(icapis.com)

NORDIC APIS

MENU

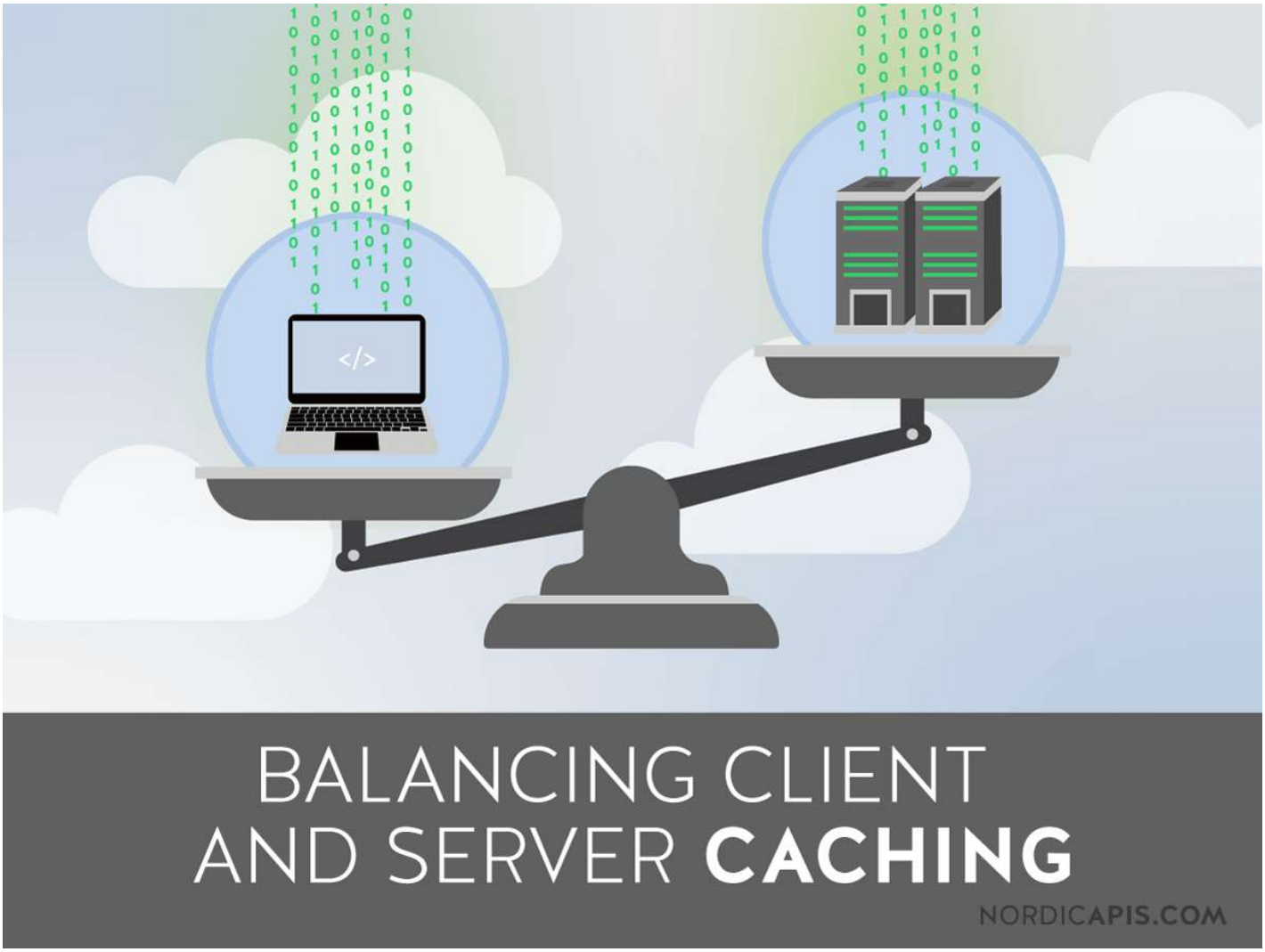
Balancing Client And Server Caching in Web Application Development (https://nordicapis.com/balancing-client-and-server-caching-in-web-application-development/)

Kristopher Sandoval(https://nordicapis.com/author/sandovaleffect/) December 20, 2018

(https://www.facebook.com/sharer.php?u=https%3A%2F%2Fnordicapis.com%2Fbalancing-client-and-server-caching-in-web-application-development%2F)

(https://twitter.com/intent/tweet?url=https%3A%2F%2Fnordicapis.com%2Fbalancing-client-and-server-caching-in-web-application-development%2F&text=Balancing+Client+And+Server+Caching+in+Web+Application+Development)

(https://ww client-and-serv development%



Above all else, API communication is really a **negotiation** between the **client** and the **server**. Once we clear all the dust, this basic communication underlies every protocol, architecture, and approach. These negotiations are often complex, and deciding who has the responsibility for what aspect of this negotiated communication is an age-old conversation.

One aspect of this content negotiation is **caching**. Where do we cache data, and why? Who has the responsibility for collecting this cache, and what are the implications of each decision regarding this location?

The variations in how things are cached, where they are cached, and by what mechanism they are cached are **paramount to an API**. Today, we're going to talk about this balance in the context of **web application development**. We'll define what caching actually is, and dive a bit into the general approaches one might take.

What is Caching?

In the online space, many techniques are collectively referred to as "caching" due to their ability to mirror this functionality. In the most simple terms, **caching** is a general computer concept that provides **efficiency** through **data availability**. The mechanism by which this is accomplished is through the **storage of commonly accessed data** in several places, and then serving that data to requesters from the common data store. This is opposed to generating new content each time it is requested.

By providing **commonly requested data** to users who often request that data when calling the same functions, you can avoid a lot of extra data generation, optimize (https://nordicapis.com/optimize-developer-experience-api/) the request workflow, reduce time to delivery (https://nordicapis.com/how-to-handle-the-continuous-delivery-of-microservices/), and remove congestion on your API input/output path by saving vital processing and networking resources.

By storing this data, **efficiency** is improved. When a requester makes a request for this data, the API prioritizes stored versions, wherever that cache might be, over data generation, allowing for your delivery while freeing up otherwise occupied sources. This can be used for **computed data**, but it's also extremely helpful for certain types of **static information** – for instance, if the requester is asking for the location of an unmoving file, a directory, or even a version number, this information should be cached ahead of time and used for a quick, efficient response.

Data Cost

Before we dive into the various methodologies behind caching, we should briefly discuss the idea of **data cost**. Data cost is simply the idea that every operation, every function, every aspect of what we do with an API results in some sort of cost. This cost can arise from a variety of factors and systems, both originating from the server and the client, but ultimately, every data bit has a cost associated with its generation, transfer, and storage.

The question is then: **who should own that cost?** It is easy for developers to assume the user is ok with this cost since they have requested the data. Unfortunately, this cost isn't always controllable, and in many cases, ***you cannot assume the client is capable of handling that cost***. The user by definition requires data that it does not have, and the server, by definition, needs to send this data to the user as a business transaction. There is a desire to offload as much to the client as the server can, but clients can't always hold this information.

Of course, in many situations, the client does want control over content cost. Therefore, what it boils down to is that **there's no way to perfectly balance this cost**. Caching then helps reduce the cost at its most basic level by reducing the needless calls that generate excess cost and isolating that cost to only where the demand is valid and necessary.

Client Caching

Client caches help limit the data cost incurred by the user by keeping commonly referenced data **locally**. The client often requests data that may not be large but is indeed continuously needed.

For instance, if an **API** feeds into a web GUI, instead of requesting those images that make up the logo and other branding, that content can be stored locally. If the API feeds a directory to the user, the user can cache this directory locally instead of requesting it from the server, which cuts out the directory lookup stage from the client. All of this helps reduce the data cost in terms of network utilization and processor demand and improves the efficacy of the overall system.

From the API point of view, the client makes a request to the API. The client first looks locally for the relevant data. If this data isn't found, the request is then sent to the external resource, and the content is generated for the requesting client.

In many systems, this content is then stored for a time with a given expiry date from the last time it was requested. This allows the cache to be **dynamic**, providing the content that is commonly requested to the user while preventing size bloat and clearing unneeded data once it is no longer useful.

The benefit here is that the network for the client does not face heavy traffic demands without cause, as a great many content requests can be kept local. Additionally, this frees up time on the server side, which no longer has to field repeat queries that have already been answered.

Server Caching

Server caching helps limit the cost incurred by the server and its underlying systems. Many requests made by clients can either be responded to using the same data, or responded to using parts of the same requests made by others.

For instance, **database queries** are often used for specific purposes – a client synchronizing a local directory listing of the server resource map might request a full survey of the resources every 2 hours. In this case, the directory can be cached on server, and then each request for this synchronization can then be patched through to the cached copy, which is checked against the reality on the server. In this way, the database is saved from making a ton of calls it would otherwise be responsible for, saving data cost and improving efficiency.

To the API, this request looks like this:

1. The client makes a request to the API.
2. This request is received by the server.
3. The server checks for a local copy of the file requested. While this check has a cost, the cost is still extremely low compared to the actual cost of computing a massive database check or generating content.
4. If the local resource exists, the server responds with its resource URI.
5. The client grabs the cached content.
6. If the local resource does not exist, the request is processed normally.

While this doesn't really save much cost for the client, **the savings to the server can be quite significant**, especially when databases or high-volume resources are concerned. Caching commonly requested content can result in some massive data cost savings and improved network congestion, as these requests can often be offloaded onto other servers that aren't handling live queries. This means those servers can be lower power, less-resource intensive, and still provide this data with minimal overhead.

Hybrid Caching

Caching isn't just a choice between one or the other, either – you can combine client and server caching to have a more complete solution if your specific system allows it. In this approach, you leverage **both types of caching** to free up the data cost burden on both sides of the equation by asking whoever is responding or requesting to first make a local query.

From the API perspective, the flow would follow as such:

1. A client makes a request
2. It first checks for a local copy. If the copy doesn't exist, it contacts the server with a request for this content.
3. On the server side, the server will then check for its own local copy,
4. If a copy exists, it will serve it. Or, it will generate a new one if the copy does not exist.

In this method, there's now **two separate cache processes** that could potentially reduce the data cost of either side of the equation. This also allows for client or user-specific content that may not be applicable to other users to be stored on the local client cache while the server cache stores universally required data.

This caching power can be boosted through the use of other **external caching services** as well – **content delivery networks**, for instance, can store this cached content off the server, freeing up local cost for the server, and reducing the server load for content delivery. In these cases, the content is sent by **multiple servers**, meaning faster data delivery and greater redundancy, as well as a freeing of core resources for the actual main systems servers.

More on API Optimization: Optimizing The API Response Package (https://nordicapis.com/optimizing-the-api-response-package/)

Case Study – Evolv App

Let's see how this all might work in a hypothetical case study application. Let's assume we are developing a hybrid caching solution for a **Human Resources API** called **Evolv**. Evolv has a web GUI that ties into the backend (https://nordicapis.com/the-benefits-of-a-serverless-api-backend/)API, and allows users to utilize devices as part of a Bring Your Own Device policy. Evolv synchronizes employee contact data between different departments, updating local devices – it's essentially a "secure contact database" for companies, allowing you to have an authenticated user database locked behind a multifactor security system

From a technological perspective, we have several processes here – a local application that synchronizes data between client and server, a server that collects updates from the client and updates the local database, and a process that checks for discrepancies between the cached content and the current database.

Local Caching

Since the application has a process that allows users to change and manage their local contact database, a locally cached version of the contact database is kept. This allows for changes to be reverted, and enables restoration of the local copy to previous versions. Additionally, by having a locally cached version of the content separate from the local database, the content can be updated or corrected to multiple different entry versions without rechecking against the server (for instance, if you've entered the wrong new number for a person that still uses the existing number, this method would allow you to revert those changes).

Additionally, having a locally cached version allows for a second step in which the device can check for discrepancies between the most recent version of the cache and the server state. By doing this, new copies can be backed up, added, or revised seamlessly, keeping contact lists up to date without removing vital personal information as chosen by the client.

Server Caching

Meanwhile, the server still needs to maintain its own data resource to enable all of this. By synchronizing its current database with a local cache, the data source can ensure the ability to revise while also feeding new data seamlessly to applications as they request it without pinging the main server.

By synchronizing the current database nightly after the synchronization with the user changes, the server can provide a "fallback" for when new clients are created or old backups are corrupted. This enables a single database query to be performed on the server side each day barring emergency queries, thereby saving significant processing power and network transfer.

Caveats

There are some significant **caveats** to consider when caching content.

Primary of these is the fact that caching has some serious implications when it comes to **privacy** and **security**. Cached content may be slow to change, especially if you leverage a content delivery network, and in those cases, fixes to privacy issues may take some time to propagate. In many cases, the damage might already be done. Additionally, some functions might accidentally leak, in which case, these cached copies might add to the problem and make these leaky functions even worse.

Also keep in mind that a caching process is essentially a **stored call**. Accordingly, misconfiguration of these calls can propagate into the cached versions if you are not careful, which can result in everything from incorrect cached content to loss of data.

Ultimately, **caching is a balancing game, and a poor balance can incur a huge cost in terms of both performance and monetary value.** Choosing what to cache is almost as important as choosing where to cache it, so consider this a significant aspect of your caching strategy.

Conclusion

The argument about caching locations is really a selfish one. Servers will always want more control and less cost, while clients clearly want faster communication and more security. Ultimately, the correct caching relationship comes down to the age-old non-answer of "what works best for your situation."

The truth is that there are endless permutations of caching solutions and relationships, so choosing your optimum option is going to come down to the situation and layout of your various systems. Thin clients will obviously benefit much more from server caching than a traditional computation-based system calls which might utilize local storage on so-called "traditional" towers. With all of this, there are of course a number of caveats that cannot be predicted – thus, you need to look at your current system layout and find what works best in your particular instance.

What do you think? Do you think that there is an optimal balance for caching that covers most use cases? Let us know below!

api (https://nordicapis.com/tag/api/), cache (https://nordicapis.com/tag/cache/), cache data (https://nordicapis.com/tag/cache-data/), caching (https://nordicapis.com/tag/caching/), cdn (https://nordicapis.com/tag/cdn/), client (https://nordicapis.com/tag/client/), Client caches (https://nordicapis.com/tag/client-caches/), client-side (https://nordicapis.com/tag/client-side/), clients (https://nordicapis.com/tag/clients/), content delivery network (https://nordicapis.com/tag/content-delivery-network/), content delivery networks (https://nordicapis.com/tag/content-delivery-networks/), data (https://nordicapis.com/tag/data/), data cost (https://nordicapis.com/tag/data-cost/), data store (https://nordicapis.com/tag/data-store/), database (https://nordicapis.com/tag/database/), database queries (https://nordicapis.com/tag/database-queries/), development (https://nordicapis.com/tag/development/), GUI (https://nordicapis.com/tag/gui/), server (https://nordicapis.com/tag/server/), server cache (https://nordicapis.com/tag/server-cache/), server caches (https://nordicapis.com/tag/server-caches/), Server caching (https://nordicapis.com/tag/server-caching/), server-side (https://nordicapis.com/tag/server-side/), servers (https://nordicapis.com/tag/servers/), web application development (https://nordicapis.com/tag/web-application-development/), web apps (https://nordicapis.com/tag/web-apps/), web development (https://nordicapis.com/tag/web-development/)

1 Comment (https://nordicapis.com/balancing-client-and-server-caching-in-web-application-development/#disqus_thread)

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/)

The Business Caveats of... (https://nordicapis.com/the-business-caveats-of-microservices/)

Key Lessons From 100 API... (https://nordicapis.com/key-lessons-from-100-api-projects/)

Latest Posts

7 Open-Source API Security Tools

Vyom Srivastava April 15, 2021
(https://nordicapis.com/7-open-source-api-security-tools/)

How OAuth Enables a Zero-Trust Architecture

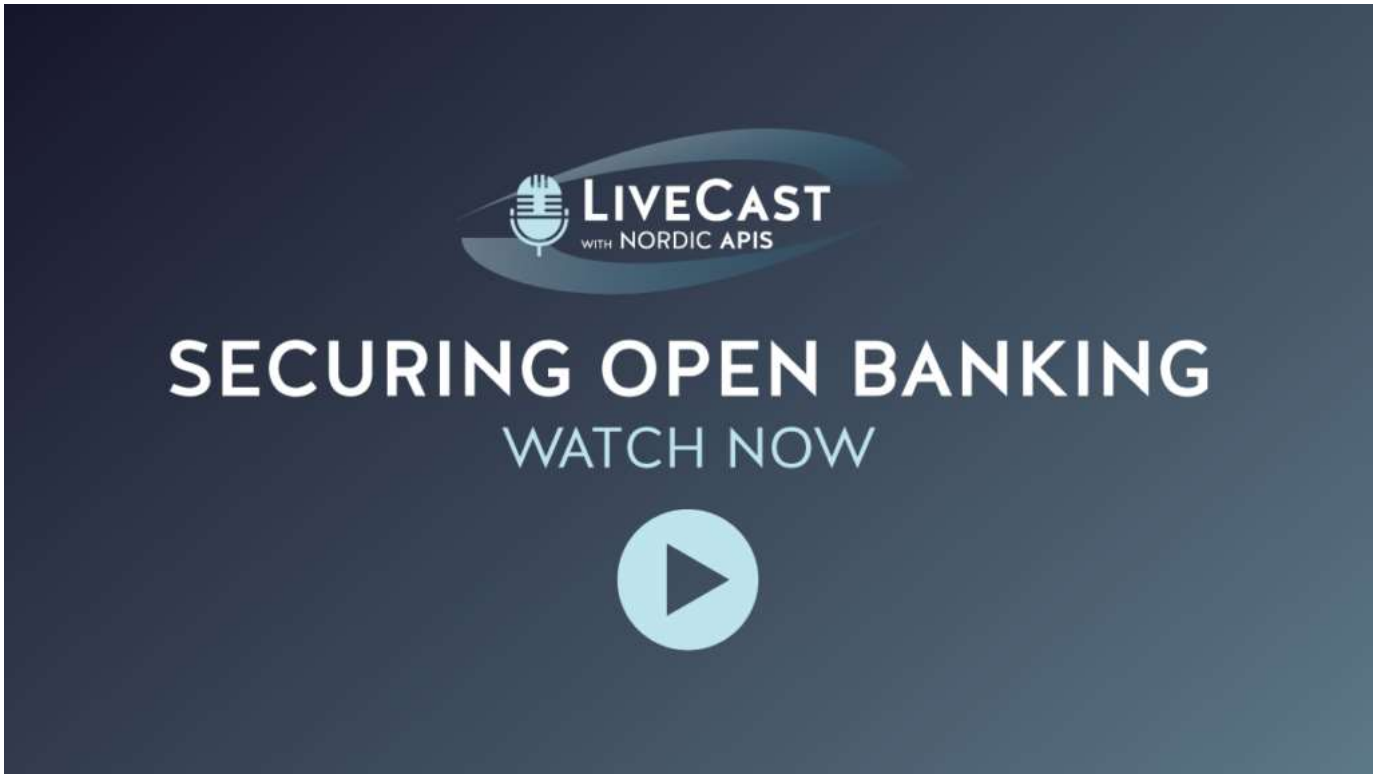
Kristopher Sandoval April 14, 2021
(https://nordicapis.com/how-oauth-enables-a-zero-trust-architecture/)

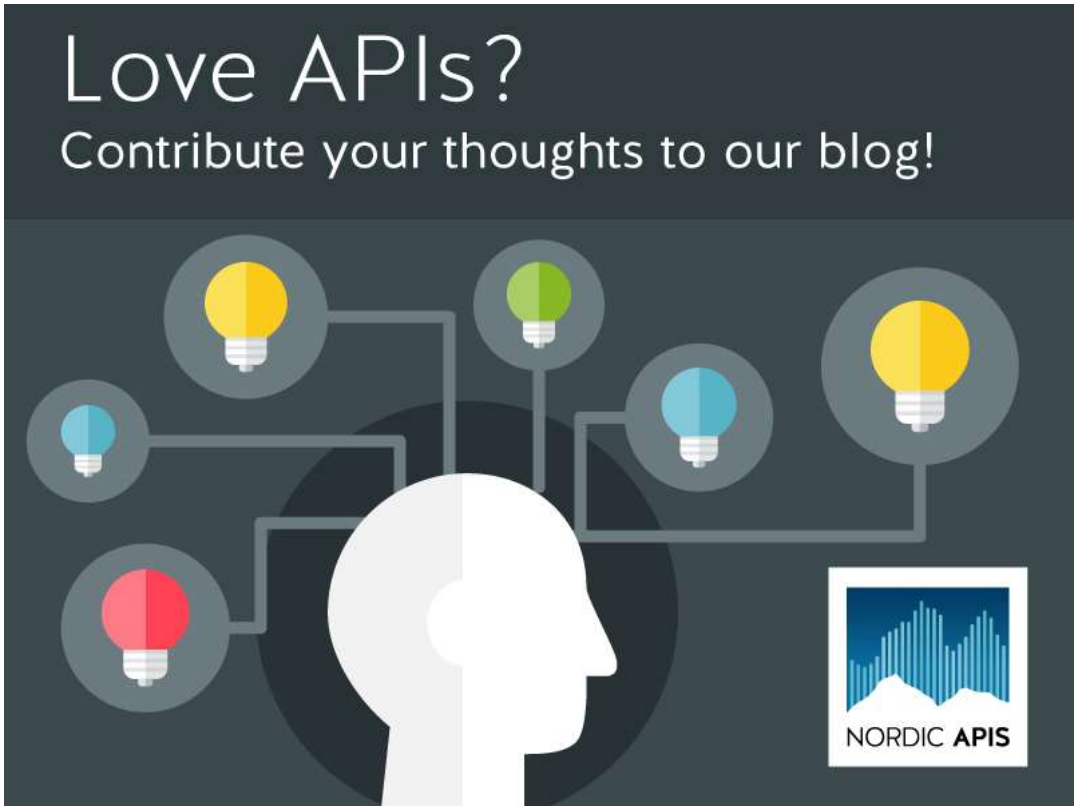
Creating the Frontend of a Headless CMS in 10 Steps

Vyom Srivastava April 13, 2021
(https://nordicapis.com/creating-the-frontend-of-a-headless-cms-in-10-steps/)

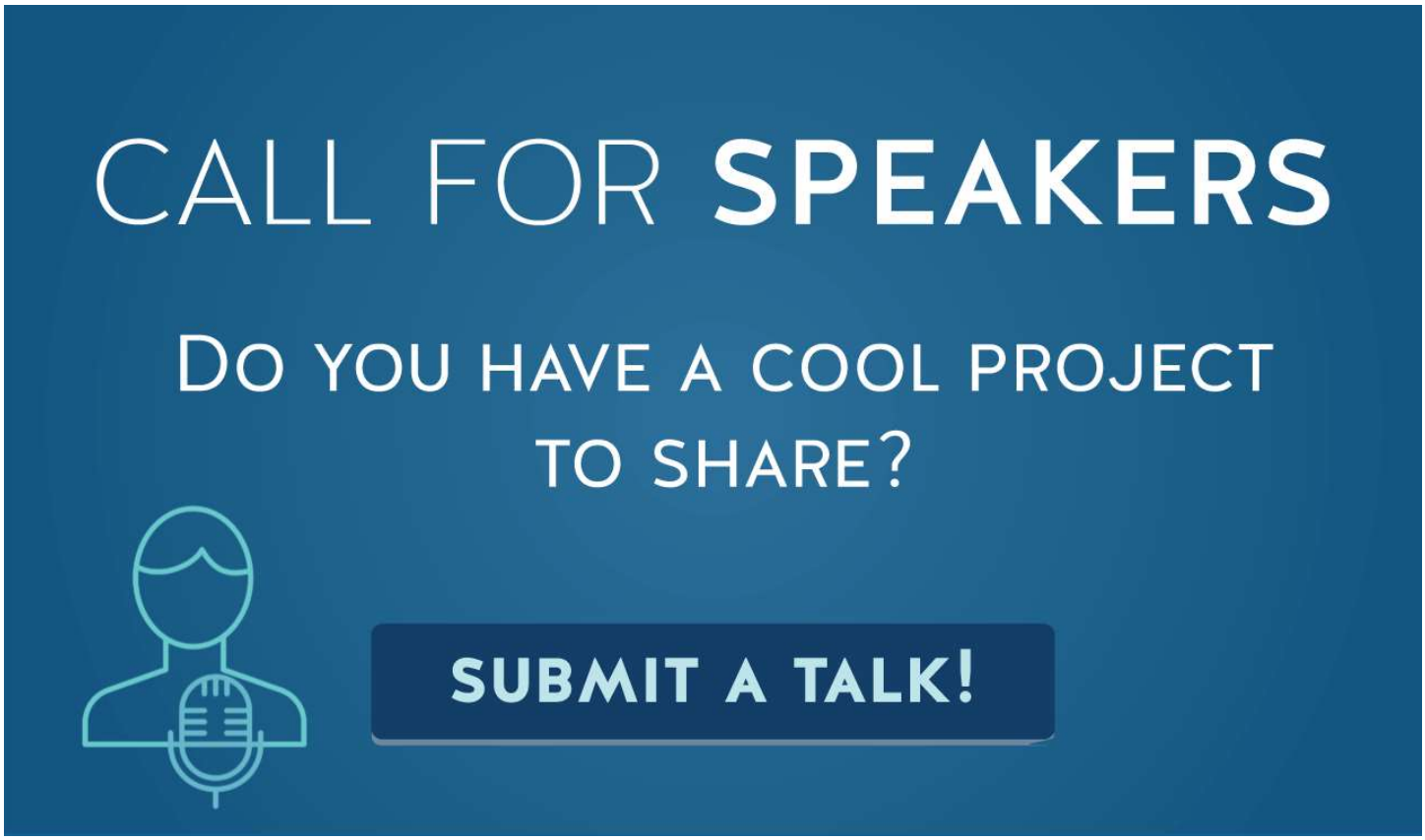
(https://nordicapis.com/events/livecast-api-as-a-product/)

(https://youtu.be/2Ud0JwC3JNI)





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



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

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/)

Events

Best Public API of 2020 (https://nordicapis.com/best-public-api-of-2020/)

Platform Summit 2020 (https://nordicapis.com/events/platform-summit-2020/)

Austin API Summit 2020 (https://nordicapis.com/events/austin-api-summit-2020/)

Blog

Blog (/blog)

Business Models (https://nordicapis.com/category/business-models/)

Marketing (https://nordicapis.com/category/marketing/)

Platforms (https://nordicapis.com/category/platforms/)

Security (https://nordicapis.com/category/security/)

Strategy (https://nordicapis.com/category/strategy/)

Design (https://nordicapis.com/category/design/)

Resources

eBooks (/api-ebooks/)

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

Call for Speakers (https://nordicapis.com/call-speakers/)

Code of Conduct (https://nordicapis.com/code-of-conduct/)

About

About (https://nordicapis.com/about/)

Press (https://nordicapis.com/about/press/)

Privacy Policy (https://nordicapis.com/nordic-apis-privacy-policy/)

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

Social

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