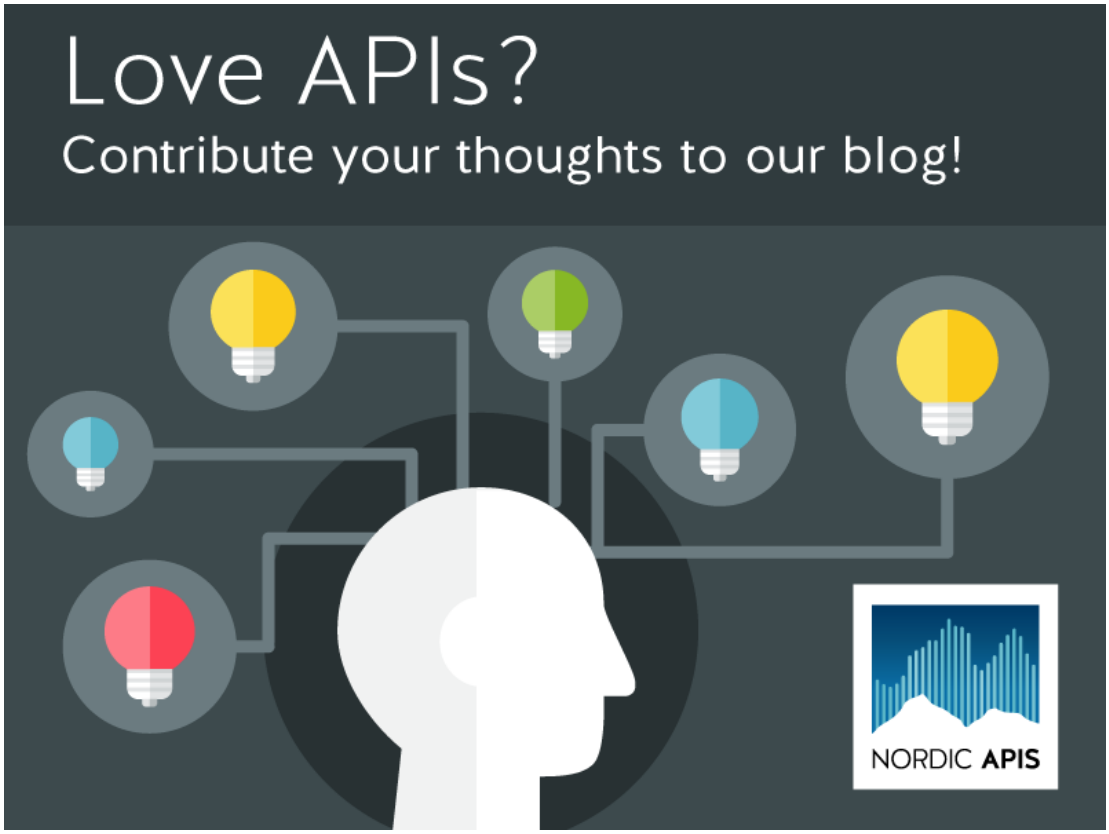


(https://nordicapis.com/events/apis-and-low-code/)

(https://nordicapis.com/ebooks/api-as-a-product/)



(https://nordicapis.com/call-speakers/)



(https://nordicapis.com/create-with-us/)

Smarter Tech Decisions Using APIs

High impact blog posts and eBooks on API business models, and tech advice

Connect with market leading platform creators at our events

Join a helpful community of API practitioners

API Insights Straight to Your Inbox!

Can't make it to the event? Signup to the Nordic APIs newsletter for quality content. High impact blog posts on API business models and tech advice.

tim@apple.com

Subscribe

Join Our Thriving Community

Become a part of the world's largest community of API practitioners and enthusiasts. Share your insights on the blog, speak at an event or exhibit at our conferences and create new business relationships with decision makers and top influencers responsible for API solutions.

Write

(https://nordicapis.com/create-with-us/)

Speak

(https://nordicapis.com/call-speakers/)

Sponsor

(https://nordicapis.com/about/contact-us/)

Volunteer (https://nordicapis.com/student-volunteer/)

Social

.com/nordicapis) (https://www.linkedin.com/company/nordic-apis) (https://www.facebook.com/NordicAPIs) (https://www.youtube.com/user/nordicapis)

© 2013-2021 Nordic APIs AB | Supported by CURITY (https://curity.io) | Website policies (/policies/)