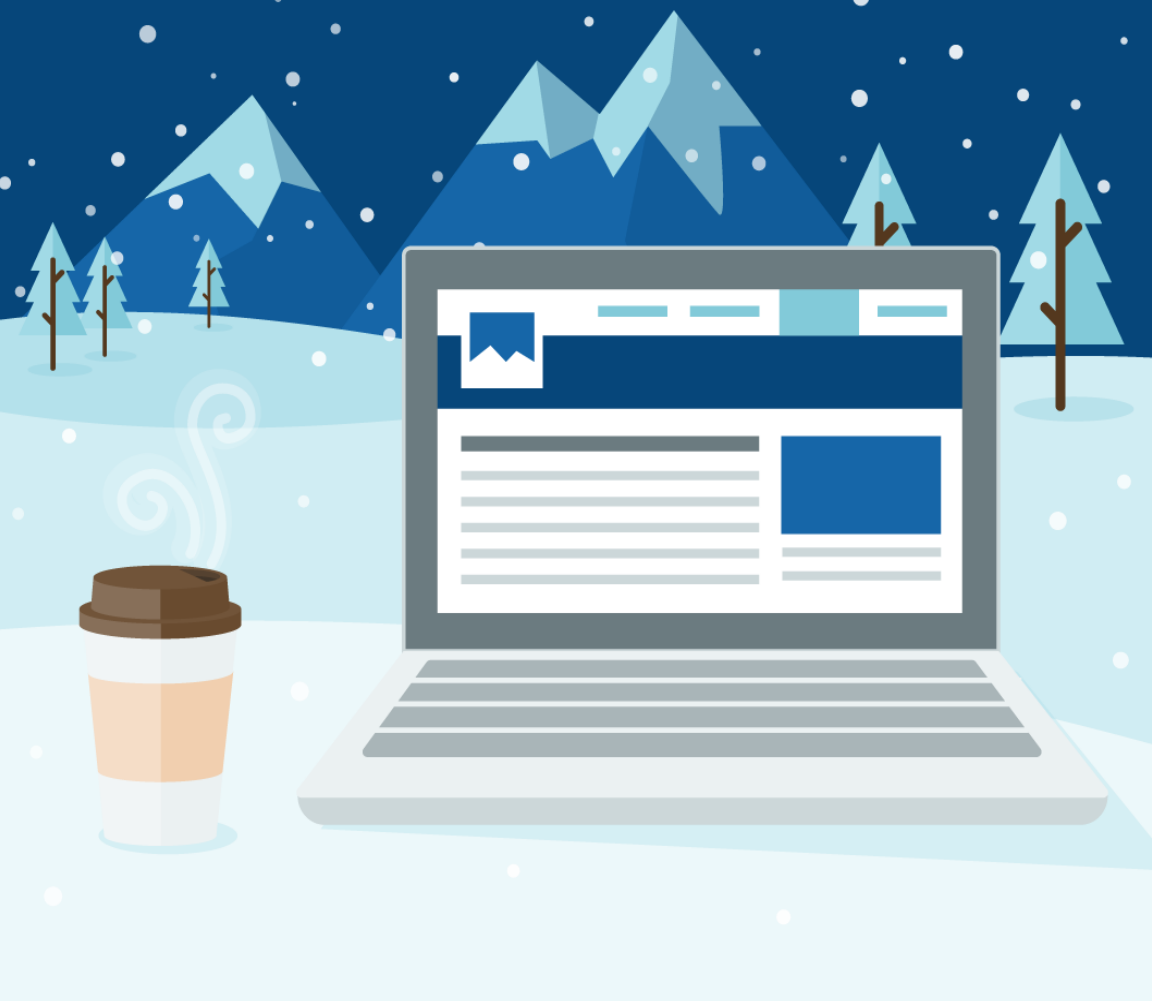
WINTER COLLECTION

A selection of posts from the NordicAPIs blog



AUTHORS

Bruno Pedro
Travis Spencer
Bill Doerrfeld
Rhys Fisher
Jennifer Riggins
Staffan Sölve

NORDIC APIS
nordicapis.com

Winter Collection

The Best Nordic APIs Topics From Late 2014 & Early 2015

Nordic APIs

This book is for sale at http://leanpub.com/wintercollection

This version was published on 2015-02-27

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Preface

The cold beckoned us indoors, and we got to work.

As we prepared for our Spring 2015 Tour during the winter season, we at Nordic APIs focused on beefing up our online content. We released some of our most successful thought pieces, helping us further establish ourselves and and spread expert advice on web APIs.

We've taken the highlights — our 11 favorite blog post from the winter season — and have consolidated them in an easy to view e-book. Contributors include our experienced team of tech professionals and bloggers, spreading insights founded on hundreds of business representatives that we met with during our 2014 seminars.

So we've convinced you that APIs rock. Well, next comes actual implementation. Many themes in recent months have been **design-centric**. Google Trends shows that the phrase "**developer experience**" has nearly tripled in keyword searches since 2011.

API architects are realizing that APIs need to be conceived with their target consumer in mind— whether human or machine. This effected a trend toward discussions concerning proper API design, documentation, onboarding advice, and more. At the 2014 Nordic APIs Platform Summit, 7 **user experience** tips for consuming APIs were covered that we still reference often.

Though design is a hot subject, our Deep Dive into OAuth and OpenID Connect article steadily rises in views, proof that **security** and properly managing **user identity** are pressing concerns in the API space.

Other topics within this e-book include API advice on business modeling, brand awareness, economy projections, and release tips—

an ideal collection of tips for this year's API practitioner. These top 11 Nordic APIs blog articles released in Fall-Winter 2014-2015 season also received the most engagement from our followers, so we believe they are the most helpful to continue to disseminate.

To bring warmth to the Nordic region this season, Nordic APIs will be donating all proceeds from this e-book as well as our last e-book to the Salvation Army in Stockholm. They're doing a great job helping the homeless in that city, and we're motivated to support their effots.

If you like what you read, consider attending our Spring 2015 Tour in mid-May. We'll be bringing industry experts together at Copenhagen, Munich, London, and Seattle to discuss the entire API lifecycle.

Please enjoy our third e-book, and let us know how we can improve!

– Thanks, Bill Doerrfeld, Editor in Chief, Nordic APIs

Connect with Nordic APIs:

Facebook | Twitter | Linkedin | Google+ | YouTube

Blog | Home | Newsletter | Contact

API Design & Testing State of the Art

By Bruno Pedro

Ensuring proper API functionality raises all the following questions:

- How do you decide which endpoints you're going to expose with your API?
- How do you create all the documentation that developers will use to interact with your API?
- What is the best way to verify that your API is working as specified by the documentation, and in keeping with the developer's expectations?

- Who should be involved in modeling APIs inside your organization?

By the time you finish reading this article you should be able to answer all these questions. API Design and Testing are very hot spaces right now. This is because they are the primary means of guaranteeing that APIs not only reflect business processes, but are being used in a way that will generate value to both consumers and providers.

We'll first cover the API Design space, presenting the tools and standards that will help you offer the best possible developer experience. Then we'll dive into the API Testing space, helping you gain complete understanding of what you can test, and how to ensure that your API will keep working repeatedly, and according to expectations. Finally, various industry experts will share their opinions on what the future holds.

API Design

API Design is the act of modeling an API taking into account three very different factors:

- **Usability**: how the API will be used, by whom, and what specific use cases need to be addressed.
- **Security**: what information is handled by the API; how it should be stored and transported; who can access what data – and when.
- **Performance**: how many consumers the API will have, how long can they wait for a response, and what internal systems affect the API speed.

Designing an API can start in many ways, from brainstorming to diving directly into the code. However, it should always produce

great documentation that can be used by developers, and, if possible, programmatically by consumer apps. It's considered a good practice to start an API by documenting it and then using the documentation to generate the code that will implement it.

This article describes three tools that let an API Designer start by describing how an API works, then how to feed tools that will generate the code that makes it work.

Swagger

Swagger is the work of Reverb Technologies, a company created in 2008 that builds language-related products. One of those products, Wordnik, was the main reason they created Swagger. Wordnik API support requests were increasing and they couldn't find an easy way to document the API and support the SDK at the same time.

developer.wordnik.com was the very first Swagger power interactive documentation website. It helped Reverb define what Swagger came to be, and iterate on its original implementation until open-sourcing it in 2011. There are now over 40 server-side integrations with Swagger and an estimated 10,000 APIs are using it to power their documentation.

> [Swagger] is successful, in my opinion, largely because it is language agnostic, vendor neutral, and simple. It doesn't try to solve all use cases, and is opinionated to a degree. This helps keep the specification lean and very effective. — *Tony Tam, Reverb CEO*

Swagger is an interesting approach to API Design because it allows developers to write the API definition only once, and it automatically generates server code, client code, and the final documentation. Swagger-generated documentation is interactive. This means that developers can immediately try to call API endpoints without having to write any code.

There are different third-party tools following the Swagger specification to help developers better implement their APIs. One of these tools, Apigee-127, lets you model your API using the Swagger editor and generate the code that actually implements the API methods described in the documentation. This is an interesting approach because it lets you easily iterate on your API Design until you're satisfied with the result.

RAML

RAML was initiated in 2013 by MuleSoft when they became "frustrated with the lack of a clean, clear way to describe practically-RESTful APIs such that you could build great experiences around that description", says Uri Sarid, MuleSoft CTO. RAML was created with the goal of assisting the API Design process with a special focus on letting you reuse the documentation.

RAML's main goals are to provide a specification that allows you to create documentation that is humanly readable, simple to understand, and can be broken down by patterns. By meeting these goals, RAML offers a unique combination of features that allows developers to easily extend it by building tools.

> RAML isn't a product [...], it's really about an API developer wanting the ability to "whiteboard" an API, build that API, then manage and provision it. — *Sumit Sharma, MuleSoft Director of API Solutions*

RAML development is used by a broad group of companies including MuleSoft, Intuit, Box, PayPal and Cisco. This has contributed to the creation of a consensual specification that is now in use by numerous large-size companies. According to SOA Software RAML is among a short list of API specification formats gaining adoption in the enterprise space.

There are now hundreds of API portals using RAML and a combination of compatible tools. One of these tools, the API Designer lets you easily edit RAML using your browser. This tool, combined with a code generation utility like the JAX-RS Codegen, will help you quickly iterate on your API Design.

API Blueprint

Apiary, an API Design company, created the API Blueprint at the turn of 2012/2013 because they found that there wasn't a common and simple way to describe APIs. API Blueprint is unique because it uses the popular Markdown format that most developers love.

> Our goal [...] was to create a fundamentally very human-readable/writable format that was easy to use, but also powerful in what could be generated out of it.
> — *Jakub Nesetril, Apiary CEO*

The API Blueprint format is available to use and modify under the MIT license which lets any individual or company build on top of it, even commercially. Among these are SmartBear, a company that builds API-related tools. SmartBear offers an API Blueprint for their popular SoapUI Pro tool which lets enterprises easily test their APIs.

There are now more than 100,000 API Blueprint-powered documents in the wild, making it probably the most used API documentation standard. The number of available tools also makes its adoption very easy and straightforward, letting you render documentation into a number of formats, generate code, test API implementations, and even generate API Blueprint documents from existing code.

API Testing

API Testing is often associated with API Design because it's a way of closing the iteration loop. After your API is modelled,

you generate the server code and then run exhaustive tests to understand how it will perform in the real world.

You should focus on three factors when testing your API:

- **Usability**: how easily can your API be used by developers. This kind of testing is often performed by multiple developers with different backgrounds.
- **Performance**: how fast your API can respond to requests. This test can be performed by an automated testing tool like the ones described in this article.
- **Compliance**: how compliant your API is to the documentation. (Can developers easily follow the documentation and make requests to your API?)

Remember that testing is an ongoing process that you should run repeatedly in an automatic fashion, if possible. If you don't proactively test your API someone else will – and, if they find any issues, it can create a negative marketing impact on your organization.

Let's look at two tools that help you go from API Design to API Testing in almost no time. We describe POSTMAN, a browser-based application, and Runscope, which offers cloud-based testing.

POSTMAN

POSTMAN was created by Abhinav Asthana in 2012 as a strategy for working more easily with APIs. Abhinav was working at Yahoo! at the time, and couldn't find any tool that would let him easily test API calls and share the results with other developers.

> At that time I tried out tools like Curl and I wasn't very happy with them. […] What I saw as a basic problem was the inability to communicate to other developers

> how a particular API worked. — *Abhinav Asthana, POSTMAN Founder*

POSTMAN is available as an installable Chrome app, which is now being used by more than 600,000 developers. POSTMAN also offers some interesting tools such as Newman which lets you run POSTMAN test collections from the command line and the Interceptor, which makes it easy to intercept and inspect any HTTP call made with Chrome.

Runscope

Runscope was launched in 2013 as a way to easily inspect API calls traffic. John Sheehan joined forces with Frank Stratton to solve common problems they were finding while working on their previous jobs at IFTTT and Twilio.

> Runscope was born out of the frustration I was experiencing working with so many different APIs [...] Every day a new one would break, have performance issues, or be impossible to integrate with. — *John Sheehan, Runscope CEO*

Runscope is now built and maintained by a team of 13 and they're growing rapidly as they see increasing demand. John Sheehan believes that APIs are "the one technology that is common to every language and framework and every part of the stack from infrastructure to end user", especially on enterprises.

The Future

There are still many challenges to be solved in the API Design and Testing space. This requires that tools evolve to bring more value

to the table by allowing people with different skills to collaborate in the definition and verification of APIs. Kin Lane, who's been working in the technology space for more than 20 years, and is best known as the API Evangelist, believes that "[…] we are going to need a robust stack that helps everyone participate in the API design process, including non-developer business owners". Abhinav Asthana, from POSTMAN, also agrees that "if all stakeholders who are involved in the development and testing of APIs can collaborate faster, APIs are going to be more robust".

Another big challenge is how developers should cope with the growing number of APIs that organizations are releasing. With so many releases there will be new "API products [and] ensuring that they can create synergy with the existing APIs can be extremely difficult", says Jason Harmon, Head of API Design at PayPal. API Design applications should provide features that let people define business logic without having to worry how different endpoints talk to each other.

> We should all be working hard on business logic and let tooling take care of the plumbing. — *Tony Tam, Reverb CEO*

APIs should also be treated like any other product where you are "[…] using metrics & feedback to understand customer behavior, and use it to make product decisions about how to evolve your API", says Jakub Nesetril from Apiary. By following a metric-centered approach you can better align your API with the needs of developers and, in the end, improve the way your customers interact with your app.

A good question when speaking of metrics and customer behavior is what is considered success. In the product development space success is often associated with more actions being performed, or with less time being spent to perform them. While this is still a blurred vision in the API space, John Sheehan from Runscope

affirms that "[...] it's going to be possible to test and monitor an API in the future without having to define up front what success looks like".

As you can see, the future looks very bright and we'll almost surely be looking at better tools in the next few years. API Design and Testing is playing a bigger role in businesses and is expanding out of the technical realm.

Conclusion

API Design is not a job exclusively for technical people. Because APIs can affect how businesses operate, different people inside organizations should be able to contribute to the process of modelling APIs. You should now have a good idea about which tools you can use to design your API and, at the same time, produce ready to use documentation.

We've covered three ways to design APIs that range from a more technical approach to using a document format that any person can read and understand. Swagger is interesting because it lets you easily generate server and consumer code while not being difficult to understand. RAML is more positioned at the enterprise level, and offers a very comprehensive standard that lets you specify almost all aspects related with an API. The API Blueprint standard is probably the easiest, as you can use Markdown to write your API specification which is, at the same time, its documentation.

On the API Testing space we covered POSTMAN, a browser based testing tool that also lets you share API calls with your co-workers, making it easy to collaborate on API consumers. We also covered Runscope, a powerful cloud-based tool that also lets you automate API testing and alerts you when something is not working as expected.

Do you know any other tools or services that help with API Design

and Testing? Share you thoughts in a comment here, on Twitter, or on Facebook.

Designing APIs For Humans

By Bruno Pedro

One of the most discussed topics at the 2014 NordicAPIs Platform Summit was **API Design** and how it can affect the way APIs are consumed. The discussions focussed on how design might affect the end users' perception of your product. A number of speakers considered this a very important topic, and several discussions occurred throughout the event. According to Jason Harmon, Head of API Design at PayPal, "APIs are starting to look more like the product and less like the technology." In other words, API Design is starting to look more and more like User Experience (UX), and less like a simple technical mapping of your app's database.

Honza Javorek, from Apiary, clearly stated during his talk that "developers don't really relate with the database style API design approach." APIs should be seen as a way to provide functionality to the end user; they should follow the same usability rules defined by Peter Morville, best known as the UX Honeycomb:

Lessons from UX

In order to design an API using this new approach, you must first answer these seven questions:

1. **Useful**: Is the API useful from an end user point of view?
2. **Usable**: Can the API be quickly used by a developer and provide easy-to-use functionality?
3. **Desirable**: Is the API and the functionality it provides something that generates desire in developers and end users?
4. **Findable**: Can the API documentation be found easily, and can developers start using it immediately?
5. **Accessible**: Can the API provide functionality that makes third-party apps accessible to people with disabilities?
6. **Credible**: Is the data provided by the API trustworthy?
7. **Valuable**: Does the API contribute to the company's bottom line and improve customer satisfaction?

Ronnie Mitra, from CA Technologies, first presented this correlation with the UX Honeycomb. He specified that "in the API space we build something on a machine for a machine to use, and this is wrong because **there are people on the other side** of API clients."

Ronnie Mitra during his presentation

Mitra believes that what is missing is a process of API design similar to the one UX designers employs. We should follow what UX has been doing because APIs clearly impact how end users experience the product. UX experts usually follow an iterative process involving initial **user research**, where they understand how users will perceive the app. This step is so important that the Nielsen Norman Group, renowned in the UX industry, considers that there's no UX without it.

The second step is related to **ideation**, or how to generate a lot of different ideas in a short amount of time. The belief is that the more ideas you have the more creative you are, and this in turn leads to a better design. Finally, the last step of the iteration is testing and **validation**. This can be done using manual or automated processes. These can range from simple user interviews to fully automated A/B tests that reveal which API features are used, and how frequently.

According to Mitra, you should focus on the ideation part of the process, and sketch as many API scenarios as possible. Coming up with different sketches helps you better understand what your API is about, and how it will finally be consumed. Sketching involves defining the supported API resources, their vocabulary, and which operations will manipulate those resources.

Sketches are the first step of a process that Mitra believes will be commonplace in the initial design phase of API products. You start by generating as many sketches as possible in order to provide a low-fidelity prototype. This first prototype can then be used by developers to test how the API responds to their requests. API Blueprint, for instance, lets you quickly generate mockups that can be tested against any existing client specification; similarly, you can create virtual APIs using SmartBear's Ready! API. This increasingly important initial step, is only the first first in the process of delivering what a client needs. Next, you should deliver a second, high-fidelity Proof of Concept (PoC). This can be done by launching a pilot inside the company (if your API is an internal one), or within a preselected number of users (if the API is public). With other technologies such as Swagger or RAML, you can achieve similar results.

What Consumers Want

At the end of this process, you should have an API that is focused on what developers and end users need. Including this process helps you create an API that not only looks much simpler, but is much easier to use. APIs should not look complicated because, according to Johannes Lundberg from 46Elks, "the real value is on removing the complexity for your API consumers."

Lundberg believes that API features are 'sticky.' This means that you shouldn't remove them at some point in the future. Removing or changing a feature might break any existing client's implementations, and that *must be avoided*. Harmon adds that thinking about

how different APIs connect with each other is more important than defining a single API specification. System thinking is needed rather than a one-off thought process.

A possible way of closing this gap between what the API delivers and what consumers really want is to adhere to the HATEOAS design more strictly. HATEOAS, or **H**ypermedia **a**s **t**he **E**ngine **o**f **A**pplication **S**tate, says that consumers interact with APIs entirely through hypermedia, which providers generate dynamically. By employing hypermedia techniques, providers can offer hints on how to use the API by inspecting the responses themselves, making clients more adaptable to any particular change. Although this strategy frees providers from having to deliver and maintain high-quality documentation, it pushes the responsibility of understanding the API to client implementations.

Several speakers mentioned HATEOS and how it might help the API industry evolve into a more consumer-oriented space. Jakob Mattsson, from FishBrain, believes that "a consumer should interact with the application through the hyperlinks provided in the responses" and that "it shouldn't require prior knowledge to navigate the API response structure." While we're still far from this reality, Mattsson suggests three rules that you should follow if you want to transition quickly:

1. **No fixation**: You should *not* use fixed resource names or hierarchies.
2. **No types**: You should use standardized media types and relation names, not the ones you create.
3. **No prior knowledge**: You should use a single entry point, where the state is driven by client selection on server-provided options.

The use of *standardized media types* is probably the most important of these rules. According to Irakli Nadareishvili from CA

Technologies, "the key is the media type." Nadareishvili believes that standard media types are what clients will use to understand what the API is about, and what is the expected response format. Clients will obtain that information from media types and adjust accordingly. Familiarity is very important, and that can only be obtained with a set of standardized media types that both providers and consumers understand.

While the future looks bright, we are still a long way from our goal. Pau Ramon from Redbooth believes that HATEOS will not be broadly used anytime soon. Ramon's advice for now is to "bet on flat APIs" that provide responses with the requested data and nothing more. Harmon is not so radical – and his advice is to "be smart about the size of resources because there's no perfect answer to this."

Conclusion

Designing APIs for Humans shouldn't be overly complicated, but it should be done following some proven process. The Nordic APIs Platform Summit provided evidence of a consensus confirming that API practitioners should follow the same procedures UX experts have been using for decades. In the end, APIs are going to be consumed by real people. Therefore, it follows that UX principles should be applied.

On the technical side, things are still unclear as to which standards and paradigms should be followed and when. On one side are the HATEOAS and Hypermedia followers who want to push the standardization of APIs that can be fully consumed without any human intervention. On the other side are the majority of API practitioners, who believe that the current standards will live for a long time – and are sure there is no easy way to implement a 'one-size-fits-all' API.

Wherever you stand on this controversy, one thing is clear: APIs

affect your business' bottom line and the way your product is perceived by real customers. This makes it imperative that you pay attention to API design, and follow the UX principles outlined above.

Do you have a different opinion? What's your thought on this? Leave a comment here or get in touch to discuss this more!

API Security: Deep Dive into OAuth and OpenID Connect

By Travis Spencer

OAuth 2 and OpenID Connect are fundamental to securing your APIs. To protect the data that your services expose, you must use them. They are complicated though, so we wanted to go into some depth about these standards to help you deploy them correctly.

OAuth and OpenID Connect in Context

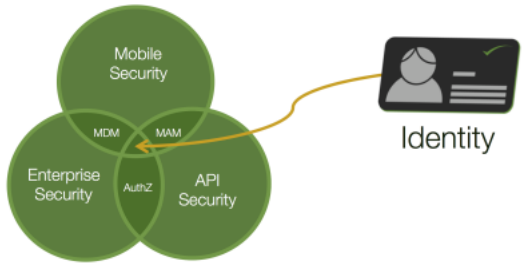
Always be aware that OAuth and OpenID Connect are part of a larger information security problem. You need to take additional measures to protect your servers and the mobiles that run your apps in addition to the steps taken to secure your API. Without a holistic approach, your API may be incredibly secure, your OAuth server locked down, and your OpenID Connect Provider tucked away in a safe enclave. Your firewalls, network, cloud infrastructure, or the mobile platform may open you up to attack if you don't also strive to make them as secure as your API.

Enterprise Security


API Security


Mobile Security

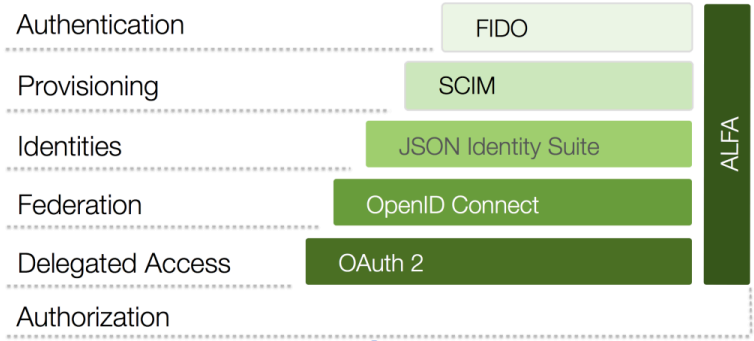
To account for all three of these security concerns, you have to know who someone is and what they are allowed to do. To authenticate and authorize someone on your servers, mobile devices, and in your API, you need a complete Identity Management System. At the head of API security, enterprise security and mobile security is identity!



Only after you know who someone (or something) is can you determine if they should be allowed to access your data. We won't go into the other two concerns, but don't forget these as we delve deeper into API security.

Start with a Secure Foundation

To address the need for Identity Management in your API, you have to build on a solid base. You need to establish your API security infrastructure on protocols and standards that have been peer-reviewed and are seeing market adoption. For a long time, lack of such standards has been the main impediment for large organizations wanting to adopt RESTful APIs in earnest. This is no longer the case since the advent of the Neo-security Stack:



This protocol suite gives us all the capabilities we need to build a secure API platform. The base of this, OAuth and OpenID Connect, is what we want to go into in this blog post. If you already have a handle on these, learn more about how the other protocols of the Neo-security Stack fit together.

Overview of OAuth

OAuth is a sort of "protocol of protocols" or "meta protocol," meaning that it provides a useful starting point for other protocols (e.g., OpenID Connect, NAPS, and UMA). This is similar to the way WS-Trust was used as the basis for WS-Federation, WS-SecureConversation, etc., if you have that frame of reference.

Beginning with OAuth is important because it solves a number of important needs that most API providers have, including:

- Delegated access
- Reduction of password sharing between users and third-parties (the so called "password anti-pattern")
- Revocation of access

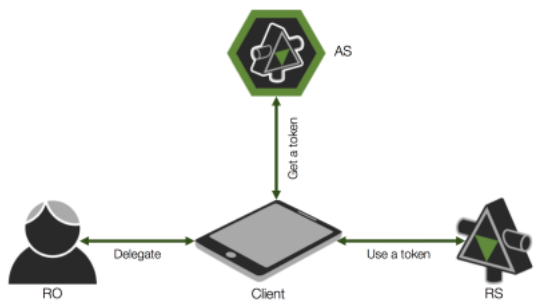
When the password anti-pattern is followed and users share their credentials with a third-party app, the only way to revoke access

to that app is for the user to change their password. Consequently, all other delegated access is revoked as well. With OAuth, users can revoke access to specific applications without breaking other apps that should be allowed to continue to act on their behalf.

Actors in OAuth

There are four primary actors in OAuth:

1. **Resource Owner (RO)**: The entity that is in control of the data exposed by the API, typically an end user
2. **Client**: The mobile app, web site, etc. that wants to access data on behalf of the Resource Owner
3. **Authorization Server (AS)**: The Security Token Service (STS) or, colloquially, the OAuth server that issues tokens
4. **Resource Server (RS)**: The service that exposes the data, i.e., the API



Scopes

OAuth defines something called "scopes." These are like permissions or delegated rights that the Resource Owner wishes the client to be able to do on their behalf. The client may request certain rights,

but the user may only grant some of them or allow others that aren't even requested. The rights that the client is requested are often shown in some sort of UI screen. Such a page may not be presented to the user, however. If the user has already granted the client such rights (e.g., in the EULA, employment contract, etc.), this page will be skipped.

What is in the scopes, how you use them, how they are displayed or not displayed, and pretty much everything else to do with scopes are not defined by the OAuth spec. OpenID Connect does define a few, but we'll get to that in a bit.

Kinds of Tokens

In OAuth, there are two kinds of tokens:

1. **Access Tokens**: These are tokens that are presented to the API
2. **Refresh Tokens**: These are used by the client to get a new access token from the AS

(Another kind of token that OpenID Connect defines is the ID token. We'll get to that in a bit.)

Think of access tokens like a session that is created for you when you login into a web site. As long as that session is valid, you can continue to interact with the web site without having to login again. Once that session is expired, you can get a new one by logging in again with your password. Refresh tokens are like passwords in this comparison. Also, just like passwords, the client needs to keep refresh tokens safe. It should persist these in a secure credential store. Loss of these tokens will require the revocation of all consents that users have performed.

Passing Tokens

As you start implementing OAuth, you'll find that you have more tokens than you ever knew what to do with! How you pass these around your system will certainly affect your overall security. There are two distinct ways in which they are passed:

1. By value
2. By reference

These are analogous to the way programming language pass data identified by variables. The run-time will either copy the data onto the stack as it invokes the function being called (by value) or it will push a pointer to the data (by reference). In a similar way, tokens will either contain all the identity data in them as they are passed around or they will be a reference to that data.

If you pass your tokens by reference, keep in mind that you will need a way to dereference the token. This is typically done by the API calling a non-standard endpoint exposed by your OAuth server.

Profiles of Tokens

There are different profiles of tokens as well. The two that you need to be aware of are these:

1. Bearer tokens
2. Holder of Key (HoK) tokens

You can think of bearer tokens like cash. If you find a dollar bill on the ground and present it at a shop, the merchant will happily accept it. She looks at the issuer of the bill, and trusts that authority. The saleswomen doesn't care that you found it somewhere. Bearer tokens are the same. The API gets the bearer token and accepts the contents of the token because it trusts the issuer (the OAuth server). The API does not know if the client presenting the token really is the one who originally obtained it. This may or may not be a bad thing. Bearer tokens are helpful in some cases, but risky in others. Where some sort of proof that the client is the one to who the token was issued for, HoK tokens should be used.

HoK tokens are like a credit card. If you find my credit card on the street and try to use it at a shop, the merchant will (hopefully) ask for some form of ID or a PIN that unlocks the card. This extra credential assures the merchant that the one presenting the credit card is the one to whom it was issued. If your API requires this sort of proof, you will need HoK key tokens. This profile is still a draft, but you should follow this before doing your own thing.

Types of Tokens

We also have different types of tokens. The OAuth specification doesn't stipulate any particular type of tokens. This was originally seen by many as a negative thing. In practice, however, it's turned out to be a very good thing. It gives immense flexibility. Granted, this comes with reduced interoperability, but a uniform token type isn't one area where interop has been an issue. Quite the contrary! In practice, you'll often find tokens of various types and being able to switch them around enables interop. Example types include:

- WS-Security tokens, especially SAML tokens

- JWT tokens (which I'll get to next)
- Legacy tokens (e.g., those issued by a Web Access Management system)
- Custom tokens

Custom tokens are the most prevalent when passing them around by reference. In this case, they are randomly generated strings. When passing by val, you'll typically be using JWTs.

JSON Web Tokens

JSON Web Tokens or JWTs (pronounced like the English word "jot") are a type of token that is a JSON data structure that contains information, including:

- The issuer
- The subject or authenticated uses (typically the Resource Owner)
- How the user authenticated and when
- Who the token is intended for (i.e., the audience)

These tokens are very flexible, allowing you to add your own claims (i.e., attributes or name/value pairs) that represent the subject. JWTs were designed to be light-weight and to be snuggly passed around

in HTTP headers and query strings. To this end, the JSON is split into different parts (header, body, signature) and base-64 encoded.

If it helps, you can compare JWTs to SAML tokens. They are less expressive, however, and you cannot do everything that you can do with SAML tokens. Also, unlike SAML they do not use XML, XML name spaces, or XML Schema. This is a good thing as JSON imposes a much lower technical barrier on the processors of these types of tokens.

JWTs are part of the JSON Identity Suite, a critical layer in the Neo-security Stack. Other things in this suite include JWA for expressing algorithms, JWK for representing keys, JWE for encryption, JWS for signatures, etc. These together with JWT are used by both OAuth (typically) and OpenID Connect. How exactly is specified in the core OpenID Connect spec and various ancillary specs, in the case of OAuth, including the Bearer Token spec.

OAuth Flow

OAuth defines different "flows" or message exchange patterns. These interaction types include:

- The code flow (or web server flow)
- Client credential flow
- Resource owner credential flow
- Implicit flow

The code flow is by far the most common; it is probably what you are most familiar with if you've looked into OAuth much. It's where the client is (typically) a web server, and that web site wants to access an API on behalf of a user. You've probably used it as a Resource Owner many times, for example, when you login to a site using certain social network identities. Even when the social

network isn't using OAuth 2 per se, the user experience is the same. Checkout this YouTube video at time 12:19 to see how this flow goes and what the end user experiences.

We'll go into the other flows another time. If you have questions on them in the meantime, ask in a comment below.

Improper and Proper Uses of OAuth

After all this, your head may be spinning. Mine was when I first learned these things. It's normally. To help you you orient yourself, I want to stress one really important high-level point:

- **OAuth is not used for authorization**. You might think it is from it's name, but it's not.
- **OAuth is also not for authentication**. If you use it for this, expect a breach if your data is of any value.
- **OAuth is also not for federation**.

So what is it for?

It's for delegation, and delegation only!

This is your plumb line. As you architect your OAuth deployment, ask yourself: In this scenario, am I using OAuth for anything other than delegation? If so, go back to the drawing board.

Consent vs. Authorization

How can it *not* be for authorization, you may be wondering. The "authorization" of the client by the Resource Owner is really consent. This consent may be enough for the user, but not enough for the API. The API is the one that's actually authorizing the request. It probably takes into account the rights granted to the client by the Resource Owner, but that consent, in and of its self, is not authorization.

To see how this nuance makes a very big difference, imagine you're a business owner. Suppose you hire an assistant to help you manage the finances. You *consent* to this assistant withdrawing money from the business' bank account. Imagine further that the assistant goes down to the bank to use these newly delegated rights to extract some of the company's capital. The banker would refuse the transaction because the assistant is not authorized – certain paperwork hasn't been filed, for example. So, your act of delegating your rights to the assistant doesn't mean squat. It's up to the banker to decide if the assistant gets to pull money out or not. In case it's not clear, in this analogy, the business owner is the Resource Owner, the assistant is the client, and the banker is the API.

Building OpenID Connect Atop OAuth

As I mentioned above, OpenID Connect builds on OAuth. Using everything we just talked about, OpenID Connect constrains the protocol, turning many of the specification's SHOULDs to MUSTs. This profile also adds new endpoints, flows, kinds of tokens, scopes, and more. OpenID Connect (which is often abbreviated OIDC)

was made with mobile in mind. For the new kind of tokens that it defines, the spec says that they must be JWTs, which were also designed for low-bandwidth scenarios. By building on OAuth, you will gain both delegated access and federation capabilities with (typically) one product. This means less moving parts and reduced complexity.

OpenID Connect is a modern federation specification. It is a passive profile, meaning it is bound to a passive user agent that does not take an active part in the message exchange (though the client does). This exchange flows over HTTP, and is analogous to the SAML artifact flow (if that helps). OpenID Connect is a replacement for SAML and WS-Federation. While it is still relatively new, you should prefer it over those unless you have good reason not to (e.g., regulatory constraints).

As I've mentioned a few times, OpenID Connect defines a new kind of token: ID tokens. These are intended for the client. Unlike access tokens and refresh tokens that are opaque to the client, ID tokens allow the client to know, among other things:

- How the user authenticated (i.e., what type of credential was used)
- When the user authenticated
- Various properties about the authenticated user (e.g., first name, last name, shoe size, etc.)

This is useful when your client needs a bit of info to customize the user experience. Many times I've seen people use by value access tokens that contain this info, and they let the client take the values out of the API's token. This means they're stuck if the API needs to change the contents of the access token or switch to using by ref for security reasons. If your client needs data about the user, give it an ID token and avoid the trouble down the road.

The User Info Endpoint and OpenID Connect Scopes

Another important innovation of OpenID Connect is what's called the "User Info Endpoint." It's kinda a mouthful, but it's an *extremely* useful addition. The spec defines a few specific scopes that the client can pass to the OpenID Connect Provider or OP (which is another name for an AS that supports OIDC):

- openid (required)
- profile
- email
- address
- phone

You can also (and usually will) define others. The first is required and switches the OAuth server into OpenID Connect mode. The others are used to inform the user about what type of data the OP will release to the client. If the user authorizes the client to access these scopes, the OpenID Connect provider will release the respective data (e.g., email) to the client when the client calls the user info endpoint. This endpoint is protected by the access token that the client obtains using the code flow discussed above.

Not Backward Compatible with v. 2

It's important to be aware that OpenID Connect ***is not*** backward compatible with OpenID 2 (or 1 for that matter). OpenID Connect is effectively version 3 of the OpenID specification. As a major update, it is not interoperable with previous versions. Updating from v. 2 to Connect will require a bit of work. If you've properly architected your API infrastructure to separate the concerns of federation with token issuance and authentication, this change will probably not disrupt much. If that's not the case however, you may need to update *each and every* app that used OpenID 2.

Conclusion

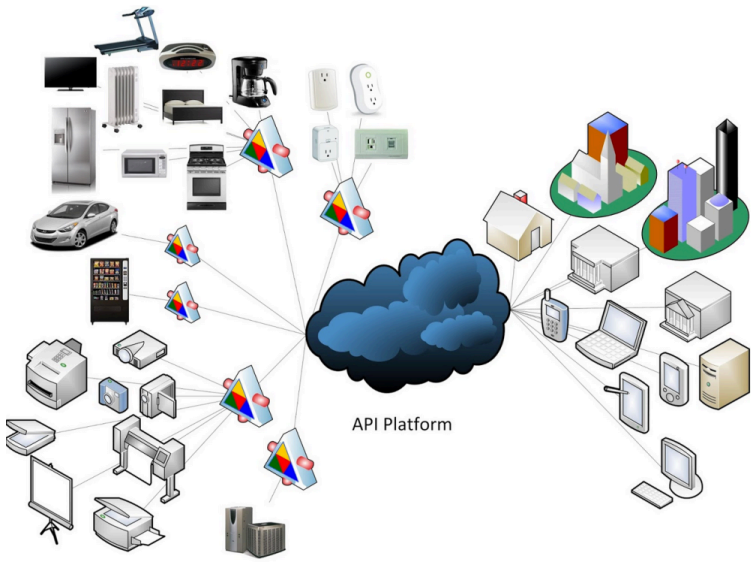
In this post, I dove into the fundamentals of OAuth and OpenID Connect and pointed out their place in the Neo-security Stack. I said it would be in depth, but honestly I've only skimmed the surface. Anyone providing an API that is protected by OAuth 2 (which should be all of them that need secure data access), this basic knowledge is a prerequisite for pretty much everyone on your dev team. Others, including product management, ops, and even project management should know some of the basics described above.

If you have questions beyond what I was able to cover here, checkout this video recording of a talk I gave on this topic, this Slideshare presentation, or drop a comment below.

APIs Power the Internet of Things

By Travis Spencer and Jennifer Riggins

We live in a Web 3.0 era. Cloud computing, mobile devices, enormous bandwidth capacity, and the Semantic Web have ushered in the next age of the WWW. The Web is nearly 10,000 days old now, and the predictions of Kevin Kelly (and others) have come true. The Web is now *one*, enormous, global computer. Our devices – be them phones, tablets, TVs, watches, glasses, etc. – are all *portals* into this single supercomputer, known as the Web. In the 9,215 days since the Web was invented, lightning-fast Internet access, widespread adoption of cloud computing, and everyday examples of the Semantic Web like Siri and Google Now have made Web 3.0 a reality. The most impactful change that has happened since Kelly's Ted talk in 2007, where he predicted our current reality, has been the massive explosion of mobile devices. We were starting to see it then, but not like we are today. We now have devices for our devices! Cisco is predicting that this single, gigantic supercomputer will be made up of over 50 billion connected devices by 2020. It's no wonder we're hearing more and more about the Internet of Things (IoT).



The IoT is the part of the Internet that is made up of "uniquely identifiable embedded computing devices," as Wikipedia states. Just like the World Wide Web runs over the Internet, so does the IoT. Similarly to how the Web is a mesh of computers, so too is the IoT. Like little portholes on a massive cruise ship, the billions of devices in the IoT give us access to exabyte upon exabyte of data. Without data and services, the little computers embedded in thermostats, house keys, baby monitors, trash cans, fire extinguishers, and store shelve are nearly worthless.

For the IoT to be useful, the devices that make up this mesh must be connected to the cloud. The way in which they do this is via APIs. Cloud-based services are the way in which the IoT is connected to data. APIs are the skybridge – IoT on one side, useful information and plentiful data crunching capabilities on the other. APIs make IoT useful, turning limited little *things* into powerful portals of possibilities.

APIs are Driving the Internet of Things

IDC predicts that the size of the IoT market will reach a whopping 7.1 trillion dollars by 2020. "IoT solutions are at the heart of IDC's view of the … four pillars â€" mobility, social business, big data/analytics, and cloud â€" resulting in millions of applications available to billions of end points," said Carrie MacGillivry, IDC Vice President of Mobile Services, IoT, and Network Infrastructure. With Cisco projecting that each person in the world at that time will connect 6 devices to the Web, it's easy to see the market potential. This represents a huge opportunity for entrepreneurs. What's especially interesting to us is that this market size is actually *zero* without APIs. APIs are the market enabler, in that the potential usefulness of (and thus demand for) IoT devices would be almost none without APIs.

APIs are the inter-connector which provide the interface between the *Internet* and the *Things*. JavaWorld's Andrew Oliver calls APIs "the glue and interesting part where the Internet of Things starts to become useful and more than a buzzword." APIs expose the data that enables multiple devices to be combined and connected to solve new and interesting workflows.

Solving Common IoT-related Challenges

As more and more devices are connected to the Net, however, certain recurring problems must be solved. We can either continue to address these over and over again, or we can develop common solutions and frameworks to the everyday challenges introduced by the IoT. An important part of solving these problems is addressed at the API layer. If we can make APIs interoperable, secure, scalable, well-documented, and discoverable, we have come a long way in solving many of the difficulties brought about by the IoT. We also need to find reusable ways of building secure and persistent, real-

time communication between these cloud-based services and the little devices running on the IoT.

A good example of a group trying to solve the challenges on both the API- and device-side is the IoT Services and Frameworks project of the Eclipse Foundation. The group says that they "want to provide a set of services and frameworks that application developers can use for building M2M and IoT applications." Eclipse is providing multiple such frameworks for building IoT Gateways. These open source offerings are useful because "IoT Gateways help manage the interaction between sensors and actuators, and the enteprise [sic] and cloud services. Implementing these types of gateways requires specialized knowledge in communication protocols, device management, software update, and hardware configurations," writes the Eclipse group on their homepage. Rather than building such gateways over and over again, the Elipse Foundation believes it is better to create common, reusable IoT software like their incubator projects, Kura and Mihini.

The EU's Internet of Things Architecture (IoT-A) is another example of the way in which we can solve common IoT-related challenges. The collaboration between the EU and various software vendors has produced a standard architectural design and starter-kit for ensuring interoperability between different devices. This architectural reference model is a scalable and secure solution intended to guide the design of protocols, interfaces and algorithms necessary to scale the IoT. The group identified similarities across different domains, systems, and application areas, and then designed the candidate architecture to suite all of these. This common framework will hopefully lead to increased collaboration and innovation.

Overcoming the Security Risks of the IoT

One impediment to this invention and newness is security. Besides interoperability, security is the biggest challenge to the success and

acceptance of the IoT. Researchers at the University of Michigan recently showed how networked traffic lights are vulnerable to cyber attacks; hackers breached Internet-enabled baby monitors, harassing parents and young children; and U.S. Senator, Brian Schatz, sited a recent study that found 70% of all IoT devices are vulnerable to cyber attacks. As the IoT grows to include devices designed to safeguard life (e.g., baby monitors, car brakes, and traffic lights), insecurity becomes an untenable risk.

IETF, the standards body behind many of the Internet protocols we use every day, says in an information note that "the Internet and the IoT domain still do not fit together easily. This is mainly due to the fact that IoT security solutions are often tailored to the specific scenario requirements without considering interoperability with Internet protocols. On the other hand, the direct use of existing Internet security protocols in the IoT might lead to inefficient or insecure operation." When the group analyzed the threats and vulnerabilities facing the IoT, they identified a large set of scary issues. With an extensive list of dangers facing all-IP networks of things, it is clear that this is a major obstacle to the growth of the IoT. The IETF presented a number of security profiles and next steps in defining security standards that will help us overcome these issues.

The IoT API Platform Play

Overcoming the challenges presented by the IoT is hard. There's lot of protocols to know, computer engineering to do, odd legacy systems to deal with, and a bunch of bit twiddling. Providing an API that achieves critical mass is also really hard. There are additional protocols to learn, a different set of requirements to satisfy, and you will be coding at a much higher level. The differences are so great that you'll essentially need two different teams – the IoT and Web API teams – complete with developers, testers, product owners, marketers, sales people, etc. Acquiring all this talent in house is one way to go, but there are other strategies.

A leaner approach would be to provide the very best IoT solution or the greatest API platform. Then, partner with others who have chosen the other alternative. Choose the former if your company is specialized in embedded programming, computer engineering, and has expertise and business opportunities in the IoT space. Use something like Eclipse's IoT framework to provide a gateway service to your customers. Let them consume this rudimentary API, so they can incorporate it into their platform.

If you have expertise in building scalable Web applications but not computer engineering, take the API platform route. This is the harder of the two options, but it also has the biggest reward if you're successful. In this strategy, you will need to balance a two-sided market. On the one hand, you'll have a bunch of IoT companies and, on the other, you'll have people who want to use those devices.

As a platform, you would add value on top of the IoT device suppliers and deliver that via an API. Additional value could include things like:

- Searchability across multiple IoT device suppliers
- A well-designed and documented API that is easier and cheaper to integrate
- Valuable computations calculated using device data
- Visualizations that span various IoT gateways

Examples of companies that have built API platforms around devices include Twilio, 46Elks, and Evercam. These API platform pioneers are standing in front of service providers that are exposing SMS capabilities and Closed Circuit TV (CCTV). This strategy is also being employed by various energy companies. They are working with various device manufacturers who have developed smart bulbs, plugs, meters, etc. and are exposing these as APIs. This API platform play is a greener business strategy for energy companies; some of them are seeing that the IoT, their brand strength,

and customer bases put them in a great position to capitalize on the opportunities that smart cities present. This opportunity isn't reserved for very large multinational energy companies, however. Twilio started small, and look at them now!

Conclusion

We are now using version 3 of the Web. In the 25 years since its inception, the Web has grown into a single, supercomputer. The billions of devices that are connected to it are peripherals that give us access to its extraordinary capabilities. With its unfathomable amounts of memory, CPU power, and hard drive space, we can use it to do things that were unimaginable a generation ago. Leveraging it to perform tasks and automate workflows now involves many different devices.

These devices are getting smaller and are being embedded into everything. This Web of things is presenting entrepreneurs with immense opportunities. To capitalize on this prospect, you must expose these devices via APIs. Using frameworks, patterns, and best practices that are being developed by open source groups, software vendors, and governments alike, the IoT is becoming more and more of a reality every day. As these reusable solutions take shape and the security and interoperability challenges are overcome, the next era of the Web will unfold. Opportunistic organizations have a chance *right now* to get out ahead.

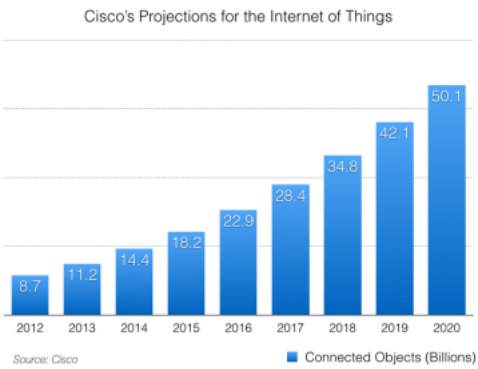
The primary opportunities that are at hand are as an IoT gateway provider, an API platform, or both. The later is beyond hard, and something you should only grow into. The first two are the more feasible possibilities. The first – becoming an IoT gateway provider – is a good option if you have computer engineering and manufacturing capabilities. If you don't, becoming an API platform in the best choice. This option also isn't easy, and will require you to bring together two different markets in order to reach critical mass.

So, you tell us – What opportunities does the IoT present? What other frameworks exist for building IoT gateways? How else are APIs driving the Internet of Things? Who is creating awesome APIs for the IoT? Let us know on Twitter or Facebook.

Designing APIs for Machines

By Bruno Pedro

The number of Internet-connected devices is growing at an astonishing rate. According to Cisco, in 2012, there were about 8.7 billion connected devices! In 2013 alone more than 280 thousand new devices were connected every hour, on average. This year, that number grew to more than 360 thousand new devices per hour. Cisco predicts that six years from now there will be more than 50 billion total connected devices making this new mesh the largest available network ever.



This prediction creates interesting connectivity challenges and, more importantly, lots of API- related challenges. Irakli Nadareishvili, who presented CA Technologies at the NordicAPIs Platform Summit, believes that "you'll be writing **APIs that millions of devices depend on**. If you break your API, millions of devices will break."

While cloud-based software is easily updated if there is an API change, things could be different where devices are concerned. To start with, devices have much more limited storage and processing power. Just as an example, Arduino Uno, which is one of the most popular DIY devices available, has only 32 KB of memory, and operates at just 16 MHz. This means that most of the instructions will have to be hardcoded into the hardware itself, and making changes will be extremely difficult.

Another thing to remember is that after devices are deployed it will be also be difficult to perform software updates. Updating devices remotely will, most certainly, become prohibitive due to bandwidth costs. Furthermore, performing a manual update is only possible if devices are within easy reach – which is not always the case.

Designing APIs that Last

An interesting comparison can be drawn between software development and civil engineering. While civil engineering worries about the longevity of solutions, software development is more focused on how the solution can be evolved and reshaped over time. Eric S. Raymond, a well known open source advocate and author of the book the Cathedral and the Bazaar, popularized the "release early, release often" motto followed by millions of developers today. This philosophy is perfect for cloud-based software development but proves disastrous in an environment where machines will be the main consumers.

According to Roy Fielding, one of the creators of the HTTP specification, most software is built following the assumption that there is a *single entity* controlling the whole system. In the case of the Internet of Things, the whole system is more distributed, and there is no single, central, controlling entity. This makes it hard for devices to consume APIs that haven't been built for a distributed world. A possible solution is to follow our previous steps, and use

a *proven* technology that allowed us to arrive where we are today. One part of the Web architecture that offers longevity and works well in a distributed world is **hypermedia**. In fact, "the Web as we know it is nothing more than millions of hypermedia entities interacting with each other", says Nadareishvili.

Hypermedia offers better longevity than other solutions because it decouples server implementations from the way clients consume APIs. Jakob Mattsson, from FishBrain, believes that "the only thing clients really need is a generic understanding of hypermedia." There will be no need to change client implementations due to changes on the server, because the clients will adapt themselves. To make that happen, API responses should include data and also controls that describe API affordances. Clients will then read those controls and find their way on the list of possible affordances. Clients will, in fact, behave like we humans do when consuming a Web site. Whenever a Web site changes, we don't need to read any documentation, we simply browse and find our way.

Making Machines Think?

The challenge with hypermedia is that, while we humans are very good at adapting to changes and finding our way, machines are not. While designing APIs for humans should be about understanding how end users will interact with your API, designing APIs for machines should be all about making responses easy to process. The key to this challenge is finding familiarity among similar resources. By defining a set of similar affordances for similar resources, it is possible to create a vocabulary that machines are able to understand. This has been the strategy behind user interface design for decades.

> "Every time I get stuck, I just think: what if this was a Web site?" – Irakli Nadareishvili, CA Technologies.

The main difference between Web sites and APIs, according to Nadareishvili, is that "most Web sites are consumed by real people that can understand semantic meaning and learn how to perform actions on it." The fact that machines do not have the ability to interpret semantic meaning the way people do is defined as the **semantic gap**. One solution is to use Artificial Intelligence techniques. If machines are enabled with the ability to understand a limited vocabulary, they can go on to derive the appropriate actions from it.

RFC6906 describes a way to accomplish this by defining **profiles** that specify how servers and clients communicate a set of semantics associated with resources. A good example of a profile is the podcast. Podcasts have a very specific list of semantics and associated affordances. While it should be possible to consume a podcast using any client, a client that is aware of these semantics is able to provide a more sophisticated experience. A growing number of profile-related standards exist to implement the required semantics. The following are two of the standards that deserve special attention:

1. **XMDP**: the XHTML Meta Data Profiles format is used for defining HTML meta data profiles that are easy for both humans and machines to read.
2. **ALPS**: the Application-Level Profile Semantics specification is a data format for defining simple descriptions of application-level semantics, similar in complexity to HTML microformats.

It's All About the Standards

Although ALPS is becoming the standard for machine-readable profiles, Nadareishvili says that "there's a lot of opportunity for collaboration." The key is to use the appropriate media type so that clients can adapt accordingly. It's expected that some clients will

only understand certain media types, and will simply discard any profile information they don't understand. Ideally, clients should parse, and possibly cache, all information about interesting profiles.

How this can be done by billions of low-powered machines with limited connectivity is still an open question. Nadareishvili believes that "devices can use MQTT and even lower-power communication." Connectivity protocols like MQTT are designed to be abstracted from the communication layer, and are specifically well suited for limited or intermittent connectivity. Another option is to make extensive use of local caching, and implement communication protocols in a very efficient way.

Another obvious solution is one that allows API management providers to play a bigger role. API management services can provide a *middle layer*, and translate API calls between servers and machines. This will supply API endpoints that machines can consume – plus a guarantee that they will never change. These translation services can then parse the profiles themselves, and make the necessary adaptations for the consumer. Questioned about this, Nadareishvili expressed the belief that API management providers will eventually offer such a service as soon as the market demands it.

Conclusion

There are huge differences between cloud-based and device-based API clients. The old "release early, release often" philosophy that makes so much sense in the startup world is not as workable when providing an API for mass consumption by billions of low powered devices. Because it is so difficult to update code on deployed devices, APIs cannot change. This makes their maintenance an interesting challenge.

One of the possible solutions is to employ hypermedia-related tactics when designing APIs, so that device-based clients can easily

adapt to any changes. By defining the appropriate media types and profiles, API providers can offer the client clues about what affordances they support, and the best way to consume them. Clients, on the other hand, will be required to parse and understand those hints in order to adapt accordingly. The big challenge in all this is that with limited processing power and bandwidth, how will all those billions of devices gain that capability?

How do you think this challenge can be solved? Tweet to us with your thoughts!

How to Spark API Adoption with Good Documentation Practices

By Bruno Pedro

While almost everyone seems to agree that documentation is of key importance when launching an API, little is said about how it can actually affect API adoption. Jeffrey Hammond, a Principal Analyst at Forrester, claims that "adoption patterns are shifting towards developers," giving them the power to "block or aid the adoption of software." This means that a significant way to persuade decision makers to choose your software over your competitors' is to gain the trust and confidence of developers.

One of the best ways to increase developers' awareness of and interest in your product is to make your API as immediately usable as possible. This begins with the documentation.

How Does API Documentation Affect Adoption?

Documentation plays a central role in the way an API is perceived by developers. Poor documentation is often a sign of a badly maintained API – one that developers will try to avoid at all costs. The more you focus your API documentation on the developers, the more it will build their confidence and improve their experience.

Onboarding Experience

Begin with the first impression. It shouldn't take a developer more than five minutes to understand your API and begin using it. The only way to make this a reality is to provide a streamlined and concise onboarding experience. Documentation should quickly take developers to a stage where they're already using the API with little or no effort.

One possible way to make this happen is to create a "sandbox environment," where developers can play with the API without actually hitting your production servers. This allows you to offer a minimum signup process, asking only what is really needed instead of a lengthy registration process that can drive many developers away.

Good documentation should clearly inform developers of what they must do to get started – and how to do it. If your API works with API tokens, generate one on-the-fly and let developers use it right away. If you use OAuth, provide fake consumer information, and even an *access token*, so developers can start making API calls immediately.

Remember that this is only the first step of engagement, and developers are still evaluating your API. You should let them experiment as much as possible, but without compromising your production systems or any real user information.

The Power to Experiment

To let developers experiment with your API you can follow any or a combination of the following strategies:

1. **Offer an API console**: This is the minimum you should offer as an experimentation tool. With an API console, developers are encouraged to immediately test what they see in the documentation, and see real API calls taking place.
2. **Offer an SDK**: Publish an open source SDK in as many programming languages as your target developers use. Provide comprehensive SDK documentation and make it easy to install and configure. Popular ways of distributing SDKs include npm for Node.js, Packagist for PHP, RubyGems for Ruby and PyPI for Python.
3. **Publish tutorials**: Begin with a *Quick Start Guide* and follow with tutorials showing them how to implement interesting use cases with your API. Provide sample snippets using the SDKs in as many languages as possible, so that developers can simply copy and paste the code and communicate with your API with minimum effort.

By following this approach you will offer a rich and comprehensive documentation with all the tools that a developer needs to get started immediately. Developers will be able to choose their favorite programming languages and follow your tutorials on how to implement the specific use cases they're looking for.

Building all this from scratch is no easy task. Documentation is something that needs a lot of focus on detail, and should continually reflect the latest API changes. Our advice is to always follow the industry standards and use proven method and tools.

Tools that Help you Build Great Documentation

Among all the API-related tools, documentation is probably the area showing the most growth. This is particularly interesting because documentation is traditionally something that developers pay little attention to when launching code. There are now several standards and tools that will cut down documentation implementation time dramatically.

Swagger, for instance, is an open source tool chain that lets you easily create interactive documentation. Apigee is using Swagger on its Apigee-127 toolkit. Apigee-127 is a model-first toolkit for building rich, enterprise-class APIs that run on any PaaS provider that supports Node.js. To use the toolkit, you start by modeling your API with a built-in Swagger editor, and from there your API code is automatically generated.

RAML, or *RESTful API Modeling Language*, is a specification and a set of tools that lets you model your API and provide documentation from it. It has been gaining a lot of adoption in the enterprise space, probably because it follows three main principles: it's human-readable, simple, and can be broken down by patterns.

With an even more human approach, API Blueprint lets you write your API specification in Markdown and serve it as your documentation. The same Markdown file is also used by a range of tools to generate code, run integration tests and debug the API. The number of ways API Blueprint can be manipulated is impressive as there are more than 15 tools that can convert it into other formats.

Read the docs is a hosted documentation service that lets you write documentation without worrying about hosting it yourself, or maintaining its changes. Without concern for these details you can simply focus on the quality of the documentation. In the end, this is what matters most. In addition, an interesting feature is their webhook support, This feature allows you to easily connect your

version-control system (e.g. GitHub), and initiate a documentation update automatically whenever you do a commit.

Conclusion

Since documentation is more and more the main driver for adoption, leading the way for developers to understand and appreciate your API, it should be the most important thing on your agenda. According to numerous authorities in the API space, developers are playing an increasing role in the decision-making process when considering a new product or service.

Documentation is the face of your API and is the first thing developers see when they land on your web site. You should make their experience as smooth as possible and offer an engagement process that can start with an easy and quick onboarding, and drive them into experimenting your API by implementing their preferred use cases. You should consider offering SDKs in popular programming languages, and a range of tutorials and guides to make integration implementation a very smooth ride.

In the end, if developers' opinion about your API is positive, they will recommend your product over that of the competition. The result is that you will win new customers!

Do you have more ideas or examples of ways to improve API Documentation and increase adoption? If so, let us know on Twitter or Facebook!

3 Steps to Increasing Brand Awareness With APIs

by Rhys Fisher and Staffan Solve

Ever wondered what's the most efficient way of building a powerful brand? If you ask this question to various marketers, a common reply will be inbound marketing. We've been wondering though if there's another way, using strategies that haven't gone mainstream yet.

Doing inbound marketing right means spending time, money, and effort to craft compelling content. When and if it's discovered, this kind of material builds trust with customers. If this is how you're currently building your brand, then hopefully you're trying just as hard to promote your content – pushing it through channels such as social media, email newsletters, your blog, and YouTube. Doing all this comes at a cost, one that's rising as it becomes more challenging to be heard online. It's hard to punch through the noise with good content, so forget about trying with mediocre stuff!

Is there an alternative channel that's less crowded, but can still deliver brand engagement?

The answer is yes – APIs!

The basic idea is to build an API, and use it to create a distribution channel. This will make it possible for customers to engage with your media and data in various new and useful ways. By owning the channel, you can brand the content at the point of consumption, and promote your brand in a new light.

Lets take YouTube as an example. Using their iframe player API, third-party websites can enhance their visitors' experience with

better video functionality without the technical complexities or cost involved. In exchange for providing websites with this free service, YouTube gets to display their brand to potentially millions of people daily – reminding the world that they rule the online video market.

In this post, we'll expand on this type of API branding strategy. You will learn how to sidestep the noise and increase your brand awareness using APIs. This strategy begins by finding a problem and developing an API that delivers a solution.

Step 1: Find a Problem Worth Solving

In the process of bringing new products and services to market, companies aim to reach something called "problem/solution fit." It basically means that you have found a problem worth solving – a problem that real people are having and not one you just assume they have.

Just like new products and services, your API needs to solve a *real* customer problem if you want it to get adopted. Finding this problem can be a little tricky because it needs to be solved with an API, and in a way that earns you the precious attention of your target audience. Don't worry though. By involving your developers, marketers, and customers in the conversation, it's very possible to discover a real API-related opportunity.

Like any good product implementation, speaking with your customers is the best place to start. Your goal is to understand what digital services they are using, what are they using them for, and what are they trying to get done. This understanding can then be profiled and fed into your API development team who will use it to keep their prospective solutions relevant.

If you've been avoiding the common API marketing mistakes and

built a healthy email list, then you might find this template email useful for booking interviews with your most engaged customers.

Subject: [Your company name] [What you'll give them] for a friendly interview Hi [name], It has been great having you as a customer. We're looking to tackle another problem that you might be facing, and I'm wondering if it would be useful to you. I'm trying to understand a bit more about your digital interactions. My goal is to learn about the digital services you use, and what you try to get done with them. Would you be available for a 20 minute in-person interview or phone call? I'm not looking to sell anything –just get your input. Since you're one of our most loyal customers, I'd love to get your input because we really don't want to build something that isn't useful to you or our other customers. If you have a few minutes, how does Thursday or Friday morning work for you? Thanks for the Help, [Your contact details]

Step 2: Build an API Solution Worth Using

Once you have found a bona fide problem, you need to find an API-related solution if you're going to use this less crowded channel to increase brand awareness. Start by reflecting.

Take a look at your business or organisation, from an outside perspective. Ask yourself what media, data, or functions could you bake into an API solution.

Maybe you're sitting on a collection of digital learning material, for instance, and wondering how else to use it. By letting relevant third-party websites embed that material into their websites for free via an Iframe API, you could piggyback of others audiences and promote yourself as an authority resource.

Here's another example: Lets assume you have collected data on living conditions in different cities. That could be turned into a

useful "index/comparison interface" just like nomadlist did. This tool helps freelancers choose the best cities to live in, and has generated some serious traffic since it went live. This one isn't necessarily an API, but it does show you that building something useful with simple data can deliver huge brand engagement, saving you the cost of expensive inbound marketing.

If you're looking to build an API that developers would use, just as you would study the app store market place if you were building an app, you should study what the readers of ProgrammableWeb want to know more about. There's a good chance the list of top ten tracked APIs says something about what attracts those who actually build with APIs.

Step 3: API Adoption Means Finding a Growth Model that Scales

At this stage, you've created an API that solves a real problem and captures value by promoting your brand –it's time to scale! The focus of this stage is on growing your API adoption as efficiently as possible.

It's important to understand that this stage has two primary goals:

1. lowering your cost per API adoption, and
2. increasing your acquisition rate

Sometimes, one of these comes at the cost of the other, but both require that you understand your funnel and growth model in detail. This means measuring growth with some sort of analytics package (e.g., Google Analytics). There are many analytics packages on the market, and you'll want one that allows you to perform cohort analysis and to segment your data on a user level. Here's why.

Cohort analysis allows you to compare your funnels performance changes by looking at when a customer first interacts with your digital platform. This is helpful because it allows you to see spikes in your key metrics very clearly. Then by drilling into the data, you can discover insights that help you understand what you need to do more of, and what you need to STOP doing immediately.

Being able to segment data on a user level helps increase your customer understanding – something that should be directly fed into API marketing strategy.

** It's critical to be tracking the right metrics**, and understand how your metrics are being reported. Not doing so could result in making very important decision based on funky data. It's easier to do than you might think, so bring in the talent if you haven't got the bandwidth to spend some time in your numbers.

During this stage you'll be rolling out many experiments, some more important than others, but all potentially able to deliver impressive results. In order to speed up your rate of growth, you'll need to increase your rate of learning. This means hastening the rate at which you run experiments – all without sacrificing on accuracy!

Learn how to execute experiment like a scientist, and then focus on doing so faster and faster.

In order to lower your cost per API acquisition, you'll want to spend resources optimizing your funnel at every stage, starting with where the most customers are getting stuck. The AARRR model is extremely useful and versatile for this type of exercise, but you should map your funnel using whichever model you feel more comfortable with. If your team is big enough, you could assign individual team members the responsibility of tracking key metric conversion rates.

At this stage your API is successfully adopted, growing consistently, and your brand is reaping the benefits of that exposure – time to go celebrate!

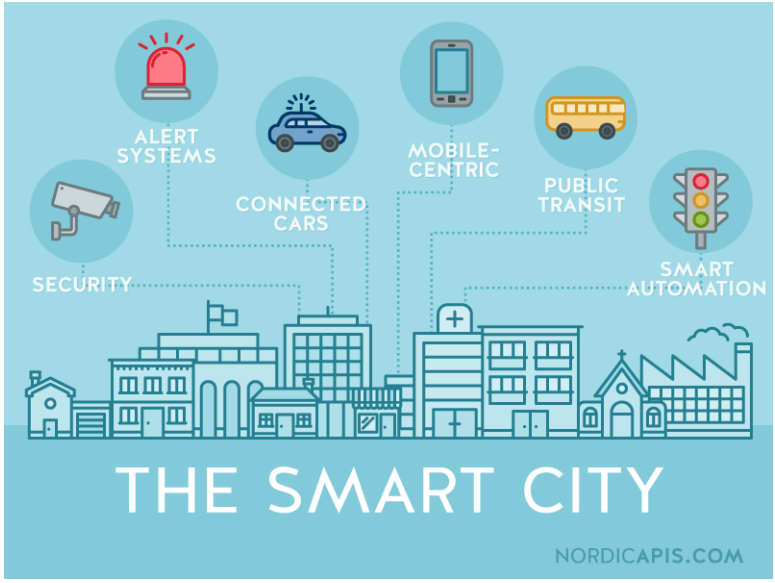
Conclusion

This post has discussed how you can use an API to deliver brand exposure by creating useful APIs that distribute data or media that tackles a customer problem. We've highlighted three stages to this initiative that align with how innovative companies launch new products and services. The first stage is to find a problem worth solving; the second is to build an API solution worth using; the third and final stage is about growing your API consumption in the most efficient way possible.

Do you have any examples of how brands have used APIs as an alternative channel for brand building? Message us on Facebook, or on Twitter.

How APIs Are Driving the Smart City

By Jennifer Riggins and Bill Doerrfeld



Application programming interfaces (APIs) are running your smart city. The bicycle rental system that suggests the best route. The city-wide FixIt app that can be used to send photographic proof of that painful pothole. The sensor at that dang traffic light that leads to that unhappy notice a week later. Your city's 311 system. Nearly everything your town council is enacting in an effort to become a smarter, more technologically savvy city involves APIs working from behind the scenes.

But the future smart city isn't just about being the most high-tech; it's much more about being the most clever.

Iemke Idsingh of Oracle's Smart City Platform believes a truly smart city is one that maximizes on existing resources. According to Idsingh, the "resources are already there—you've already paid for it, so you better make use of it."

As we look to open up APIs to connect citizen-focused products and services, what should we be concentrating on?

Speaking at the 2014 Smart City Expo in Barcelona, Idsignh said that developers and government officials working together to design a smart city should envision a city-wide nervous system. "Sense what is happening from humans, sensors, and businesses. Sharing responsibilities." Sentient infrastructures should maximize on existing data that is going unused.

Smart cities are lean, re-innovating with collaboration and shared responsibilities. "Budget constraints more and more drive collaboration, harmonization and modernization," Idsignh continued, "and social networks allow citizens to engage again, tapping into each other's creativity." The public sector must work with the private in developing and utilizing sensors plugged into the Internet of Things (IoT).

As cities are naturally unpredictable environments, the interconnectivity of APIs enable cities and citizens to predict and plan again.

What Defines a Smart City?

Possibly much broader than even trying to define what the Internet of Things is, a smart city is one that looks to use technology to make a city more affordable and more livable. A smart city can improve upon:

- public transit

- waste management
- power generation and lighting
- safety and security
- parking
- aesthetics such as fountains
- building management
- environmental factors
- border control
- tourism …among countless other areas.

No matter what factors contribute to a smart city, APIs that are creating the **interoperability** to connect them all. Because of the massive amounts of data being generated and shared across functions and departments via these APIs, smart cities are becoming more evidence-based, and collected urban data is beginning to influence policy decisions.

As lean governance shakes the city life, a smart city must be focused on preparedness instead of plans, balancing interests instead of control.

Katya Serova, vice president of the Russia-based Habidatum International spontaneous urban data software, described the new urban tech revolution as "A new culture of looking at the city not as a territory but as a space-time object that needs reaction," following citizens with real-time data.

Smart cities are based on lean planning in design and development too. Smart cities are **agile**. The interoperability of APIs enables real-time data that can reduce four or five-year plans to four or five-week plans.

And smart cities are inherently **mobile** cities.

The common spine for the security, safety and functionality of a smart city is often the sensors, but, as Donald Clark of Schneider

Electric pointed out, where these aren't installed, "the best sensor is a person with their mobile."

For that to be viable, all APIs, applications, and services designed for the smart city must be **geolocated** and integrated across functions. This empowers citizens to be consistently active in the government.

APIs Break Down Silos in the Smart City

Besides the typical bureaucratic barriers that come with enacting any widespread change, perhaps the greatest challenge facing aspiring smart cities is that so much of the data is in silos, walling off collaboration.

Amr Salem, a planner of Cisco's Internet of Things Forum, believes these silos should be broken down to integrate services between agencies. Salem argued this must be done with a "change to infrastructure first, deconstructing the nature of the multiple silos." For example, a sensor being used for traffic control should also be used to deliver data to a security center.

He further recommends that cities should open up this data to startups and corporations in order to fuel local economic, social, and environmental industry.

"In actuality, people don't want more data, they want more answers…actionable insights." According to Salem, data silos have to be broken down in a way that privileges smart analytics to encourage innovation.

The Citizen, Not the Government, Should Be at the Center of Smart City Design

As we've written about before, there are seven necessary questions to ask when designing an API with the human being in mind. These

parameters are even more crucial when designing an API for a smart city and its citizens:

1. Is the API Useful?
2. Is the API Usable?
3. Is the API Desirable?
4. Is the API Discoverable?
5. Is the API Accessible?
6. Is the API Credible?
7. Is the API Valuable?

The Massachusetts Institute of Technology MIT's SENSEable City Lab is focused on creating a flexible and accessible API for sensors in cities. Carlo Ratti, the lab's director, believes that connector modules are critical for developing a platform that utilizes many different types of real-time data. In order to accomplish this, APIs must be structured with easy usability in mind. "API development is aimed at enabling a data query mechanism that allows users with little programming experience to easily tap the data pool brought together on the platform," Ratti said.

The Smart City Isn't Just About Connectivity, It's About Functionality

"**Connectivity** is an enabler but not a necessity for a smart city. **Functionality** is the citizen who is empowered by and whose life is improved by the Internet of Things," said Clark. Creating and implementing the Internet of Things isn't just about building a brand; it's about tying everything together with not only vertical but **horizontal** integrations.

Clark compares a smart city to the human body—every part is suited for a purpose in symbiosis. The heart must be able to function on

its own, but, if the heart stops, the brain is immediately notified. If any information is delayed, the system as a whole fails. A perfect analogy to the need for real-time data exchange in a smart urban environment.

For an ecosystem to prosper, each contributing technology must be designed with low-level interoperability in mind. Clark offers the example of a barcode on a can. From the supply chain, to the vendor, and to the moment a clerk swipes it at checkout, every barcode scan is seamlessly repurposed and integrated with larger systems.

City-wide IoT Requires Open and Shared Resources

Rather than reinvent the Internet, a common technique for implementing IoT is for cities to publish on **repository hosting services** and **open data resources**, allowing other public departments, private entities, and even citizens to access and build on each city's APIs. This is exactly what the City of Philadelphia is doing.

Philadelphia stakes claim to being the largest government user of GitHub, publishing all their open data, and even selectively contracting external developers under the condition they will publish their source code.

Tim Wisnieswski, chief of the data office for Philadelphia, said that following this practice "allows us to build cheaper and quicker." Rather than increase economic competition, Wisnieswki believes this helps foster a collaborative spirit.

Smart Cities Collaborate With Each Other to Save Time and Money

Smart cities aren't racing other cities to create the bigger, better, or faster product. Rather, API-enabled **data exchange** between cities is

recognized by many as a much easier and faster route to mutually prove success, replicate, and enact new advancements. Platforms are popping up around the world to enable cities to do just that.

Bart Rousseu, from the administration of Belgium's second largest city Ghent, admits that "As a city, as a government, we're not used to getting free consulting, but these networks" of cities sharing information help. 50 percent of Rousseu's job is simply getting people on board with new technologies. However, with cities and even countries working together to share data and statistics, its easier to convince lawmakers of the value to be had in modernizing their cities.

Collaborating across borders, cities gain access to a wider resource pool. The open distribution of costs, failures, and successes associated with city-wide technological adoption could dramatically reduce the cost and development time for new smart cities across the globe.

CitySDK Looks to Use APIs to Connect E.U. Cities and Citizens

The City Service Development Kit (CitySDK), co-funded by the European Union, is an initiative to increase smart city development. The group is implementing solutions for Lisbon, Helsinki, Amsterdam, Barcelona, Manchester, Istanbul, Lamia, and Rome, and welcomes the participation of other cities.

Though an intergovernmental program, CitySDK welcomes private and corporate developers to participate as well, as interested developers often have apps that "could easily be redeveloped for other cities—if cities were more interoperable."

CitySDK aims to enable API **interoperability**, pushing for common interfaces and data sets in similar formats. This is being done by establishing:

- various European city uniform APIs
- relevant data sources for smart participation, smart mobility and smart tourism
- expansive markets that require only minimum adoption of apps in order to function
- easy-to-use APIs
- helpful resources such as code libraries, SDKs, apps, and platforms
- consolidated knowledge and experience through a global CitySDK developer community

There are three main APIs that the CitySDK is focusing on:

- Open311 API: Allows citizens to send service requests to city's feedback system. XML and JSON formats can be used.
- Linked Data API: For transport, mobility and geographical data. Returns JSON, JSON-LD, Geo-JSON and RDF/Turtle.
- Tourism API: A RESTful location-based mobile service for tourists.

According to Amsterdam's representative for CitySDK Wouter Meys, "collaboration isn't something extra but it should be simply a best business practice," encouraging city governments to "collaborate like startups."

Will API Collaboration Ultimately Produce Smarter Cities?

Only time will tell if future smart cities are the successful results of current efforts toward intergovernmental collaboration by groups such as CitySDK. What we do know is that when cities, businesses, and especially citizens open up APIs, they open up collaboration that drives innovation, and, only with innovation do smarter cities have a chance to arise.

Open Data: How to Make it Work For Your Business

By Bruno Pedro

If you have a large amount of information, you might consider releasing it to others who can also benefit from it. This is an especially important consideration if you are already providing a public API, but are not offering easy access to your full dataset. If you're not, read on to learn more about the Open Data movement and how letting your API consumers have full access to all your information can *actually* benefit your organization.

This article explores several aspects of Open Data and how your company can utilize it. You will start by learning how Open Data is commonly used, by both for-profit and not-for-profit organizations. We'll then explore two possible business models inspired by Open Data (both of which are contingent up free data access). We will also provide guidance on tools that you can use to convert the information residing in your database to formats that are more sharable. Finally, we'll help you compare the available licenses that stipulate what consumers are allowed to do (and not do) with the data you open up to them.

Status of Open Data

The concept of Open Data is generally understood as the act of making information freely available to anyone, without imposing any kind of restriction. Information is typically controlled by using a combination of copyright, patents, and other types of license

agreements. In the case of Open Data, none of these means of control are in place, letting anyone consume, distribute and use the information in any way they like. In most cases, attribution is all that is asked for by organizations that open their data.

Open Data is often used in those sectors that do not have to deal with revenues or ways of generating business out of information sharing. Major providers are scientific and government agencies throughout the world. The United Nations Statistics Division, for instance, uses on Open Data to deliver access to UN databases through its UNdata service. Information is available under a proprietary license that simply states that consumers can copy, duplicate and distribute it provided that UNdata is always cited as the reference.

Another example of a service that follows the Open Data strategy is the OpenStreetMap project, an alternative to popular mapping services that is rapidly growing. All data managed by OpenStreetMap is obtained from the community and, in return, it is freely available for anyone to consume without any restriction. The only thing they ask is that anyone who uses their information credits the OpenStreetMap project as its source. This is stipluated in the license they use: the Open Data Commons Open Database License.

Why you Should Open your Data

Obviously, opening data is relatively easy for not-for-profit organizations. However, if you're a company that needs to generate revenue you might be reluctant to allow free and unrestricted access to your information. One good reason to open your data is to provide content syndication. By opening your content to third-parties you'll be driving more consumer traffic into your app which, in turn, can generate more business. This strategy works well for product comparison apps, classifieds and vertical search platforms such as commercial flights or hotels.

Amazon is one prominent company following this strategy. It has been opening formerly proprietary data in order to drive sales by exposing product information to the largest possible number of potential customers. Their Product Advertising API is a good example of how a company can expose a lot of business-related information without losing control of how it's used by consumers. In the case of Amazon they make sure that their API terms of use explicitly defines how developers can use the obtained information. They protect themselves against possible competition by stating that only consumers who "do not have as their principal purpose advertising and marketing the Amazon Site and driving sales of products and services on the Amazon Site" can use the information. In this way, they protect against consumers that might want to do a full copy of the Amazon Website. This means that Amazon is not releasing Open Data *per definition*, but they are a good example of how to be inspired by the Open Data movement.

Twitter is another business that provides Open Data, although this time the purpose of providing free information is different. Their public statuses sample API endpoint lets anyone obtain a random sample of all public tweets. This information can be used by organizations studying behavior on the popular social network, or by companies interested in doing real-time analysis of what people are tweeting. In the first case, Twitter is able to promote its brand to people that would otherwise not communicate about it. Universities often use this endpoint as a means of doing research on topics such as sentiment analysis and natural language processing. By offering their public statuses sample, Twitter is able to tap into this audience at no cost, promoting its brand. In the second case, Twitter uses this endpoint to upsell their other commercial products that offer access to the full dataset.

How to Open your Data

Andreas Krohn from Dopter AB, during his *Building Open Data*

Platforms presentation at our Platform Summit, explained that you shouldn't "choose a technology first, but start by understanding the use cases and then adopt a technology that addresses it." An important first step when opening your data is to *identify your audience*. The way you expose your data depends on who is going to consume it, and how they intend to use it.

If, for instance, you're targeting journalists and you're publishing statistical data, you should probably make it available as a CSV file that is easily imported into Excel. If, instead, developers are your target and your goal is to make the statistical information available on third-party apps, you should expose a public API. The most important thing is to understand how your data will be consumed and to adapt accordingly.

There are several tools that will help you transform data stored on databases into different formats, including ways to expose it as an API. If you're using MySQL, there is a tool that lets you easily move information between your database and Excel. The MySQL for Excel tool lets you access a database from within Microsoft Excel, where you can easily import all your data.

Using the Appropriate License

Creative Commons provides a list of licensing options. Their list of choices will help you pick the best possible license for your information. They even offer a wizard to help you choose wiseely. While Creative Commons is usually used with artistic works, the range of provided licenses is acceptable for Open Data as well.

If you don't care how your information will be used, but want to guarantee that you will not be held responsible for its usage in any event, you can use a simple adaptation of the MIT License. While this license has been created for use with software, a few small changes to replace the word software with data can make it suitable for Open Data.

Open Data Commons is a project from the Open Knowledge Foundation created in 2008 to provide a legal framework for Open Data. It's a not-for-profit organization whose main objective is to maintain a number of Open Data related licenses. Because these licenses have been specifically crafted to be used with Open Data, they're probably your best option.

Conclusion

While Open Data is traditionally used by science and government organizations, nothing prohibits your company from benefiting from it and being inspired by it. In fact, companies like Amazon and Twitter are using similar strategies to help their businesses grow and generate more revenues. For all its benefits, it's important to use caution when employing an Open Data strategy. Although it can help your business grow, care must be taken to assure that the free information you offer does not negatively affect your organization.

Open Data shouldn't follow a specific format or standard. Instead it should be adapted to the way your consumers are going to use it. For example, opening data to a journalist involves a completely different format than targeting developers who are building third-party apps that consume your information. A number of tools are available to help you with the process of transforming information into accessible formats. Another thing that depends on the use case is the license you apply, indicating the way your data is to be consumed. Again, a range of options are available, from the most permissive MIT License to the more robust Open Data Commons. Choose the option that best fits your business objectives.

Are you already following an Open Data strategy? If not, what's holding you back? Get in touch to discuss this more!

How to Release a Free API and Get Paid Indirectly

By Bruno Pedro

API-related business models usually involve a customer actively paying to use the API itself. But consider this: What if you could allow *anyone* to use your API freely while you retain the power to generate revenue from it? And what if by doing so, you could actually increase your chances of generating even more revenue?

The free use of your API will lead to increased revenue because third-party developers will *implement on top of your API*. Consequently, your product will benefit from its increased usage. This article explores some ways of indirectly generating revenue from API usage. Read on if you'd like to understand what business models can be used to provide this indirect revenue generation, and how you can adapt them for use with your product.

Business Models

A number of different business models benefit from the fact that you are offering a free API. You will increase the prospect of consumers getting in touch with your product. As more consumers discover and use your product, the number of opportunities for additional revenue will increase.

Let's start by looking at models that are not directly associated with your product's features. Weâ€™ll then explore how to sell more of your app to API consumers.

Content Acquisition

A business model worth investigating is the *content acquisition* model.

Businesses that are based on user-generated content usually have to find ways of increasing the amount of quality content. This often involves rewarding users in some way each time they publish

additional content. Alternately, they can create a user interface that will encourage and assist users in producing more usable content.

Most businesses in this model depend on quality user-generated content that will result in more sales. Typical examples include: * Auction websites; * Travel and hotel e-commerce platforms; and * Any website that allows users to write reviews.

A free API allows third-party developers to build apps that make it easier for end users to generate content. The more third-party apps the better, because different user groups will be able to engage with your product. This produces the content that is most likely to generate more revenue.

A good example of this strategy is the eBay Selling API. Access to the eBay marketplace is implemented through the *item listing functionality*. Third-party developers can easily use this API to list items programmatically, helping eBay grow their inventory and, in the end, increasing their revenue.

Content Syndication

While content acquisition is focused on how to motivate users to produce more material, *content syndication* works the other way around. Here the goal is to open the content highway as widely as possible, allowing more consumer traffic, and, eventually, more interaction with your product.

These are typically media related-companies, but lately we've been seeing a number of other businesses employing the same strategy to attract more users. Typical examples include:

- Product comparison apps;
- Classifieds; and
- Vertical search platforms (such as flights or car rentals).

By offering a read-only API that is free and easy to use, these businesses enable parties to include their own offerings within the external apps. Developers will feel more encouraged if some sort of revenue sharing is used. In this case, the best revenue-sharing model is the *affiliate model*, in which third-party developers receive a cut of each item sold to users coming from their app.

Amazon is one example of a company that uses content syndication to distribute their inventory on as many third-party apps as possible. They do this with their Product Advertising API which provides programmatic access to their product selection-and-discovery functionality. Third-party developers are able to use this API to offer product search and detailed information, including product reviews and recommendations. As an added plus, they also receive a share of the revenue generated from sales.

Upsell

If your business doesn't revolve around content, and you have the typical list of user plans with associated features, you can still use the API to help you generate more revenue. By offering access to higher plans through the API, you encourage third-party developers to offer those features on their apps. Third-party apps users must then switch to a higher plan in order to continue using those features.

Another option is to make API access available only on certain plans. With this method, you do not charge *directly* for API usage. Instead, you are making a feature on a specific plan. This usually works for fairly sophisticated apps that offer a comprehensive set of features distributed among several plans.

Engaging third-party developers in a revenue-share plan will increase the chances of selling more to end users. Developers feel more motivated to integrate with your higher-paying features when they know they too will profit from the additional revenue you receive.

Companies like Salesforce are following this strategy. If you look at their plans page, only the enterprise edition offers API access. They generate more revenue by doing this, because users will upgrade to the enterprise plan in order to access the API.

Brand Building

Another way to generate revenue with a free API is to use it as a part of your branding strategy. Although *brand building* does not generate extra revenue immediately, your brand will be reinforced. In the long term, you will attract more customers.

Companies that might benefit from this strategy are those that are looking to expand their market to include a more sophisticated and technology savvy audience. By targeting technical people, including developers, a company stands out as innovative and trendsetting.

Absolut, the popular vodka brand, recently launched what they call ADDb, the Absolut Drinks Database. This API lets you access Absolutâ€™s drink recipe database and related assets. Launching this API, and a set of example apps, shows that they care about innovation. It also indicates that they are open to new ideas, and willing to be influenced by their own customers. If you want to know more about how Absolut used their API to build their brand and connect with their partners, check out their presentation from one of last year's Nordic APIs conferences.

Conclusion

This article proposes different ways of releasing a free-to-use API, and using indirect business models to generate revenue from it. You should now have a clear idea of whether this strategy applies to your business and, if so, which business model you should choose to implement it.

We've analyzed three main business models that are adequate for different types of apps:

1. The first one, the *content acquisition model*, focuses on obtaining more user-generated content by exposing your free API on the highest possible number of third-party apps.
2. Next, we explored the *content syndication model*, which has the goal of spreading content to as many users as possible. The strategy here is to drive content syndication by making it easy for third-party apps to consume and distribute it.
3. The third model investigates how a product can *upsell* features or plans by offering an API, or a part of it, that is only available to paying users.

Do you know of any other models that will let you generate revenue while exposing a free API? Get in touch to discuss it with the community!

How to Achieve Accelerated Growth with APIs

By Bruno Pedro

Growth is a word on every entrepreneur's mind, especially when it's associated with the value of a startup, or with how much profit it can generate. Most startups struggle, and apply many different techniques, to reach a point where they can show some growth.

This article shows you how growth can be measured, and what variables you can use to control it. It also shows you a different way of achieving higher than usual growth. You can do this by

combining traditional customer acquisition channels with an API, and making third party developers the focus of your business marketing.

Defining Growth

When discussing the various aspects of business, we usually associate growth with the generation of significant earnings that increase at a faster rate than the overall economy. Paul Graham, a long time investor and founder of YCombinator puts it better by saying that for a startup "the only essential thing is growth; everything else we associate with startups follows from growth."

So, how do you put your business in 'growth mode'? The first step toward achieving growth is to fully understand how your product or service will be used, and served to your potential customers. Only after that can you understand how to generate earnings and make them grow at an increasing rate. A good starting point – know the cost of acquiring a new customer. From there, decide how much value each new customer brings. With this knowledge, you will know how to drive the growth of your business.

Measuring Your Customer Acquisition Cost

Let's start by analyzing how you attract new customers. There are several techniques you can use to acquire them. These *acquisition channels* , as they are often called, provide a variety of costs and results to consider. Here are some channels I consider interesting to evaluate:

- Organic search, or SEO: one of the channels used most by the industry because it is perceived as a low-cost strategy.

This is deceptive: SEO may look easy to master, but in reality usually involves creating a large amount of new content that will rank well in searches, and attract new leads to your web site. SEO-related work can go from USD $100 per hour for contract work, to USD $5,000 per month for full service from an agency.

- CPC, or Cost Per Click advertising: often used with SEO (or when CPC alone doesn't provide the desired results). Depending on the product you're advertising, you can spend thousands of dollars per month on this channel.
- API: your API can become an inbound channel with the creation of third-party apps. These apps are integrated with your API and expose some or all of its features. This attracts more users to your app, which will eventually convert into new customers. If you already use your API yourself, the associated cost will be quite low compared to SEO or CPC. Plus, you can also monetize your API making this channel profitable by itself. A good example is Uber's API third-party ecosystem which features important partners such as Starbucks and TripAdvisor. By using Uber's API through these partners, more business is flowing in Uber's direction, resulting in accellerated growth.

Once you understand how much in total you're spending on customer acquisition, you only need to 'do the math' to see clearly how much each new customer is costing you. This number will vary from month to month, but you should have a defined target, and stay within that cost-per-customer limit, to avoid losing money.

Let's say you're spending USD $1,000 a month on a combination of acquisition channels and you're able to attract ten new customers. In this case, your customer acquisition cost, or CAC, will be USD $100. If your customer lifetime value is lower than USD $100 you're losing money on each new customer. If it's higher you're generating earnings.

Measuring Your Customers Lifetime Value

A way to measure Customer Lifetime Value, or CLTV, is to calculate how much, on average, each customer is spending per month, and then multiply the value by their expected lifetime in months. The obtained value will vary each month and, after some time, you should have a clear picture of where you're going.

CLTV depends on the total number of customers you retain, and the total amount they spend. Therefore, the less the churn the higher the value you'll receive from each customer. Assuming you do not change your pricing model, the best way to increase CLTV is to control and increase your customers' retention rate.

Once you understand what your CLTV is, it becomes easier to see whether you are generating earnings – or diving into losses. If your bottom line is negative, you should pay attention to your CAC and reduce it, or otherwise change your business in significant ways. If your bottom line is already positive, you're in good shape to achieve growth.

How to Drive More Growth with a Lower CAC

One easy way to drive more growth is to increase total spending on acquisition channels. This drives up the CAC. Funded companies usually use this approach to drive more growth by spending more on advertising. Is it possible to do the same while actually lowering the CAC?

Ocne you understand the dynamics of your acquisition channels, you can start to invest less on the paid ones and measure the resulting CLTV. In the beginning you might see a decrease in total number of new customers. However, if you move your attention to

the API inbound channel and use the divested money there, you should begin to see positive results that last for the long term.

APIs are becoming the number one inbound acquisition channel because they offer a lower barrier to decision-making. Traditionally decision-making takes a long time, and involves many different stakeholders, but with APIs it only takes a group of developers to convince an entire organization.

By investing in an API that can be easily integrated with third-party apps, and avoiding common marketing pitfalls you will create a sustainable customer acquisition channel that you can maintain at a reasonable cost. Not only will more customers be using your app, but you'll also be able to retain those customers for a longer period.

So, by using your API as the preferred customer acquisition channel you'll be able to create a sustainable growth with an interesting retention rate. This would be more than enough for most businesses but here we're trying to reach *hyper* growth, something on the range of ten times the growth you would usually obtain.

Engaging Hyperdrive

You can follow the same acquisition tactics explained above to attract third-party developers who will integrate your API with their apps. You can effectively improve the API acquisition channel results by targeting developers on the other channels.

First, start by measuring how much it costs you to attract a developer to use your API. You should try to minimize this cost by using less expensive channels, and by avoiding excessive marketing campaigns prematurely. With developers, it's always better to show commitment over time than to launch a big campaign and then forget about it.

Next, you should understand how much business developers are

creating for you. On average, you should be able to tell how much revenue each developer is generating. If this number is lower than the amount you spend to attract those developers, you should obviously rethink your acquisition strategy.

The less expensive way to start is by revamping your documentation and developer portal. Then you need to follow a platform strategy that allows you to provide developers with all the tools they need to integrate with your API. The SEO channel should be used at all costs to maximize the number of new developers signing up to use your API. Other channels, like CPC, can and should also be used in combination with broader developer-oriented marketing activities.

After some time, you'll start seeing more and more developers building third-party apps which will attract more and more customers to your own app. This, combined with the high retention rate obtained from integrated third-party apps, will drive your CLTV up and put you in hyper growth mode!

An interesting example is EasyPost, a company providing a shipping API that lets developers integrate with USPS, UPS, DHL and FedEx. EasyPost went live in 2013 and, according to Sawyer Bateman, it's been enjoying a 100% month-over-month growth. The trick? They focus 100% on their API, and on making developers happy.

Conclusion

If growth is the only thing that matters for a startup, finding and increasing it should be the most important business activity. You've discovered ways to measure and influence several factors that can affect growth. In addition, you should now be able to evaluate different acquisition channels and associated CAC, and be aware of all the means available for controlling or even improving your CLTV and achieving maximum growth.

Moving to faster growth means being able to control all these factors and attract more customers at lower costs, or obtaining more revenue per customer. I've shown you a way to obtain the same growth factor without incurring more costs. Simply use your API as the main inbound channel and use all the other channels to acquire developers – not customers.

Treat third-party developers as your main source of customer generation. Using this strategy, you can achieve hyper growth by obtaining a customer base with a higher retention rate.

Do you have more ideas or examples of ways an API can increase revenue? If so, let us know on Twitter or Facebook!

Resources

Here are the Nordic APIs videos of the talks referenced in this e-book, by order of appearance:

- "Scaling API Design", Jason Harmon
- "API Design Lifecycle", Honza Javorek
- "The Art of Effective API Design", Ronnie Mitra
- "The Architecture of an API Platform", Johannes Lundberg
- "Your HTTP-based API is not RESTful", Jakob Mattsson
- "The Internet of Things Challenge: Building APIs that last for Decades", Irakli Nadareishvili
- "REST Clients", Pau Ramon
- "OAuth and OpenID Connect Deep Dive, Travis Spencer
- "OAuth and OpenID Connect for Microservices, Jacob Ideskog
- "Enabling the Future of Work: Balancing a Two-Sided Market, Steffen Hedebrandt
- "Minimum Viable Platform , Marco Herbst
- "APIs are The New Dial Tone, Ben Nunney
- "Building Open Data Platforms, an Insider's Perspective , Andreas Krohn

Endnotes

Nordic APIs is an independent blog and this publication has not been authorized, sponsored, or otherwise approved by any company mentioned in it. All trademarks, servicemarks, registered trademarks, and registered servicemarks are the property of their respective owners.

- Select icons made by Freepik and are licensed by CC BY 3.0
- Select images are copyright Twobo Technologies and used by permission.
- The Mobile/Enterprise/API Security Venn diagram was created by Gunnar Peterson and also used by permission.
- User Experience Honeycomb by Peter Morville

Nordic APIs AB Box 133 447 24 Vargarda, Sweden

Facebook | Twitter | Linkedin | Google+ | YouTube

Blog | Home | Newsletter | Contact


NORDIC APIS
API TOUR 2015
MAY 11 – 15
COPENHAGEN, MUNICH, LONDON, SEATTLE
Join us in our 2015 tour and discuss the API lifecycle
More information at nordicapis.com