

# Blog

## 7 Unique Benefits of Using GraphQL in Microservices (<https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/>)

POSTED BY KRISTOPHER SANDOVAL ([HTTPS://NORDICAPIS.COM/AUTHOR/SANDOVALEFFECT/](https://nordicapis.com/author/sandovaleffect/)) | JANUARY 2, 2018

DESIGN ([HTTPS://NORDICAPIS.COM/API-INSIGHTS/DESIGN/](https://nordicapis.com/api-insights/design/))

3

**f** Facebook (<https://www.facebook.com/sharer/sharer.php?u=https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/&t=7+Unique+Benefits+of+Using+GraphQL+in+Microservices>)

**2** **t** Twitter **G+** Google+ (<https://plus.google.com/share?url=https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/>)

95

**in** LinkedIn (<https://www.linkedin.com/shareArticle?mini=true&ro=true&trk=EasySocialShareButtons&title=7+Unique+Benefits+of+Using+GraphQL+in+Microservices&url=https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/>)

**Reddit** (<http://reddit.com/submit?url=https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/&title=7+Unique+Benefits+of+Using+GraphQL+in+Microservices>)

**Y** HackerNews (<https://news.ycombinator.com/submitlink?u=https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/&t=7+Unique+Benefits+of+Using+GraphQL+in+Microservices>)

Total: 100

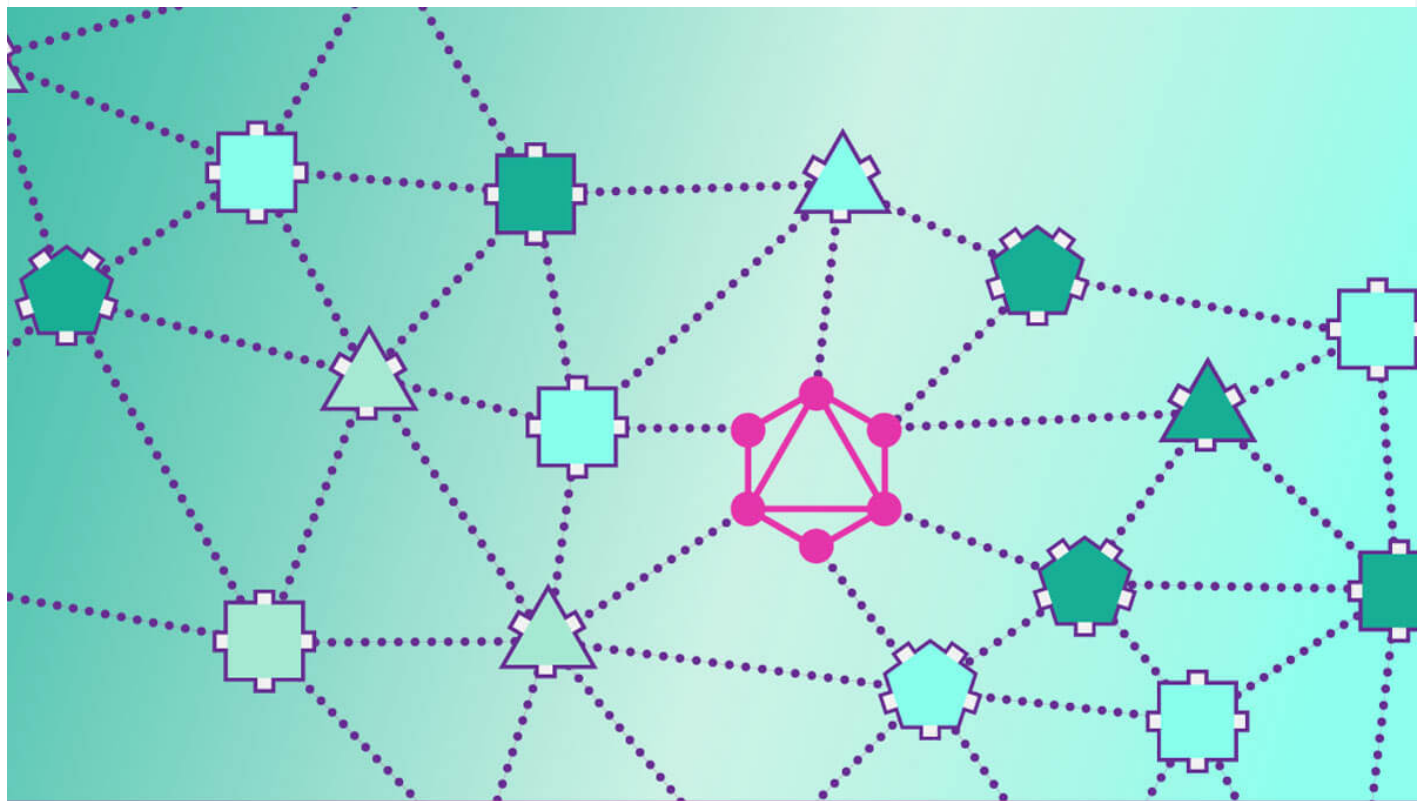
*This post is included in our eBook on GraphQL. Download a free copy here! (<http://nordicapis.com/ebook-released-graphql-bust/>)*

(<http://19yw4b240vb03ws8qm25h366-wpengine.netdna-ssl.com/wp-content/uploads/7-Unique-Benefits-of-Using-GraphQL-in-Microservices.jpg>)

We've talked about GraphQL at length previously, and for very good reason – GraphQL is, in many ways, one of the more powerful tools an API provider has in terms of providing singular endpoints to the consumer and controlling data flow. While this value has been proven time over time, however, it seems that some of the more salient and special benefits of adopting GraphQL are often lost in the conversation.

Today, we're going to talk about these unique benefits, and what they actually mean to a production API. Many are familiar with **microservices**, so in this piece we'll discover positive impacts GraphQL brings a microservices arrangement, such as data owner separation, granular data control, parallel execution, service caching, and more.

This piece was, in part, inspired by Tomer Elmalem (<http://nordicapis.com/speakers/tomer-elmalem/>), who delivered the awesome *GraphQL APIs: REST in Peace* talk at the 2017 Nordic APIs Platform Summit.



# 7 Unique Benefits of Using GraphQL in Microservices

NORDICAPIS.COM

## Clearly Separated Data Owners

One of the main benefits of having everything behind a single endpoint is that data can be routed **more effectively** than if each request had its own service. While this is the often touted value of GraphQL (<http://nordicapis.com/5-potential-benefits-integrating-graphql/>), a reduction in complexity and service creep, the resultant data structure also allows data ownership to be extremely **well defined**, and clearly delineated.

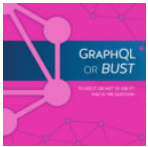
This is in large part because the request itself is pointed towards a **single endpoint**, and that request must package the expected data format (<http://nordicapis.com/what-data-formats-should-my-api-support/>) and the resource owner designation, whether it be for a specific function or a specific data type, is thus intrinsically tied to that request.

This is different from a typical REST API (<http://nordicapis.com/designing-a-true-rest-state-machine/>), in which the request to each microservice might be requesting data that the microservice endpoint doesn't support or cannot deliver, which would then be passed on to another endpoint, and so on. This ends up creating a **web of requests** and passed communication, and to the average viewer, knowing who actually owns the requested resource becomes hard to decipher.

## Data Load Control Granularity

Another benefit of adopting GraphQL is the fact that you can fundamentally assert **greater control** over the **data loading** process. Because the process for data loaders goes into its own endpoint, you can either honor the request partially, fully, or with caveats, and thereby control in an extremely granular way how data is transferred.

In many ways, this granularity is what REST has tried to achieve with some degree of success when implementing specific schemas and request forms. That being said, GraphQL integrates this as a specific structural element of how it works, and as such, does so more effectively than most other solutions.



(<http://19yw4b240vb03ws8qm25h366-wpengine.netdna-ssl.com/wp-content/uploads/Graphql-or-bust.jpg>) Download our free eBook *GraphQL or Bust*. Learn about GraphQL and gauge if it's right for your API.

## Parallel Execution

Because a single GraphQL request can be composed of many requested resources, and because GraphQL can choose dynamically when, how, and to what extent a response is issued to such a request bundle, GraphQL can leverage a sort of **parallel execution** for resource requests. What this functionally results in is the ability for GraphQL requests to partially fail, but for the response to deliver **more useful data to more requests** in a single issuance.

This ultimately has the benefit of allowing a single request, even when partially failed, to serve the place of what would traditionally be multiple requests to multiple services over multiple servers. This also allows a single request to use a relatively more restrained amount of processor time and power to deliver the same amount of information that would otherwise be required of those multiple requests, thereby delivering greater power with less requirements.

“Instead of having six sequential calls, what [a parallel dataloader] will do is give you all six IDs in one go, so you can make one request instead of six. This will make your downstream databases and APIs way happier – no one will be on fire, no one getting paid at 2 am in the morning.” -Tomer Elmalem

## Request Budgeting

In GraphQL, requests can be given a “maximum execution time”, and this value can then be used to **budget requests**. Each request is given a value, and from that, the server budget is calculated and calls are prioritized. As an example, let's assume our server has a total budget of **2 seconds**, and look at a sample batch of requests.

We receive **four requests** – one is a single second, two of them half a second, and the final request a full two seconds. When budgeting our requests, the GraphQL server can accept the first three requests, and either delay or refuse the last request, as it would exceed the allotted time that the server has open for requests of its nature.

What this in effect does is allow the server to prioritize requests and grant them where appropriate, which ends up **reducing timed out requests** in the long run. Additionally, this system can return information to the requester – such as would be the case with a 6 second request, for instance – that can then inform the user to break their requests into smaller pieces or wait for a low-budget time to make the request.

“[As an example] we set the maximum budget to one second. Some sleep function takes about half a second, and we've got about half a second in budget remaining. [...] Our budget is one second, but we have some function that takes a second and a half, so we're actually negative 500 milliseconds in terms of budget – so everything downstream fails. If this happens on the first service call, then we can skip executing the next 6 calls [which avoids] a lot of wasted processing power when requests are timing out.”

## Powerful Query Planning

Combining two of these major benefits – parallel requests and budgeting – allows us to more **effectively plan out our query schedules**. By being able to send some queries to an endpoint, and have others execute in parallel at a later time per their weight and processor demand, you can effectively plan out those queries over a relatively broad set of criteria and per the time allotted.

While this seems simple, the ability to plan out queries over time and address per priority is one of the elements of GraphQL that is so incredibly powerful.

## Service Caching

GraphQL utilizes **Object Identifiers** for one of the biggest savers in terms of server processing demands – **caching**. By being able to cache resource for services which request them, GraphQL can essentially build a cache of often-requested data and save processor time and effort.

According to Elmalen, this is rather easy to implement, in fact – GraphQL documentation (<http://graphql.org/learn/caching/>) suggests utilizing a system of **globally unique IDs** in order to note the resources being cached, thereby providing them with minimal processing demand.

“Since you’re handling a lot of network requests, you don’t want to have to keep making those requests over and over and over again if you’re seeing the same data frequently.”

## Easy Failure Handling and Retries

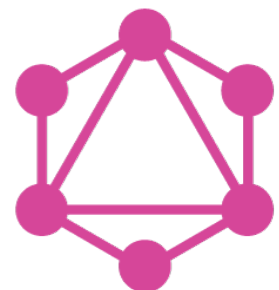
GraphQL is unique, in that, in a normal connection and request cycle, there is no real “fail” or “succeed” – everything is either succeed or partially fail. GraphQL requests can partially succeed, returning only elements of the data requested or specific datasets as allowed, and this control can be very granular.

As a result of this, a resolver to a request can be more than “all or nothing,” partially resolving as a single resolver delivers content, and the rest of the request is dumped. This means that **GraphQL handles failure rather gracefully**, notifying the requester of the failure but returning useful, actionable data (<http://nordicapis.com/best-practices-api-error-handling/>) alongside the noted failure and what caused said failure.

As part of this, GraphQL requests can be **parsed** to find the null fields returning in a failed request, and note what the error actually was. If the error was a simple misconfiguration or poorly formed call, it can be made in a better format upon instruction from the GraphQL layer itself – if the data is prohibited or absent, the requester can also ascertain this. This means that **automatic retries** are possible, as are granular levels of call termination and auto-failure handling.

“In GraphQL we have a lot of flexibility with how the resolvers work. Query execution doesn’t have to be all or nothing like it might be in a REST API, where if one endpoint call throws an exception everything might blow up. Resolvers execute independently of everything else, so one resolver can fail but the entire query can still resolve.”

## Case Study of Microservices In Action: How GraphQL Benefits Yelp



(<http://19yw4b240vb03ws8qm25h366-wpengine.netdna-ssl.com/wp-content/uploads/yelp-plus-graphql.png>)

To see how **GraphQL** is useful in a microservices approach (<http://nordicapis.com/api-ebooks/strategies-microservices-architecture/>), we can look at Tomer's argument in favor of integration at **Yelp**. Yelp is foundationally a business that ties to additional businesses, and as such, utilizes a **public API** rather heavily. Their public API is supposed to give clients data connections by which the resources can be collected, reviewed, and collated.

In their original Yelp API, this **external demand** for data caused a significant problem. The API was too large, had too many endpoints, and as more data was requested from outside partners, the API was expanded either with more bloated endpoints or with additional endpoints to feed out the data. This resulted in **an extremely large solution** that was hard to maintain, hard to iterate upon, and generally less agile (<http://nordicapis.com/what-makes-an-agile-api/>) than desired.

As a solution, Yelp began to implement a new design paradigm in GraphQL. Their chief reasoning was the fact that, for their outside businesses, those partners wanted to make a single request for the data they needed, and nothing more. **GraphQL fit this requirement perfectly**, as a single endpoint could now serve a multitude of resources in a defined and predictable way. More important, only a single request would have to be made, as that request could be formed in the way that the data requester needed the data and it could be served based upon the agreed data served by the provider.

This process and choice really encapsulates the purpose of GraphQL in a microservices architecture (<http://nordicapis.com/microservices-architecture-the-good-the-bad-and-what-you-could-be-doing-better/>). Sure, it's true that Yelp could have simply added more endpoints, created more microservices, expanded the resource handling (<http://nordicapis.com/optimizing-the-api-response-package/>), and slimmed the bloat process of their API. All this would have, however, increased complexity, bloat, and the cost to maintain the API.

By adopting GraphQL, however, their microservice-oriented design functions more agile, more responsively, and, perhaps most importantly, more adherent to the design requirement at hand – **a single request delivering the data as requested**.

## Final Thoughts

It should be noted that GraphQL is extremely powerful, and that the benefits (<http://nordicapis.com/5-potential-benefits-integrating-graphql/>) at hand are especially powerful for most microservice structures – though not all. In some cases, especially in cases in which the data does not change, new endpoints are not added, and additional functions are not required, GraphQL may add additional unwarranted complexity.

GraphQL has often been sold as a perfect solution for every problem, but the reality is that it meets one requirement better than most others, and if that's not part of your requirements, it may not be the best solution for your implementation.

For situations like that faced by Yelp, however, GraphQL fits perfectly and solves the major issues at hand. For this reason alone, should developers find themselves in a similar microservice architecture and requiring a greater flexibility in data delivery and structuring, GraphQL should absolutely be a top consideration.

3

**f** Facebook (<https://www.facebook.com/sharer/sharer.php?u=https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/&t=7+Unique+Benefits+of+Using+GraphQL+in+Microservices>)

**2** **T** Twitter **G+** Google+ (<https://plus.google.com/share?url=https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/>)

95

**in** LinkedIn (<https://www.linkedin.com/shareArticle?mini=true&ro=true&trk=EasySocialShareButtons&title=7+Unique+Benefits+of+Using+GraphQL+in+Microservices&url=https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/>)

**Reddit** (<http://reddit.com/submit?url=https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/&title=7+Unique+Benefits+of+Using+GraphQL+in+Microservices>)

**Y** HackerNews (<https://news.ycombinator.com/submitlink?u=https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/&t=7+Unique+Benefits+of+Using+GraphQL+in+Microservices>)

Total: 100

 [api](https://nordicapis.com/tag/api/) (<https://nordicapis.com/tag/api/>), [API design](https://nordicapis.com/tag/api-design/) (<https://nordicapis.com/tag/api-design/>), [API optimization](https://nordicapis.com/tag/api-optimization/) (<https://nordicapis.com/tag/api-optimization/>), [API performance](https://nordicapis.com/tag/api-performance/) (<https://nordicapis.com/tag/api-performance/>), [APIs](https://nordicapis.com/tag/apis/) (<https://nordicapis.com/tag/apis/>), [benefits](https://nordicapis.com/tag/benefits/) (<https://nordicapis.com/tag/benefits/>), [caching](https://nordicapis.com/tag/caching/) (<https://nordicapis.com/tag/caching/>), [data owners](https://nordicapis.com/tag/data-owners/) (<https://nordicapis.com/tag/data-owners/>), [ebook](https://nordicapis.com/tag/ebook/) (<https://nordicapis.com/tag/ebook/>), [GraphQL](https://nordicapis.com/tag/graphql/) (<https://nordicapis.com/tag/graphql/>), [GraphQL layer](https://nordicapis.com/tag/graphql-layer/) (<https://nordicapis.com/tag/graphql-layer/>), [load control](https://nordicapis.com/tag/load-control/) (<https://nordicapis.com/tag/load-control/>), [microservices](https://nordicapis.com/tag/microservices/) (<https://nordicapis.com/tag/microservices/>), [microservices architecture](https://nordicapis.com/tag/microservices-architecture/) (<https://nordicapis.com/tag/microservices-architecture/>), [Object Identifiers](https://nordicapis.com/tag/object-identifiers/) (<https://nordicapis.com/tag/object-identifiers/>), [parallel execution](https://nordicapis.com/tag/parallel-execution/) (<https://nordicapis.com/tag/parallel-execution/>), [Platform Summit](https://nordicapis.com/tag/platform-summit/) (<https://nordicapis.com/tag/platform-summit/>), [query language](https://nordicapis.com/tag/query-language/) (<https://nordicapis.com/tag/query-language/>), [query planning](https://nordicapis.com/tag/query-planning/) (<https://nordicapis.com/tag/query-planning/>)

**0** Comments ([https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/#disqus\\_thread](https://nordicapis.com/7-unique-benefits-of-using-graphql-in-microservices/#disqus_thread))

planning/), request (<https://nordicapis.com/tag/request/>), request budgeting (<https://nordicapis.com/tag/request-budgeting/>), REST API (<https://nordicapis.com/tag/rest-api/>), Tomer Elmalem (<https://nordicapis.com/tag/tomer-elmalem/>), unique (<https://nordicapis.com/tag/unique/>), Yelp! (<https://nordicapis.com/tag/yelp/>)



## About Kristopher Sandoval

Kristopher is a web developer and author who writes on security and business. He has been writing articles for Nordic APIs since 2015.

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

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



eBook Released: GraphQL or... (<https://nordicapis.com/ebook-released-graphql-bust/>)

Scalable APIs are Built From...



(<https://nordicapis.com/scalable-apis-are-built-from-consistency/>)

0 Comments

Nordic APIs

 Login

 Recommend

 Share

Sort by Best



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 


Name

Be the first to comment.

ALSO ON NORDIC APIS


### Digitize Your Notes With Microsoft Computer Vision API

4 comments • a year ago

 AI Nelson — Hi everyone, AI here. If you have any extra comments or questions about the article let me know and I'll be glad to help. What other use cases can you think ...


### Introduction to API Versioning Best Practices

2 comments • 9 months ago

 SUDOisEvil — api.mydomain.tld -> returns a list of api version end points and links to their respective documentation. 100.api.mydomain.tld -> returns ...


### 20+ API Management Solutions

8 comments • 3 months ago

 Bill C. Doerrfeld — added, thanks for reading!

### High-Grade API Security For Banks

2 comments • 7 months ago

 Martin Flower — I wonder if "\$250 billion annually on cybersecurity" should actually be \$500 million annually ?

 Subscribe  Add Disqus to your siteAdd DisqusAdd  Disqus' Privacy PolicyPrivacy PolicyPrivacy Policy



(<https://nordicapis.com/events/the-2018-platform-summit/>)



(<https://nordicapis.com/best-public-api-of-2018/>)





(<https://nordicapis.com/events/livecast-the-role-of-identity-in-api-security/>)

## SMARTER TECH DECISIONS USING APIS

Subscribe to our mailing list

Subscribe

## POPULAR POSTS



(<https://nordicapis.com/7-frameworks-to-build-a-rest-api-in-go/>) 7 Frameworks To Build A REST API In Go (<https://nordicapis.com/7-frameworks-to-build-a-rest-api-in-go/>)

by Kristopher Sandoval (<https://nordicapis.com/author/sandovaleffect/>) | posted on July 4, 2017



(<https://nordicapis.com/is-rest-still-a-relevant-api-style/>) Is REST Still A Relevant API Style? (<https://nordicapis.com/is-rest-still-a-relevant-api-style/>)

by Bill Doerrfeld (<https://nordicapis.com/author/billdoerrfeld/>) | posted on August 8, 2018



(<https://nordicapis.com/defining-stateful-vs-stateless-web-services/>) Defining Stateful vs Stateless Web Services (<https://nordicapis.com/defining-stateful-vs-stateless-web-services/>)

by Kristopher Sandoval (<https://nordicapis.com/author/sandovaleffect/>) | posted on May 11, 2017



(<https://nordicapis.com/5-lightweight-php-frameworks-build-rest-apis/>) 5 Lightweight PHP Frameworks to Build REST APIs (<https://nordicapis.com/5-lightweight-php-frameworks-build-rest-apis/>)

by Kristopher Sandoval (<https://nordicapis.com/author/sandovaleffect/>) | posted on May 4, 2017



(<https://nordicapis.com/top-specification-formats-for-rest-apis/>) Top Specification Formats for REST APIs (<https://nordicapis.com/top-specification-formats-for-rest-apis/>)

by Kristopher Sandoval (<https://nordicapis.com/author/sandovaleffect/>) | posted on September 8, 2015

Search



## RECENT POSTS

Assisted Token Flow: The Answer to OAuth Integration in Single Page Applications (<https://nordicapis.com/assisted-token-flow-the-answer-to-oauth-integration-in-single-page-applications/>)

Benefits Of The DevSecOps Approach (<https://nordicapis.com/benefits-of-the-devsecops-approach/>)

Case Study: How Square Automates SDK Generation (<https://nordicapis.com/case-study-how-square-automates-sdk-generation/>)

Is REST Still a Relevant API Style? (<https://nordicapis.com/is-rest-still-a-relevant-api-style/>)

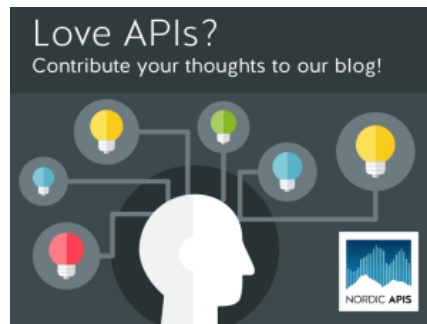
How APIs Make Marketing Data Analysis and Reporting Infinitely Easier (<https://nordicapis.com/how-apis-make-marketing-data-analysis-and-reporting-infinitely-easier/>)

## SUBSCRIBE TO OUR FEED



Nordic APIs RSS (<http://nordicapis.com/feed/>)

## CREATE WITH US



([https://docs.google.com/a/twobotechnologies.com/forms/d/12Ng9A\\_QKUjmAHDgv8Pxb4uLkECGJawV3vwAWJ4WxTs/viewform](https://docs.google.com/a/twobotechnologies.com/forms/d/12Ng9A_QKUjmAHDgv8Pxb4uLkECGJawV3vwAWJ4WxTs/viewform))



TWITTER (<HTTPS://TWITTER.COM/NORDICAPIS>)



FACEBOOK (<HTTPS://WWW.FACEBOOK.COM/NORDICAPIS>)



YOUTUBE (<HTTPS://WWW.YOUTUBE.COM/USER/NORDICAPIS>)



SLIDESHARE (<HTTP://WWW.SLIDESHARE.NET/NORDICAPIS>)



INSTAGRAM (<HTTPS://WWW.INSTAGRAM.COM/NORDICAPIS/>)



RSS (<HTTP://NORDICAPIS.COM/FEED/>)

© 2013-2018 Nordic APIs AB | Supported by  CURITY (<https://curity.io>) | Website policies (</policies/>)