API AS A PRODUCT

BY
THE NORDIC APIS WRITING TEAM

WITH A FOREWORD BY
BILL DOERRFELD

NORDIC APIS

API as a Product

Tips for Running an API-centric SaaS Business

Nordic APIs

Contents

Foreword: The Potential of API-as-a-Product

by **Bill Doerrfeld**

At Nordic APIs, we've been tracking the emergence of many API trends throughout the years. On our blog and at our conferences, contributors have time and time again described new ways to generate value from APIs.

However, few subjects encapsulate as much interest as the API-as-a-Product concept. By opening up specialized software functionality through an API, companies can commoditize data and operational components on a per-call basis. Commoditized web APIs help software teams avoid reinventing the wheel for common functionally. This could be anything from payments to geolocation, artificial intelligence, weather data, log-ins, messaging, and more. You may be familiar with the phrase "there's an API for that."

I'm excited about this format, as it's arguably a more on-demand, real-time incarnation of Software-as-a-Service. The API-as-a-Product trend isn't just smoke and mirrors either — many startups are beginning to treat their API as a core offering due to its potential for scalability and explosive growth. Take the story of the boot-strapped Car Registration API, which went from zero to three million calls in under two years. Some API-first companies have even IPO-d in recent years.

Due to their nature, APIs have a lot going for them:

- **They fit the startup agenda**. Arguably, APIs appeal to the concepts of specialization and disruption.
- **They are platform-agnostic**. Using HTTP as a delivery mechanism, any developer user can integrate APIs into their

application, regardless of what platform or language they're using.

- **They can be self-service**. If set up correctly with awesome documentation and testing environments, API portals do a lot of the upfront work, requiring minimal human support.
- **APIs are standardized**: Developers are familiar with APIs. API products typically adopt a REST style and serve JSON over HTTP. OpenAPI Specification (formerly Swagger) has also made major crossroads to document these services.

However, we must acknowledge that it takes substantial effort to build and maintain a functional, self-service API-as-a-Product. There's a lot to consider. You must balance openness with charging for access. Providers must find their secret sauce of freemium, rate limiting, charging per call, or subscription pricing. You must support standard protocols and track emerging communication styles. It also helps to know your developer users well, monitor platform use, and optimize to find the correct model.

And, designing the program is only the beginning. API-as-a-Products require unparalleled developer experiences and high stability. The provider may also experience unexpected challenges, such as identifying spam users and thwarting black hats. Or, the API may struggle to get off the ground and attract a developer foothold necessary to sustain a business.

As Editor-in-Chief of the Nordic APIs blog, I've worked with our writers and contributors to explore the API-as-a-Product subject from many angles. As a result, our community has produced a ton of knowledge on treating an API as a product.

In this eBook, I've collated some of our most helpful insights to help you begin your API-as-a-Product journey. Chapters touch on all the concerns outlined above. I've tried to include high-level perspectives but also nitty-gritty details. (For example, one chapter involves calculating API-as-a-Product maintenance expenses, with fine-grained pricing estimates). Even if your API doesn't adopt a

public model, I believe it still helps to consider how you can benefit from taking a product perspective to integration.

Keep in mind this is a very new, nuanced subject. Software integration styles are changing, as are the standards for API design. This evolution, paired with industry consolidation and new abstraction layers, could significantly affect the overall strategy API products take to compete in the future market. So, take this text with a grain of salt and always stay updated with the forces at play.

With that said, happy reading, and the best of luck on your API journey!

By the way, we at Nordic APIs are big supporters of APIs. We're always happy to connect and hear your story. If you'd like to share what you've learned throughout your API journey, our blog is open for submissions. You can view our submission guidelines on our Create With Us page.

– Bill Doerrfeld

Editor in Chief, Nordic APIs
Doerrfeld.io
@DoerrfeldBill

What is API-as-a-Product?

by **Kristopher Sandoval**

API-as-a-Product is a growing concept in the software development sphere. As such, it bears some further definition and clarification. So, just what is API-as-a-Product? What are some ways we can monetize this approach? And, where is this trend heading?

An API-as-a-Product is a type of Software-as-a-Service that monetizes niche functionality, typically served over HTTP. Tech companies with API products often adopt a freemium model and utilize strategies like rate limiting to enable subscription tiers.

An API may exist alongside a primary offering or as a product in its own right. For example, if a business is developed first as a physical product or a Business-to-Business (B2B) offering, the API may be a premium extension. The API-business logic is instead the core offering for a growing number of new companies.

This relationship has resulted in a new paradigm of API thought

and design – API-as-a-Product. This model implies an API is core to the business logic, driving most of the business value. For API-as-a-Products, the API is not just the delivery mode; it is the product itself.

API-as-a-Product is roughly synonymous with an API-driven Software-as-a-Service (SaaS) offering, wherein the API powers the SaaS itself and the business logic around it. API-as-a-Product is the natural evolution of the B2B landscape, but instead of creating bespoke offerings for specific instances, API-as-a-Product essentially says, "here's what we can do — how you integrate is up to you."

Some good examples of this type of product offering are as follows:

- **Stripe**: A payment processing and eCommerce platform, Stripe offers an API to facilitate online commerce interactions. This API is not a business storefront itself — there's no "Stripe Store" where you can sell and buy goods like Amazon. Instead, Stripe enables this commerce through platform offerings designed to ease overhead.
- **Twilio**: Twilio is, at its core, a communications facilitation platform. Designed to allow agents and customers to communicate on a wide variety of platforms, Twilio is not in itself a chat platform but is instead a connection platform. This core product offers communication benefits to companies who do not want to create their own communication channels and methods but would like to integrate proven channels into their existing infrastructure.
- **Mailchimp**: Mailchimp is a marketing automation solution that offers a productized solution to unified marketing efforts across different channels. This is an excellent example of a product that "levels up" existing efforts while opening new avenues for creation — while marketing campaigns may exist in the entity utilizing Mailchimp, it nonetheless offers a

significant enough improvement in these efforts that it's best considered a distinct product.

API-as-a-Product Consumption and Monetization Models

API-as-a-Product can be monetized in various ways. We've discussed this before, but it bears repeating. Monetization is highly dependent on the specific type of API product offering in question — for example, technical API products might charge per integration or per a certain amount of data transfer, whereas commerce or payment APIs might charge as a portion of revenue or through subscription models. The type and model of monetization are variable, and as such, API developers have relatively free range to choose the best monetization model for their specific use case.

Freemium/Tiered Monetization

A prevalent model in the API space is the freemium or tiered monetization strategy. In this model, developers offer essential API functions for free with some limitations. Limitations can include time-bounded free trials, a certain number of allowed calls (akin to rate limiting), or even a completely free option for non-enterprise users subsidized by enterprise clients.

Ultimately, this model is based on the concept of accelerated cost for accelerated usage. A certain amount of balance must be employed in this model to ensure that what is provided in the free offering does not reduce demand for the premium offering while ensuring that the free option is indeed worth interacting with.

Bulk Cost Model

This model goes by several names depending on how revenue is generated from the bulk cost process. Pay as you Go is a bulk cost model in which the number of API calls made has an associated cost that is then passed onto the requesting entity. This costing can be for individual calls (usually assigned a premium cost) or bulk calls (typically discounted related to the individual call cost). For example, a bulk cost monetization model might say "each call costs 99 cents" and "1000 calls cost 500 dollars."

While this model allows the customer to control their costs in a highly granular fashion, it can also obscure the end cost if they do not pay enough attention or model their costs accurately. This can be mitigated partially through proper modeling techniques or estimates from the provider, as well as clarifications on how each call is monetized (for instance, if replays are part of the API, does this count as a new call, or a repeat of an already paid call?).

Subscription Models

A subscription model is a cost-capped version of the Pay as you Go model and typically has a time period in which the subscription is active (for example, subscriptions may be weekly, monthly, or yearly). These models often have a top-level tier for utilization (e.g., 100 dollars per week and up to 1,000 calls) and typically offer discounted costs for any overage outside the subscription level.

Unit Costing

This model utilizes discretely defined units to charge utilization. An API provider might charge per unit of use for infrastructure (e.g., number of GB utilized, number of discrete processor cycles employed, etc. — common with solutions like AWS), but can also

charge per "unit" of process (typically called an "instance"), e.g., "5 Docker instances" or "3 users of the API".

A version of this unit costing model is called the "per seat" model in which each user has a set maximum number of calls they can employ and a time-limited period of access — in essence, you are buying a seat at the table for the solution.

Revenue Sharing

In this model, API providers claim a portion of the revenue generated from the API's use. In a product sense, this is akin to something like Google's advertising systems, in which a site can integrate advertisements can claim a certain percentage of the revenue, but not the revenue in whole.

For something like a commerce app, this may come as a premium charge on top of the item's base cost or product. Integrating an eCommerce solution may mean that each product sold for 10 dollars results in 10 cents in revenue being shared with the API company, which, at scale, can add up significantly, even if there's no "direct cost" to the company implementing the API. This can actually be a significantly better implementation for many companies, as the cost paid is part of the revenue generated, which mainly eliminates "idle cost," where you are paying for a subscription that does not generate any value.

The Future of API-as-a-Product

As the web has moved to the microservice paradigm, thoughts around API development have similarly moved away from creating a single, catch-all solution for all use cases to develop extensible frameworks that allow the creation of solutions. The idea is simple — why try and solve every use case when you can create a low-level

toolset for the problem owner to make their own solution? The end result is usually more tailored to the desired outcome.

With this movement towards framework solutions, the idea of API-centric business has become ever more critical as a core business logic. APIs are created for mass consumption, and as such, the new paradigm thought pattern has shifted from "what problem **does** this solve" toward "what problems **can** this framework enable solutions for."

Of course, this movement has resulted in increased expectations for API products. APIs can no longer just provide a simple function — they require excellent developer experience, security, and reliability. For a real-world metaphor, the top car manufacturers in the world don't simply make cars that go from A to B — they make cars that do so while offering top-of-the-line features and qualities that make them more attractive. The same is true of APIs as products.

8 Unexpected Challenges of Running an API-as-a-Product

by **Bill Doerrfeld**

Many software entrepreneurs are interested in creating a working business around an API. As such, the API-as-a-Product trend has blossomed in recent years, with more and more startups beginning to embrace an API as a core SaaS business offering. This is exciting, as a self-service API could generate passive income for the provider.

However, it may not be as "passive" as you think. Sustaining an excellent API-as-a-Product requires a good deal of ongoing support. There are also many open-ended questions to answer when forming your business offering. For example, how should API calls be priced? How should we design the service for a quality developer experience? How can we evolve the API without introducing

breaking change? What challenges should we anticipate as usage scales?

For our API-as-a-Product LiveCast, we brought in Alan Glickenhouse, IBM, and Ed Freyfogle, Co-founder of OpenCage, to take a more in-depth look at the API-as-a-Product trend. In the event, we explored how to make a working business around an API and realized some unexpected realities of owning an API-as-a-Product in production.

Below, we'll review these unexpected challenges that arise with API-as-as-Product. Of course, not every API product will encounter every one of these issues, but all should be aware. So consider these tips to overcome them to create a modern, profitable, developer-friendly business.

1. It's More Than A Technical Challenge

"An API product is an API offering made available to a target market to satisfy a developer customer's needs," describes Glickenhouse. Having a standardized, machine-based delivery mechanism enables the rise of automation and overall digitalization. But creating a working, self-service API model is harder than it seems.

"It's really hard work," said Freyfogle. API providers are often naive in believing that providing an API is merely a technical challenge. "If you can get it to work, that's enough." In reality, your business challenges can significantly outweigh the technical effort.

This is, in part, due to a general knowledge gap. While there are many shared standards around API formats, design, and protocols, fewer shared business practices exist within this new economy. "There's lots of very good technical advice, there's very good discussion about newest technology, but not nearly as much about business side of things," said Freyfogle.

2. Realizing You Need an API Product Manager

Due to this condition, many API products enter the market without a product manager or a lucid product mindset. This may be because the product is a solo venture or supported by a lean team.

A traditional product manager identifies a target market, understands customer needs, and generates a package to suit those needs. Though API teams could benefit greatly from an API Product Manager, Glickenhouse finds they don't often loop one in." This is a role too many businesses try to do without," describes Glickenhouse.

Taking a product management viewpoint can help you consider the API lifecycle and estimate ongoing costs. In the process, you may discover complementary API management or gateway tools that could help host and secure the API offering, offloading potential maintenance work.

3. Developers Will Spam Your Free Trial

Ed Freyfogle has spent over five years running OpenCage, a straightforward geolocation REST API powered by open data. Throughout running his API-first SaaS business, he's noticed a weird phenomenon — "people will go through extreme lengths to *not* become customers."

Software developers want to try before they buy. Thus, having a freemium account or free trial is standard for SaaS subscription models. But, Freyfogle has found that users routinely abuse these free trials. Instead of upgrading to a paid tier, some users will rig up hundreds of free trial accounts. For the API owner, policing all

this can take a toll. “It’s frustrating, and takes a lot of time,” said Freyfogle.

4. Developers Are Resistant to Paying

Developers want to pay only for what they use. Yet, Freyfogle has noticed a stark contradiction: developers want complete pricing predictability, but they are terrible at estimating usage. Within software developer culture, people are “deeply resistant to paying for anything,” says Freyfogle.

Interestingly, in practice, Glickenhouse finds that the bulk of API business potential doesn’t even lie in direct charging, but in indirect models. “This is the real API monetization,” he describes. For example, in some affiliate models, a developer receives payment, acting as an agent for the provider. Therefore, it may behoove API providers to consider what alternative methods or partnerships they could leverage outside of direct monetization.

API providers must consider usage limits, price subscription plans accordingly, and consider how they bundle endpoint access per developer account. With all this in mind, figuring out the financial terms around an API product and proving its success in production may take some time.

5. Providing Ongoing Support is Challenging

The burden of providing support may come as a surprise. Especially for small teams, it can be challenging to provide ongoing feedback to support a functional integration. This is worsened by the fact that

many people don't read the docs. "Build it and they won't read it," joked Freyfogle.

Or, developers may approach your service who have little programming experience. "Definitely don't assume all users will be highly experienced engineers," said Freyfogle.

Global API services may also encounter language barriers. Though English is the lingua franca of software development, many users won't feel comfortable asking questions in English and will struggle to communicate with you. To help solve this issue, Freyfogle recommends eliminating jargon and anglo-centric cultural knowledge from your support materials.

6. Lowering Barriers To Adoption

Too often, developer experience isn't front and center for API products. There should be "no barriers to adopting an API product," stresses Glickenhouse. Yet, excellent self-service developer experience requires forethought and a good deal of construction.

A sleek API portal will include reference documentation, testing environments, code samples, code snippets in multiple languages, a getting started guide, an authentication guide, and SDKs for multi-platform support. By lowering the entry bar to meet the developer wherever they are, you can increase usability and adoption rates.

7. Your Real Competition May Surprise You...

The Amazon, Microsoft, and Google behemoths of the world could easily build out an API and kill your product in an instant, right? Well, in practice, Freyfolge hasn't found an issue here. It's easy for

a startup to differentiate itself. Instead, the real competitor is often the developer user, saying, "oh, I could build this myself."

Developers may question the value of your offering and assume they can build it on their own. This is especially common if your service is a layer around open-source data, as is the case with OpenCage.

"A big part of the argument is convincing them why they shouldn't build it themselves," said Freyfogle. To get through to potential users, he recommends stressing how maintenance is much harder than development. Make time and cost savings clear, and create clean developer experiences for quick, convenient use. Once developer users realize the difficulty in constructing their own API, they will more likely return to your service.

8. Build It, And They (Might Not) Come

Many fledgling API products suffer from a lack of promotion. "Don't assume you only must make [the API] available," said Glickenhouse. "You need to encourage usage, and really drive consumption, and iterate the product," he described.

API-as-a-Products must not only promote themselves but fight for attention against competing services. To stand out, there are many methods to increase the exposure for your API. For example, you could arrange a Product Hunt launch and offer perks to new users. You could profile your service in API directories and marketplaces. Or, improving developer center SEO and publishing content for non-IT visitors could help appeal to new audiences.

Final Thoughts: The Real Product is Trust

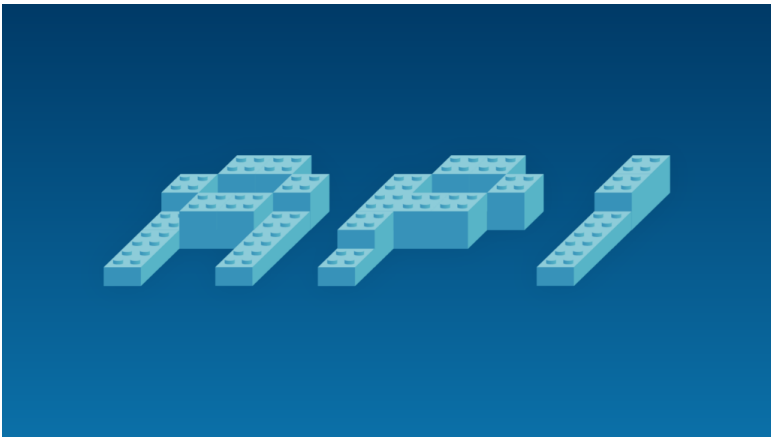
We have hardly scratched the service of what it takes to sustain a functional API product. As more companies come to rely on third-party API dependencies, there will likely be a greater need to meet SLAs and compliances. Additionally, as more and more APIs are breached, security and access management is surging in importance. Auditing your surface area for security vulnerabilities is thus vital to maintaining a stable API product.

As you can see, running a public API-as-a-Product is not as easy as it seems. It's not just about writing an OpenAPI spec and generating some docs. In practice, the bulk of your time may be spent in entirely non-technical areas to support the product.

Thinking from the developer's perspective, some common requirements include: Does it solve my problem? Can I depend on it? Is the price reasonable? With all this in mind, what you're really selling is a commitment to maintain the API. "The real product is trust," said Freyfogle.

What to Consider When Building Your API Strategy

by **Tanel Tähepõld**



In the last decade, the number of businesses that want to become more digitalized has multiplied. As part of this, the business strategy needs to include more digital strategy. The two core channels of a digital strategy have been web and mobile. The new trend is to leverage application programming interfaces (APIs) to support or enable a digital strategy.

85% of businesses consider web APIs and API-based integration fundamental to their business strategy and continued success. However, before starting with the digital strategy, we must have a well-defined business strategy and a set of goals. With an API-first development approach, we must have a valid business case before

building APIs to provide more value to your customers. A business strategy, customer success, and API strategy need to be aligned and work together to achieve the business goals.

The API-First Approach

Many companies start with building web or mobile applications. Considering today's development approaches, they also need to develop some APIs to allow web and mobile applications to consume the data. In most cases, the resulting APIs are not correctly built and tested and should not be used by third-party companies or for integration purposes.

An alternative route would be to build the API **first** and then build your web or mobile applications on top of that API. This enables us to design an API and use it for your apps to make it more real-world and developer-friendly. When building internal applications on top of APIs with developers in mind, we are laying down the foundations for others to build on. With an API-first approach, we can ensure that we are building a product of tomorrow. This process creates reusable building blocks, future-proofing the business with assets that have a more extended expiration date.

The API Economy

The notion "API Economy" describes an economy where companies make available their (usually internal) business assets or services in the form of web APIs to third parties to provide additional or new business value through the creation of new asset classes. There are several motivators for making internal assets or services available to third parties. The most common of them are:

1. Trying to reach a wider audience and make the organization's brand more visible.

2. Enabling external sources of innovation.
3. Creating new revenue sources

The API economy's value is already very well documented, and many large companies have leveraged their API well enough to generate over 50% of their revenue through APIs. Good examples are eBay, Salesforce, and Expedia, who respectively make 60%, 50%, and 90% of their income through APIs or app stores. APIs allow companies to expand into markets they may never have previously considered.

Challenges and Risks

With the drastic growth of public APIs, we see more data breaches. In fact, according to Gartner analysts, API abuse will be the largest source of data breaches by 2022. The problem is that security practices have not developed simultaneously and are often a secondary consideration for the developers shipping new applications.

Commonly, development teams work independently of their security teams, making it very complicated for the latter to effectively test or validate API security policies, leaving their organization vulnerable to an attack.

To succeed with the APIs, we need to treat them as "first-class citizens" and avoid the temptation of merely creating ad-hoc APIs only as a temporary or quick "plumbing" for web and mobile apps. We need to understand the full API lifecycle, and it needs to be part of the API strategy. A clear overview of the API design, proper documentation, and management process is a must-have when building an API strategy.

API-as-a-Product

In my opinion, APIs must be treated as full-fledged products with a designated Product Manager and API team to support them. If we want to take full advantage of APIs, then "build and forget" or "build and they will come" approaches will not work. When building APIs, we should advance step by step and enable APIs for different stakeholders and audiences in the following order: internal teams, partners and customers, then third-party developers. Let's look at how this should progress.

1. Internal Teams

The initial goal is to enable your internal teams to build new functionality and applications on top of your APIs. Even if internal teams use the APIs, we must have proper API documentation in place as we want our teams to work efficiently. Internal teams must be able to consume the APIs as a self-service product.

2. Partners and Customers

Your business partners and customers are the next stakeholders that we can provide new value via APIs. Your partner API could integrate with a customer's HR application to streamline employee information or with a CRM to improve their task management processes.

3. Third-Party Developers

The final step is to make your APIs available to the general public. If by now, you have not thought about API documentation, developer experience, or API security, then it is too late, and you are about to fail.

Focusing on the Core Business

You should always focus on your core business and leverage your strengths. I firmly believe that APIs are the best way to extend the market by allowing third parties to build specific value offerings on top of the existing core products. In my book, the best examples are Salesforce and Shopify; both developed a stable and robust core product. They then opened their platform to third-party developers to build additional value on top of their core services and offerings.

The idea is simple; Salesforce and Shopify opened their APIs to third-party developers, who now have access to hundreds of thousands of potential customers. In return, developers are building new applications for their customers, which Shopify or Salesforce wouldn't pursue because it is either not their core business or the size of the market is not big enough. At the same time, both platforms collect a small commission fee from each developer. The actual product that Shopify or Salesforce provides is not their API, per se, but access to their customer base.

Providing Seamless User Experience

APIs are great for providing server-to-server integrations with external applications. Although, if developers need to build integrations that provide a user interface, we end up with fragmented user experience and different-looking user interfaces. To unify the user experience, we would need to provide building blocks for developers to build integrations that look and feel like part of your application.

It is not a coincidence that again, we can look to Salesforce and Shopify for inspiration. Salesforce provides Lightning Design System, which includes the resources to create user interfaces consistent with the Salesforce design language and best practices. Shopify

has also provided a user interface package called Polaris that allows developers to use similar design components as available for Shopify internal teams. This will enable them to embed their application into the Shopify user interface, so the user doesn't even realize that they are using some third-party application. Providing easy-to-use building blocks lets developers concentrate on building logic rather than on pixels, experience, interactions, and flows.

On top of that, both software companies have introduced app stores where developers can promote their applications, and users can easily install them to their Salesforce or Shopify account. This approach allows Salesforce and Shopify to build their core product. At the same time, they have a competitive advantage over their competitors as there are hundreds of applications that solve niche problems and make their platforms attractive to customers.

Summary

APIs will play a significant role in building digital and business strategies in the coming years. If you want to take full advantage of APIs, you need to manage your entire API lifecycle, as the "build and forget" or "build and they will come" approaches will not work.

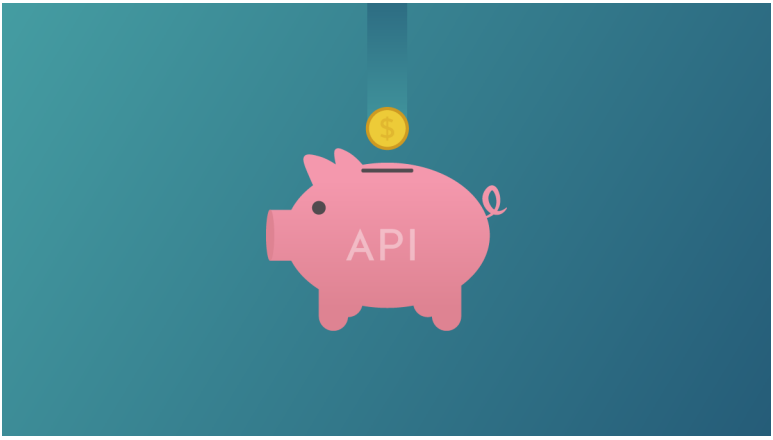
When starting with the API strategy, the first step is to map out what primary value you want to provide through your APIs and who your customers are. You also need to acknowledge that your customers also usually have multiple stakeholders: the decision-maker and the developer who will be implementing your API with their systems.

Think about the developer experience; your API must be well-documented and intuitive to speed up the integration process. Ease of use can become a decision point if your competitor provides the same value but with faster integration.

Make sure that you are building a product of tomorrow and future-proofing the business. The goal is to create solid building blocks that can be reused for decades to come.

6 API-Driven Startups Shaking Up Silicon Valley

by **J Simpson**



In mid-2020, a few eyebrows were raised when Daily.co, an API-provider that lets users quickly and easily integrate video chat into their websites, raised $4.6 million in venture capital during a round of fundraising. But this is just one of many cases of late. APIs have been changing the business world in all manner of unexpected ways in recent years.

To say that we are living in unprecedented times would be an understatement. The business world was already reeling and quaking for most of the 21st Century, in light of increasing globalization, a never-ending onslaught of new technologies, and the unexpected demise of several previously-monolithic industries.

Luckily, we've seen a rise in new industries, as entrepreneurs find new ways to leverage emerging technology into new businesses.

Considering the power and versatility of APIs, it's unsurprising that they're powering a whole new industry of API-driven products.

Let's look at some of the API-driven startups that are shaking up Silicon Valley.

1. FalconX

Blockchain technology might not be garnering as many headlines as it was a few years ago, but that doesn't mean it's lost any of its potential. In fact, blockchain has been slowly, but-steadily, integrating into the business world.

FalconX bills itself as the most advanced digital asset trading platform globally. It offers realtime assets to help people make the best cryptocurrency investments. That's not as easy as it sounds.

Cryptocurrency moves through a number of decentralized exchanges, which makes it hard to keep track of the going rate. That makes FalconX a blessing for cryptocurrency traders who've been wanting to move into realtime trading. It also addresses some of the other major concerns facing the cryptocurrency industry, like ensuring a company's ethics.

It also means that FalconX's API could be a blessing for anyone who wants to integrate realtime cryptocurrency insights into their projects. Investors seem to be excited about this, as FalconX raised $17 million during fundraising as of May 2020.

2. Flinks

Connecting apps with financial data isn't as simple or as straightforward as it might seem. Several APIs have arisen to fill this need. Flinks is a firm favorite among developers. The Flinks API

and the Connect UI lets you authenticate financial transactions by integrating realtime Digital KYC, Account Authentication, and Transaction Histories.

Flinks also supports multi-factor authentication, account selection, consent pages, or custom tagging.

To understand how Flinks works, check out some of their use cases. You can see how they handle user verification, for instance, returning `Transactions`, `AccountNumbers`, `Balance`, and other information commonly found on a voided check, reducing the amount of time it takes to reduce the time it takes to verify a transaction.

Flinks has garnered $14 million in funding, so far. Clearly, they're on to something and providing a service of real use.

3. Spruce

Real estate is big business, which means big money. Any industry that generates that kind of revenue will have a rich ecosystem of products trying to capitalize on it. That means developers need *realtime data*. Yet, many real estate assets are still handled via paperwork. Spruce is set to change that.

"Instead of using local offices with manual communication and manual processes, we provide [our clients] with API's that allow them to scale effectively and to provide great digital experiences to their customers," Spruce CEO and cofounder Patrick Burns told TechCrunch.

Spruce isn't aiming to provide real estate services on their own. Instead, they're focusing on simply providing realtime real estate data to homeowners, brokers, and lending institutions. That's how they've managed to be so effective. They're simply offering the infrastructure that empowers real estate professionals.

Spruce API comes in OpenAPI format with several endpoints specifically for real estate. They've got specific calls for `Quotes`, `Orders`, `Title Reports`, or `Settlement Statements`. All of these queries return assets in a standard JSON format. For Spruce, realtime data is one of their big selling points. Also, the API lets its platform integrate with other financial software.

Spruce has raised more than $19 million in two rounds of fundraising at the time of writing. They recently added another $29 million in backing from Scale Venture Partners. The real estate industry appears to be ripe for change.

4. Evervault

With so much money to be made with data, not to mention the ways it can be used for even more nefarious purposes, data privacy is obviously important. Evervault is an API-driven startup based out of Dublin that lets developers integrate data privacy easily and effectively.

Evervault CEO Shane Curran feels that a true data privacy solution isn't yet available. Today's data privacy laws, he argues, are bad for consumers and businesses alike. Evervault is built around the idea of 'privacy cages.' This prevents anyone but the data owner from decrypting the data. The privacy cages are modular, meaning they can be adapted easily, over time, as the security landscape shifts. Evervault's data privacy also relies on mathematical principles rather than judicial approaches like GDPR, CCPA, or ePrivacy.

Some investors see the potential in this application. Evervault raised $3.2 million in venture capital from a number of high-profile investors, like Sequoia, Kleiner Perkins, and Frontline.

5. Nylas

Email integration might not seem that exciting until you realize that nearly every app on Earth needs it. Nylas is an API that lets you integrate email, contacts, and calendars into any application.

Nylas offers a full range of communication tools via its API. These are all returned in a standard REST format, meaning you can GET, PUT, POST, and DELETE. Responses are returned in a standard JSON format.

Nylas works similarly to Twilio or Stripe. It lets developers add email connectivity with just a few code lines so they can focus on developing their apps. It essentially functions as an adapter between the app and the most popular email providers.

Adding email, contact, and calendar connectivity might seem like a simple utility, but Nylas' popularity suggests otherwise. It's already being used by high-profile clients like Comcast, Hyundai, Salesloft, and News Corp. CRM integration is one of its most powerful and popular applications. Clearly, investors see the potential in this email API. Nylas has raised over $30 million in two rounds of fundraising. This windfall paves the way for Nylas to pivot to providing B2C services, as well.

6. Segment

Businesses operating without a data strategy by this stage in the game are essentially fumbling in the dark. Big Data lets businesses know *exactly* what their customers want, leaving those without it to make educated guesses and hope for the best.

Segment is an API that offers comprehensive business intelligence business tools for gathering business data for marketing purposes, letting business owners and developers focus on building products

and developing their business. Segment collects data from all of your sales and marketing channels and your other digital assets like apps, websites, and servers. This data is collated alongside insights from CRM software, payment systems, and internal databases. This provides a powerful, customizable dashboard. It also lets you route your data from virtually anywhere to any number of popular business software tools, from Google Analytics, Mixpanel, KISSMetrics, and over 300 more.

Segment has raised over $283 million in venture capital to date. Segment was one of the earliest APIs that became a product in their own right. Perhaps it might be better to consider them an API success story and an example of what's possible when APIs meet the business world.

Final Thoughts

APIs are quickly becoming an accepted business practice. They're even becoming businesses in their own right. If data is the new oil, as the headlines claim, these API-driven startups are some of the first boomtowns. We expect to see many more as the world continues to adjust and adapt to today's data-driven climate.

5 Ways to Generate Direct Revenue With APIs

by **Kristopher Sandoval**

One of the first questions an API team has to wrestle with is how to self-sustain the service. While there are typically various free options for hobbyists, APIs at scale must sustain themselves and generate a financial return if viewed as products.

Revenue generation is a significant component of any business in the API space and entails various potential applications. These models can be boiled down to two general types — **direct** and **indirect**.

Below we'll discuss one of those types — **direct monetization**. We'll look at five different ways to monetize APIs and consider how these models are different in practice. We'll also look at some example API products for revenue ideas and identify whether these practices are as user friendly as they are effective.

Direct vs. Indirect Monetization

Before discussing specific monetization methods, let's consider the fundamental differences between API business model categories. While we will focus on direct monetization in this piece, indirect models can be equally valuable.

Conceptually, the difference comes down to the form of value generated. Direct monetization is easy to classify — direct payments, typically either in credit or cash, is the format one would expect. Indirect monetization is quite a bit different. It delivers value in non-directly-convertible cash analogs, such as brand valuation, marketing value, easier partner programs, operational streamlining, and so forth.

Direct monetization, however, results in a cash analog. Even if there's no actual money handed from one person to another, the value is directly convertible, with credit, balance transfer, or in-lieu-of-cash payments being relatively common. This piece will specifically cover direct monetization schemes.

5 Methods of Direct API Monetization

1. Direct Billing

Direct billing is — as you would guess — the most *direct* monetization method on this list. Direct billing monetizes the API calls themselves, turning the API into a sellable, meterable product. This pay-per-call model can be lucrative, but it comes with the downside of being somewhat restrictive for new users.

Without a freemium option, direct billing makes it hard for an API product to capture early success. Products need a critical mass of

users before they are "successful," and this critical mass is often the source of new growth. However, APIs are unique in that adopting any third-party API is a risk for the developer consumer. Integrations require a necessary amount of trust, which can only be built through production use.

Direct billing is perhaps the most impactful of these models. It's highly lucrative, but without a low barrier for new users, pay-per-call can stifle early adoption. For this reason, product managers should simultaneously employ other methods of direct API revenue generation.

2. Freemium Model

One way to resolve the downsides of direct billing is by adopting a freemium model. This model, which has been popular in the software world for years, is less concerned about the total conversion of user-to-value. Instead, it targets specific paid users to have those paid users subsidize the free users.

In a freemium scheme, free users are allowed access to "lite" versions of core functionality. Many freemium APIs grant open access yet include heavy rate liming and time delays. Complex functions and the lifting of restrictions, however, are reserved for paid customers.

Freemium models allow API products to build a critical mass more effectively without losing potential users. Adopters are more willing to entertain the risk of utilizing a new API if that foray has zero cost outside of their own time. The hope is they will eventually convert to a higher paid tier.

The downside with freemium models is that they must carefully balance free service provision with revenue generation. For example, offering one free endpoint from a 100-function catalog is probably not the right balance for an average consumer. Thus, striking the right balance between giving services away to entice new

users and giving away everything point-blank must be carefully implemented by the API implementer.

3. Enterprise Pricing

Eventually, API products seek not to monetize single users, but rather to monetize entire organizations. This B2B scheme is often referred to by another name — enterprise pricing.

Enterprise accounts may utilize calls that are heavily demanding of the API and require excessive resources. Such partner scenarios are necessarily more expensive than users who may only issue a single call each use period. In such a dichotomy, the most equitable and profitable monetization mode is to shift that expense to the enterprise user to subsidize the free users.

API products often include enterprise plans within tiered pricing models (Freemium, Basic, Premium, Enterprise, etc.). When it comes to user experience and monetization, tiered plans can offer the best of both worlds. Businesses have more tolerance for pricing models as they are accustomed to paying for partner IT services. And, you avoid the direct billing aversion concern with average users.

4. Ad Revenue Sharing

While the previous methods monetize the API consumer, ad revenue sharing monetizes end-user attention. Ad revenue sharing is a great monetization approach, as it monetizes use by leveraging marketing efforts to market to a broader userbase.

In this system, API providers utilize advertising networks to generate revenue. This can take a couple of forms. One of the most common is embedded ads on consumption portals, allowing developers to retain a percentage of the ad revenue.

Ad revenue sharing is fundamentally free for the API consumer. For this reason, ad revenue sharing is preferred by some developers over a freemium model, as user attention can be rationed more easily than cash. Of course, ad revenue sharing does not apply to all types of APIs. Such a model best fits eCommerce advertising, affiliate networks, or product-heavy sales channels.

5. Upsell

Within an upsell model, API use is positioned as a bonus when selling a more comprehensive platform. In this scenario, the integration typically supplements a core experience. A SaaS application may offer a basic account that grants access to a web interface — the core application. Yet, the account must upgrade to a premium tier to export data from the application with a REST API.

An upsell model differs from the idea of direct billing in that the upsell API itself is not a standalone product. Instead, the API, or perhaps a more advanced version of the API, is locked only for B2B utilization. It's part of a package. For example, a data aggregator for mobile application statistics may enable access to basic data for free and provide integration abilities at added costs.

While this seems like a freemium model by another name, the distinction is that, in freemium models, the "upsell" opens access to additional resources rather than avoiding artificial restrictions on the core API.

The danger of an API upsell is that it can often be misleading what, exactly, is allowed for a non-premium user. Open API access is almost an expectation these days, especially for paid SaaS services. Realizing API integration is locked behind a labyrinthine system of upgrades may cause frustration for developers. If adopting an upsell model, SaaS providers should do so carefully and transparently communicate these terms.

API Monetization Strategy

The way you monetize an API can have just as much impact on developer onboarding as the service's discovery, usability, and marketing initiatives. A poorly designed monetization strategy may generate significant income in the short-term, but limit a userbase before sustaining long-term growth.

On the inverse, the correct monetization approach can generate significant income while providing the financial backing to grow the underlying services and expand core offerings.

How an API provider chooses to monetize is dependent on a variety of factors, including the total cost of running an API product. Each provider should consider the above revenue options as helpful tools to craft the perfect implementation.

The Ultimate Guide to Pricing Your API

by **Lindsey Kirchoff**

Guest Post by RapidAPI with Benchmark Data on 1,800 APIs

There's big money in the API economy. But the rise of this new economy begs some key pricing questions. How do these companies actually make money from their API? What are the best practices for pricing? Are there any industry standards for API pricing or are we still in a wild west phase?

As an API hub housing over 7,500 APIs [at time of writing], we at RapidAPI found ourselves uniquely positioned to answer these questions. We've determined benchmark quotas, prices, and overage fees associated with the four most common API business use cases. The result? Everything you need to know to price your API effectively for hobbyists, small businesses, and enterprise developers. Use these benchmarks, along with the three C's of your

business (cost, competitors, and content), to help guide your API pricing.

First Things, First. How Do Businesses Monetize Their APIs?

There are three ways that companies monetize APIs: data collection, product adoption, and developer usage.

Monetization model	Definition	Example APIs
Data Collection	Collect data from third party apps to use in product design or advertising efforts.	When consumers log into a third party app with Facebook, the social media company's API learns what apps consumers use and how they use them.
Product Adoption and Customization	Allows developers to build custom integrations to increase value and make it harder to migrate to competitors.	When developers automate commands and processes with the Slack API, it becomes trickier for them to migrate to a competing chat software.

Monetization model	Definition	Example APIs
Developer Usage	Charge developers directly for API calls and requests.	The Imgur API charges developers to upload and download images from their online gallery.

The vast majority of APIs charge developers directly for calls and requests. This developer usage pricing model is what we'll be focusing on for the majority of the article.

Three Developer Usage Pricing Models

Developer usage models typically fall into three categories: fixed quotas, pay-as-you-go, or overage pricing. A *pay-as-you-go* structure means that developers pay for each individual call. A *fixed quota* model allows developers to purchase a fixed number of calls per month, but they cannot exceed the quota. Lastly, the *overage model* allows a developer a fixed number of calls, but charges a small overage fee if the developer exceeds the number of calls.

To illustrate the differences between the three models, here are some sample pricing plans for an image storage API.

Pay As You Go	Fixed Quotas	Overage Model
$0.01 per download	500 downloads / month	250 downloads / month + $0.01 per additional download

Pay As You Go	Fixed Quotas	Overage Model
$0.05 per upload	100 uploads / month	50 uploads / month + $0.05 per additional upload

Each of these three models has its pros and cons:

Pay As You Go	Fixed Quotas	Overage Model
- Pricing scales linearly with activity	- Predictable pricing	- Predictable pricing
- Pricing scales for larger apps	- Predictable revenue	- Predictable revenue
- Revenue is unpredictable	- More revenue from low volume developers	- App is never shut down
- Pricing varies monthly for developer (harder to budget)	- App shuts down after quota is reached	- Miscommunication around overages

While each model has its pros and cons, we recommend the overage model. The overage model allows the predictable pricing of the fixed quota plan, but there's also the scaling advantages of the pay-as-you-go model. Plus, with overages, a developer's app will never go down. As long as you communicate your overage model clearly to developers, we recommend this model.

Original Benchmark Data

Most APIs on RapidAPI follow the overage pricing model. We focused this original benchmark data on 1,800 public paid APIs.

These have four options: a free limited testing plan, a hobbyist developer plan, a small business plan and an enterprise plan.

We weighted the average plan price, overage fee and the quota by the number of subscribers that the API had. Here's what we found.

	Free	Hobbyist	Small Business	Enterprise
Monthly Subscription Plan Price	$0.00	$9.08	$76.00	$370.00
Overage	$0.08	$0.01	$0.007	$0.02
Quota(calls per month)	64,326	688,991	4,051,181	16,120,060

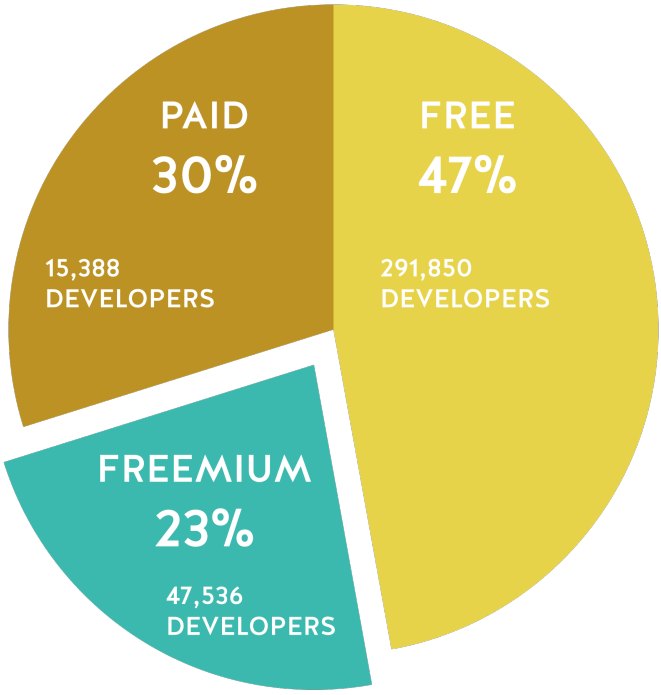
While every API is different, these benchmarks are a good place to start when setting your API prices. Based on this data, we recommend starting with the following plan prices as a rule of thumb:

- **Limited free plan**
- **Hobbyist**: $10 - $20 base plan price
- **Small Business**: Approximately $90 - $100
- **Enterprise**: $150 or more but varies widely

While this data helps by providing numerical context, it also provided a larger strategic insight.

Include a Free Testing Plan

Developers like to try before they buy. This is common advice, but for good reason. In fact, according to our data, developers are **3x more likely to subscribe to a paid API with a free tier** (aka freemium API) than an API with only paid plans.



As you can see from our chart above, even though freemium APIs make up 23% of the 1,800 APIs sampled, they have over three times the number of developer subscribers than paid APIs.

Tailor These Benchmarks for Your Business with The "Three Cs"

These numbers are a helpful place to start, but it's important to apply them in the context of your own industry. As you review your own business, look at the three "Cs" of API pricing:

- **Cost**: How much does it cost you (the provider) to fulfill a call? What are the business expenses that you incur?
- **Competitors**: How are other industry players charging? Are you in a crowded space with many alternatives or a less busy vertical?
- **Content**: What exactly are you offering with an API request? How valuable is each individual call or endpoint?

The goal with asking these questions is to put your API in context. Which "C" drives your pricing strategy? Here are some examples from our marketplace:

	Business Example	**Example APIs**
Cost-driven pricing	Shipping API	This is an industry with fewer competitors. Each shipping API provides a similar service/content. The prices are high and depend on outside vendors. We recommend pricing to account for these outside factors.

	Business Example	Example APIs
Competitor-driven pricing	SMS API	This industry is very competitive, where multiple providers provide similar offerings. The costs are fixed per call with tight margins. We recommend investing in branding, and pricing based on competitors.
Content-driven pricing	Business Data API	Each API call provides very rich business intelligence. The costs are minimal and the content is specialized information with little competition. We recommend similar pricing, but with a lower quota for this API.

The Takeaways

As you move forward with your API pricing, here are some final thoughts to keep in mind. The overage base model seems to be the most popular and offers the best flexibility over a fixed quota or pay-as-you-go model. Consider adding a limited free trial plan to increase developer adoption.

Also, when you are pricing your API, arrange it into smart pricing tiers. We've found that separating your offering into a free trial, hobbyist ($10-20), small company ($90-100), and enterprise plan ($150+) is a good way to maximize sales. Lastly, consider the business implications of your pricing model by determining which of the three "Cs" (content, competition and costs) drive your strategy when pricing your API.

Calculating the Total Cost of Running an API Product

by **Tyler Charboneau**

So, you want to develop and distribute an API. Connecting people with your services is a great way to build a user base. It's also a pathway to monetization. This success doesn't come freely. The number of monetary obligations behind running an API might surprise some developers. Expenses — both up front and ongoing — vary from one company to the next.

Our goal is to estimate *the monthly cost of running an API*. If you're a product owner or developer, we hope these ballpark figures will provide some insight into your potential operating costs. This does involve some guesstimation on our part, so take these numbers with a little grain of salt. However, the sources of these costs are

100% legitimate. We'll break down the various factors at play when developing and running your API product.

Introducing Our Use Case

For the sake of our example, we'll say our API supports 100 developer accounts, each accounting for 10,000 calls per month — one million total API calls in the same period. This engagement level will help form the basis of our other estimates. We'll calculate these costs from development to production, while including long-term maintenance. Let's first tackle what we'd consider the upfront costs associated with development.

Assessing Development Time

Probably the trickiest aspect of pricing is development, as the API design process includes numerous contributions from multiple stakeholders. You'll have your permanent team members, but some developmental stages may require outside consultation. Let's assume that we hire contractors to help conceptualize and develop the API. We can calculate potential extra costs from here:

Research

Many important considerations going into your API. Who will your users be? What data or third-party integrations should be included? These decisions will lay the groundwork for your API. They'll also influence how long your team will spend in the research stage.

Let's say we hire a contract software developer full-time, and spend three days on research. According to Glassdoor, full-time software engineers make an average of $86,774 per year. That's $7,231 monthly. However, there's more calculating to be done.

Independent contractors make approximately 50% more than their salaried counterparts:

1. $7,231 x 1.5 = $10,847 monthly (contractor)
2. $10,847 / 30 days = $362 per diem (contractor)
3. $362 x 3 days = **$1,086 (contractor cost in the research stage)**

Once you get your ideas down on paper, it's time to start building out your components. Because the database is so central to the API, we'll tackle that next.

Designing Data Structures

The database is the lifeblood of your API—the backbone of crucial information that you serve to customers and everyday users. Building this out in the most compatible, structured fashion will drive your API's success. We can follow the same principles seen above with the research stage. Design takes time, as does establishing appropriate security measures. This can take anywhere from 5-10 days. We'll average that out to 7.5 days for our calculations:

- $362 per diem (contractor) x 7.5 days = **$2,715**

Next up is prototyping, the exploratory stage of development.

Prototyping

Once your security and data structures are nailed down, it's time to throw experimental functionality together. This step precedes the development of an MVP. Prototyping verifies your API's connectivity while confirming that endpoints are functional. This can take 3-5 days:

- $362 per diem (contractor) x 4 days = **$1,448**

Making a Minimum Viable Product (MVP)

The MVP is essentially your core product, minus the bells and whistles. This stripped API typically excludes extra features that will eventually make it into production. Since this version is bare bones, it's expected to be functional before any additions come into play. This typically takes around 5 days:

- $362 per diem (contractor) x 5 days = **$1,810**

Monitoring and Galvanization

These two steps go hand in hand. You'll want to establish metrics-based oversight of your API, to assess how traffic impacts your overall ecosystem. You can use monitoring to strengthen your solution. This includes identifying bottlenecks, security holes, and more. Logging and limiting features may be baked in. We'll also have to create our alerts.

This step takes planning and experimentation—usually extending our time allocation to 5-10 days (or 7.5 days on average):

- $362 per diem (contractor) x 7.5 days = **$2,715**

Documentation

People will be using your API, so you must make sure instructions and use cases are laid out appropriately. These documents will explain the ins and outs of the API. This takes roughly three days. Technical writers typically shoulder these responsibilities. An mid-level, in-house writer costs an average of $30 per hour, adapted from Glassdoor's figures. Let's convert that rate:

1. $30 per hour x 8 hours = $240 daily

2. $240 daily x 3 days = **$720**

This is a rough estimate of course, but it gives us an idea of how costs work out during ramp up. Your initial costs make up the largest bulk of your API-related expenses, but you're not out of the woods. Now, we jump into ongoing costs.

Estimated initial cost (all development items): $10,494

Deployment and Maintenance Costs

Moving from development to deployment means publicizing your API. This includes hosting, maintenance, and other activities to help your API reach users. We'll break these items down one by one.

Hosting and Remote Database Costs

Your API acts as the middle man, connecting your users to your hosting service(s). Your API's value rests with its ability to process these requests, time after time. Good hosting providers offer maximum uptime and speed. If we *are* fielding roughly one million monthly calls, it's probably best to opt for an enterprise-grade solution (as opposed to self-hosting).

Choosing Amazon API Gateway

Luckily, Amazon API Gateway has a free tier. If our 1,000,000 monthly calls are made through a REST or HTTP API, your first 12 months will be free—as long as your calls don't exceed that threshold. This is great, but if you scale further, you'll have to pay tiered rates. If your API (knowing our monthly usage) is a WebSockets API, your upper limit is 750,000 monthly calls—thus pushing you into paid tiers. Here are those prices:

- REST API: **$3.50 monthly** per additional million (up to 333 million calls)
- HTTP API: **$1.00 to $1.17 monthly** per additional million (up to 300 millions calls, regional)
- WebSockets API: **$1.00 to $1.24 monthly** (up to one billion initial requests, regional). Plus **$0.31 per connection minute**

Note that HTTP calls are metered in 512KB increments, so any user request over that amount will incur an additional call. This may bump you to the next tier. Optimization is key to keeping costs down. In a similar vein, WebSockets requests are metered at 32KB, so any request exceeding that is processed as two (or more).

Choosing DigitalOcean for Databases

What if you choose to integrate an external database server instead of creating your own? DigitalOcean offers a multitude of plans depending on the memory you need. The company integrates fully-managed MySQL, Redis, and PostgreSQL databases—on top of a VM platform. This approach will slash (or eliminate) your in-house development time. Costs can range widely depending on memory and standby nodes:

- **$15 to $480 monthly** with zero standby nodes (memory dependent), plus **$0.022 to $0.714 hourly**
- **$50 to $1,600 monthly** with one standby node (memory dependent), plus **$0.074 to $2.381 hourly**
- **$70 to $2,240 monthly** with two standby nodes (memory dependent), plus **$0.104 to $3.333 hourly**

Maintenance Costs

Your API is finally up and running, but it will need occasional maintenance. Problems arise due to attacks, high traffic volumes, and rare server outages. These problems are often handled in-house

by DevOps engineers. These professionals make $115,666 annually, on average—or $60 hourly when rounded up.

Provided you perform 8-24 hours of monthly upkeep, that's a total cost of $480 to $1,440 in monthly maintenance (when strictly using employee salaries). Going this route, there are no added costs, like there'd be when factoring in third-party management. Note that you're not paying in-house employees an additional premium atop their salary—we're just looking at proportional allocations.

Ideally, we can automate these processes as much as possible to avoid excess (costly) human intervention. Even regular health checks and tests can reduce maintenance costs, by catching any issues before they grow in severity. Added tooling like CircleCI and Gitlab—which may be necessary—can also incur extra costs. As your application scales, that pricing may sharply increase.

On a grander scale, it's estimated that total maintenance costs are more than 50% of the complete software development lifecycle (SDLC) costs. Maintenance can be incredibly expensive, which puts the ownance on developers to create a solid pre-production API product.

Estimated monthly costs (all deployment and maintenance items): $496 to $3,684, plus database and hosting uptime premiums

Advocacy and Marketing

When you have 100 developers accounts associated with your API, you'll need to support those individuals as best you can. Helping developers get the most out of your API will keep them happy, and drive further adoption. Those resulting profits will offset your input costs. Advocates are "a bridge between the engineering team and developer community." These technical folks relay feedback they

gather—eventually driving fixes and feature development down the road.

Say you hire a few developer advocates, to really spread the word about your product. They're the voices of your user base, support arms, and public speakers. Advocates make an average of $45,152 annually. If you hire a small team of three, **that's $11,288 monthly** in staffing expenses. Will a developer advocate onyl champion your flagship API? That's unlikely. However, much of their time will be spent helping this product get off the ground and reach long-term viability.

Marketing

You'll also have to consider extra marketing costs, including staff, contractors, or outside agencies. Your API is a product, but a unique one, considering you're selling to both developers and product owners. Finding that right voice is key. Finding professionals who can develop a sound strategy is more challenging.

It may be more cost effective to hire freelancers to achieve these marketing goals. Full-time team members are costlier to hire. Your marketing effort might be a short-term blitz to introduce your API to developers—across numerous outlets. Long-term marketing approaches might not be necessary.

For the sake of budgeting, say we hire a contractor to work part-time (10 hours per week) to execute various marketing strategies. This individual works for four weeks, at a rate of $40 per hour. That's a **one-time cost of $400**, unless that individual is retained for ongoing projects.

Estimated monthly costs (advocacy and marketing): $11,688

What About Third-Party Tooling?

While our approach thus far has been decidedly in-house, there are various third-party tools available for management. These external solutions could be useful for the following:

- API management
- Testing
- Security services
- Performance metrics aggregation
- Monitoring and auditing

A number of services like Microsoft Azure API Management, IBM API Connect, and RedHat 3scale provide templates for design—plus tools to boost security and scalability. Developers can slash development times by using these "turnkey" platforms. Speaking to security, Google's Apigee platform offers plenty of authentication, SAML, and encryption options. What are the pricing breakdowns?

- Microsoft Azure API: based on our call levels, **$4.20 per million calls to ~$147.17 monthly**
- IBM API Connect: **$100 per 100,000 API calls, or $1,000+ monthly**
- RedHat 3scale Pro: **$750 monthly**

Choosing a third-party solution offers plenty of shortcuts, but those don't come freely. Depending on how your monthly calls grow (assuming you'll scale over time), these solutions can be quite expensive—both short and long term. For example, opting for an Azure self-hosted gateway will run you an approximate **$1,000.10 monthly**.

Evaluating our Totals

There's no arguing that API development can be expensive. Bigger companies may be more willing—or able—to shell out more dollars when bringing a project into the limelight. Smaller startups will find ways to save money to find a way to budget for an API product. As a recap, here are our totals:

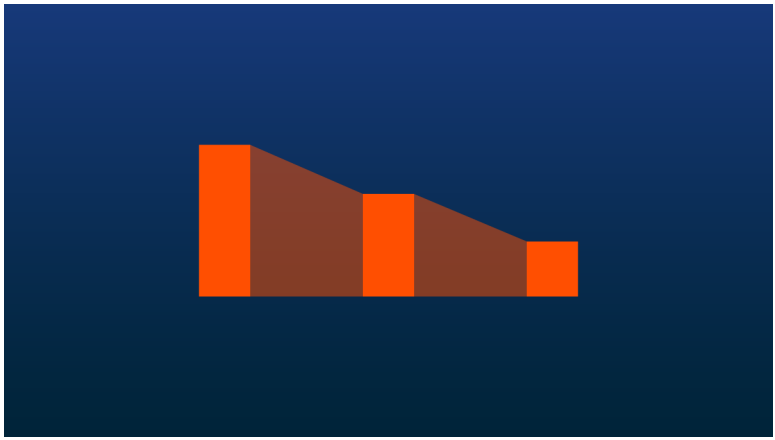
- Upfront and one-time costs: **$10,894** (development plus one-month marketing)
- Ongoing monthly costs: **$11,784 to $14,972, plus added uptime costs**. Subject to increase if third-party tooling is used

The overarching challenge with this exercise is picking suitable numbers, and including all cost-driven variables through the API lifecycle. Companies have their own budgets and functionality requirements. Simpler APIs will be cheaper, while running more-robust APIs will be expensive, both in development and upkeep. Software complexity leads to more breakages and maintenance over time, driving expenses upward. There are a plethora of factors at play when running an API product. However, your API product can pay for itself many, many times over if it's successful on the market.

If you want to estimate your own API costs, DreamFactory offers a simple, handy tool to help tabulate ongoing expenses.

13 Important Metrics for API Companies

by **Derric Gilling**



Metrics are crucial for any product. So, what are some KPIs that API companies should monitor? Below, we'll cover 13 useful infrastructure and product metrics.

When it comes to API observability and analytics, your metrics can be thought of as forming a triangle: machine infrastructure metrics for stability, software processing metrics for solving business problems, and real business metrics for managing classical business issues.

Metrics are also dependent on where you lie in the product lifecycle. A recently launched API will focus more on improving design and usage while sacrificing reliability and backward compatibility. Whereas a team supporting a well-adopted enterprise API may concentrate more on driving additional feature adoption per account

and give precedence to reliability and backward compatibility over design.

Infrastructure API Metrics

1: Uptime

While one of the most fundamental metrics, uptime is the gold standard for measuring the availability of a service. Many enterprise agreements include an SLA (Service Level Agreement), and uptime is
usually rolled up into that. Many times, you'll hear phrases like triple nines or four nines. These refer to percentage figures that measure how much uptime there is per year.

Availability %	Downtime per year
99% ("two nines")	3.65 days
99.9% ("three nines")	8.77 hours
99.99% ("four nines")	52.60 minutes
99.999% ("five nines")	5.26 minutes

Of course, going from four to five nines is far harder than going from two to three nines, which is why you won't see five-nines except with the most mission-critical (and expensive) of services.

With that said, certain services can actually have lower uptime while ensuring graceful handling of outages without impacting your service. Uptime is most commonly measured via a ping service or synthetic testing such as via Pingdom or UptimeRobot. You can configure probes to run on a fixed interval, such as every minute, to probe a specific endpoint such as `/health` or `/status`. This endpoint should have basic connectivity tests such as to any backing data stores or other services. You can easily publish these metrics on your website using tools like Statuspage.io.

More sophisticated ping services called Synthetic testing can perform more elaborate test setups such as running a specific sequence and asserting the response payload has a particular value. Keep in mind, though — synthetic testing may not be representative of real-world traffic from your customers. You can have a buggy API while maintaining high uptime.

> What is **Synthetic Monitoring**?
> As the name implies, synthetic monitoring is a predefined set of API calls that a server (usually a Monitoring service) triggers to call your service. While it doesn't reflect real-world user experiences, it is useful to see the sequence of these APIs perform as expected.

2: CPU Usage

CPU usage is one of the most classic performance metrics that can be a proxy to application responsiveness. High Server CPU usage can mean the server or virtual machine is oversubscribed and overloaded, or it can mean a performance bug in your application, such as too many spinlocks. Infrastructure engineers use CPU usage (along with its sister metric, memory percentage) for resource planning and measuring overall health. Certain types of applications, like high bandwidth proxy services and API gateways, naturally have higher CPU usage, along with workloads that involve heavy floating-point math such as video encoding and machine learning.

When debugging APIs locally, you can easily see the system and process CPU usage via Task manager on Windows (or Activity Monitor on Mac). However, you probably don't want to be SSH'ing and running the `top` command on a server. This is where various APM providers can be useful. APMs typically include an agent that you can embed in your application or on the server that captures CPU and memory usage metrics. It can also perform other application-specific monitoring like thread profiling.

When looking at CPU usage, it's essential to look at usage per virtual CPU (i.e., physical thread). Unbalanced usage can imply applications not correctly threaded or an incorrectly sized thread pool.

Many APM providers enable you to tag an application with multiple names so you can perform rollups. For example, you may want to have a breakout of each VM metric, like `_my-api-westus-vm0_`, `_my-api-westus-vm1_`, `_my-api-eastus-vm0_`, etc. while having these rolled up in a single app called `_my-api_`.

3: Memory Usage

Like CPU usage, memory usage is also a good proxy for measuring resource utilization. CPU and memory capacity are physical resources, unlike other metrics, which may be more configuration-dependent. A VM with extremely low memory usage can either be downsized or have additional services allocated to that VM to consume additional memory. On the flip side, high memory usage can be an indicator of overloaded servers.

Traditionally, big data queries, stream processing, and production databases consume much more memory than CPU. In fact, the size of memory per VM is a good indicator for how long your batch query can take as more memory available can reduce checkpointing, network synchronization, and paging to disk. When looking at memory usage, you should also look at the number of page faults and I/O ops. A common mistake is configuring an application to allocate only a small fraction of available physical memory. This can cause artificially high page virtual memory thrashing.

Application API Metrics

4: Request Per Minute (RPM)

RPM (Requests per Minute) is a performance metric often used when comparing HTTP or database servers. Usually, your end-to-end RPM will be much lower than an advertised RPM, which serves more as an upper bound for a simple "Hello World" API. This is because a server will not consider latency incurred for I/O operations to databases, 3rd party services, etc.

While some like to brag about their high RPM, an engineering team's goal should be efficiency and attempt to drive this down. Certain business functions requiring many API calls can be combined into fewer API calls to reduce this number. Common patterns like batching multiple requests in a single request can be very useful, along with ensuring you have a flexible pagination scheme.

Your RPM could also vary depending on the day of the week or even the hour of the day — especially if your consumers exhibit lower usage during nights and weekends. Some situations warrant tracking more fine-grained application metrics, such as RPS (Requests per Second) or QPS (Queries per Second).

5: Average and Max Latency

One of the most important metrics used to gauge customer experience is latency. While an increase in infrastructure-level metrics like CPU usage may not actually correspond to a drop in user-perceived responsiveness, API latency definitely will.

However, tracking latency by itself may not provide a full understanding of why an increase occurred. Thus, it's important to follow how latency is affected by API changes, such as releasing new versions, adding endpoints, or changing the API schema. This can help reveal the root cause of latency increases.

Since problematic endpoints may be hidden when looking only at aggregate latency, it's critical to look at latency breakdowns by route, geography, and other fields. For example, you may have a `POST /checkout` endpoint that's slowly been increasing in latency over time, which could be due to an ever-increasing SQL table size that's not correctly indexed. However, due to a low volume of calls to `POST /checkout`, this issue is masked by your `GET /items` endpoint, which is called far more than the checkout endpoint. Similarly, if you have a GraphQL API, you'll want to look at the average latency per GraphQL operation.

Which RESTful endpoints have higher than average latency¿'90th-percentile-by-endpoint.png" | absolute_url }})

We put latency under application/engineering even though many DevOps/Infrastructure teams will also look at latency. Usually, an infrastructure person looks at aggregate latency over a set of VMs to ensure the VMs are not overloaded, but they don't drill down into application-specific metrics like per route.

6: Errors Per Minute

Like RPM, Errors Per Minute (or error rate) is the number of API calls with a non-200 family of status codes per minute. Tracking your error rate is critical for measuring how buggy and error-prone your API is.

It's essential to understand what *type* of errors are occurring. 500 errors could imply code errors on your end, whereas many 400 errors could imply user errors from a poorly designed or documented API. This means when designing your API, it's vital to use the appropriate HTTP status code.

You can further drill down to see *where* these errors come from. Many `401 Unauthorized` errors from one specific geographic region could imply bots are attempting to hack your API.

API Product Metrics

APIs are no longer just an engineering term associated with microservices and SOA. API-as-a-product is becoming far more common, especially among B2B companies who want to one-up their competition with new partners and revenue channels. API-driven companies need to look at more than just engineering metrics like errors and latency to understand how their APIs are used (or why they are not being adopted as fast as planned). The role of ensuring the right features are built lies on the API product manager, a new role that many B2B companies are rushing to fill.

7: API Usage Growth

For many product managers, API usage (along with unique consumers) is the gold standard to measure API adoption. An API should not be just error-free but should demonstrate growth over time. Unlike requests per minute, API usage should be measured in longer intervals like days or months to understand real trends. If measuring month-over-month API growth, we recommend choosing 28-days, as it removes any bias due to weekend vs. weekday usage and differences in the number of days per month. For example, February may have only 28 days, whereas the month before has a full 31 days causing February to appear to have lower usage.

8: Unique API Consumers

Since a month's increase in API usage could be attributed to just a single customer account, it's important to measure the number of unique monthly customers. Monitoring your Monthly Active Users (MAU) can provide the overall health of new customer acquisition and growth. Many platform teams correlate API MAU to their web

MAU to get full product health. If web MAU is growing far faster than API MAU, this could imply a leaky funnel during integration or implementation of a new solution. This is especially true when the company's core product is an API; such is the case for many B2B and SaaS companies. On the other hand, API MAU can be correlated to API usage to understand where increased API usage came from (New vs. existing customers).

API tokens broken down by acquisition channel"weekly-active-tokens-by-acquisition-channel.png" | absolute_url }})

9: Top Customers by API Usage

For any company focusing on B2B, tracking the top API consumers can reveal how your API is used and where upsell opportunities exist. Many experienced product leaders know that many products exhibit power-law dynamics, with a handful of power users having a disproportionate amount of usage than everyone else. Not surprisingly, these are the same power users that generally bring your company the most revenue and organic referrals.

This means it's critical to track what your top ten customers are actually doing with your API. You can further break this down by what endpoints they are calling and how they're calling them. Do they use a specific endpoint much more than your non-power users? Maybe they found an "ah ha" moment with your API, whereas other consumers haven't.

10: API Retention

Should you spend more money on your product and engineering or put more money into growth? Retention and churn (the opposite of retention) can tell you which path to take. A product with high product retention is closer to product-market fit than a product with a churn issue.

Unlike subscription retention, product retention tracks the actual usage of a product. While the two are correlated, they are not the same. In general, product churn is a leading indicator of subscription churn since customers who don't find value in an API may be stuck with a yearly contract while not actively using the API. API retention should be higher than web retention. Whereas API retention looks at post-integrated customers, web retention will include customers who logged in but didn't necessarily integrate with the platform yet.

11: Time to First Hello World (TTFHW)

TTFHW is an important KPI for not just tracking your API product health but your overall Developer Experience (DX). Especially if your API is an open platform attracting 3rd party developers and partners, you want to ensure they can get up and running as soon as possible. TTFHW measures how long it takes from the first visit to your landing page to a first transaction through your API platform. This is a cross-functional metric tracking marketing, documentation, tutorials, to the API itself.

API tokens broken down by acquisition channel"api-adoption-funnel.png" | absolute_url }})

12: API Calls Per Business Transaction

While more equals better for many product and business metrics, it's important to keep the number of calls per business transaction as low as possible to reduce overhead. This metric directly reflects the *design* of the API. If a new customer has to make three different calls and piece the data together, the API
does not have the correct endpoints. When designing an API, it's essential to think in terms of a business transaction and what the customer is trying to achieve, rather than just features and

endpoints. A high number of calls per business transaction may also mean your API is not flexible enough when it comes to filtering and pagination.

13: SDK and Version Adoption

Many API platform teams may also maintain a bunch of SDKs and integrations. Unlike mobile, where you just have iOS and Android as the core mobile operating systems, you may have tens or even hundreds of SDKs. This can become a maintenance nightmare when rolling out new features. You may selectively roll out critical features to your most popular SDKs, whereas less critical features may be rolled out to less popular SDKs. Measuring API or SDK version is also important when it comes to deprecating certain endpoints and features. You wouldn't want to deprecate the endpoint that your highest paying customer is using without some consultation on why they are using it.

Business and Growth

Business and growth metrics are similar to product metrics but focus on revenue, adoption, and customer success. For example, instead of looking at the top ten customers by API usage, you may want to look at the top ten customers by revenue, then by their endpoint usage. For tracking business growth, it would be beneficial to use analytics tools that support enriching user profiles with customer data from your CRM or other analytics services to better understand who your API users are.

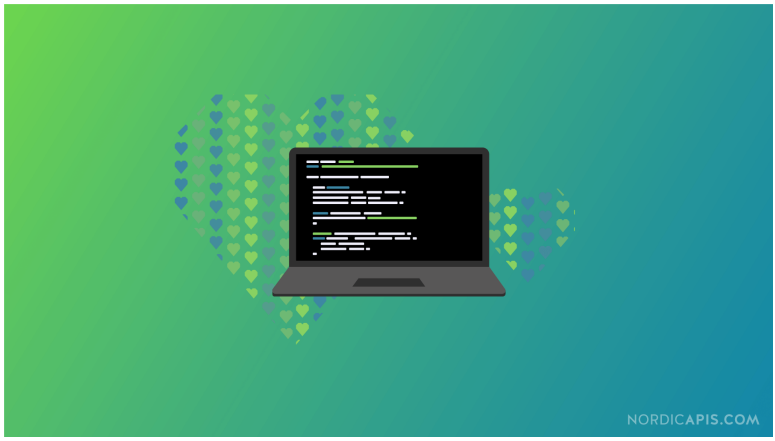
Conclusion: Track the Right API Metrics

For anyone building and working with APIs, it's critical to track the correct API metrics. Most companies would not launch a new web or mobile product without the proper engineering and product instrumentation. Similarly, you wouldn't want to launch a new API without a way to track the right API metrics.

Sometimes, KPIs for one team can blend into another team, as we saw with the API usage metrics. Also, there can be different ways of looking at the same underlying metric. However, teams should stay focused on looking at the right metrics for their team. For example, product managers shouldn't worry as much about CPU usage, just like infrastructure teams shouldn't worry about API retention.

Pointers for Building Developer-Friendly API Products

by **Thomas Bush**



"Great API products don't get built by mistake," says Rahul Dighe, the API and Platform Product Leader at PayPal. Even if you take into account all the trending principles of API ownership — *design first*, *API-as-a-Product*, *governance*, *KPIs*, and more — you might still end up with an API that's difficult to use. So, what advice would an API product strategist with over ten years of experience give?

In this post, we'll look at key pointers from Rahul Dighe's talk at the 2019 Platform Summit. These five insights should help you to create developer-friendly APIs that will stand the test of time.

1. Know Your Developers

It should come as no surprise that building developer-friendly API products start with knowing your developers. As a thought experiment, Rahul introduces three developer archetypes who might use PayPal's APIs. They include a freelancer, who is fresh out of college; a payments expert who has been in the payment space for the last ten years; and an uninterested developer, whose true interests lie elsewhere.

When you look at these archetypes, you'll see that developers all have different levels of investment and experience. Not to mention, they each have their set of own requirements, which dictates how they interact with the APIs.

As a result, Rahul suggests you consider the usage patterns of your primary developer archetypes when building new APIS. After all, APIs are a means to an end and not the end itself. Some partners might just need a single, non-API widget for their integration, while others will be looking for a full API solution to migrate to from another provider.

A particularly actionable piece of advice Rahul gives is to write a "one-pager," which contains all of the inputs and outputs and core integration patterns you hope to support with your API. This document will act as a quick reference point during the API design process, allowing you to ensure that what you're building will provide the best experience to your developer audience.

2. Be Obsessive about Naming

Rahul's next piece of advice was the inspiration for my recent article, 10+ Best Practices for Naming API Endpoints. He says you should be "obsessive" about how you name your APIs. After all, once you name an API, it stays there for a pretty long time. And

while you might be able to evolve quickly, there's no guarantee your developers will too.

If you want more specifics on how to do naming right, be sure to check out the article linked above. If you're just wondering why naming matters, the answer is simple: it might not sound like much, but the effect of redundant, inconsistent, or non-descriptive names can quickly add up, making your APIs difficult to consume.

3. Always Stick to Your Process

By this point, many big enterprises have a tried-and-tested process for building new APIs that incorporate plenty of time and due diligence. It starts with a discovery phase, where you lay out the problem you intend to solve, which is followed by a design phase, where you start to think about concrete endpoints, fields, security features, tools, specifications, and user stories. Then, you enter a development phase before finally launching your new API.

This process usually results in intuitive and all-around great APIs. However, Rahul recalls getting into a discussion with a corporate leadership team concerning why it took an entire month to add one field to an API. It's easy to be swayed by external pressures like this, but Rahul strongly believes you should stick with your guns.

As an API designer, Rahul says you're forced to build something that's needed today but will still be used five or six years in the future. You can whisk through the discovery, design, development, and deployment processes, but you'll end up with a suboptimal product. Instead, stick to the approach that works, and accept that creating or updating an API will take a little longer than others might like.

4. Build a Complete Ecosystem

While getting a single API out can be a challenge in and of itself in some business environments, Rahul says that APIs are just the start and that you can't forget about the rest of the ecosystem. In particular, he highlights the importance of SDKs, a reliable sandbox, and debug-ready support staff.

Rahul believes that these little things do matter. Sure, it takes time to put all the pieces in place, but these supporting ecosystem assets significantly contribute to the usability of your core API products. Importantly, you do genuinely have to care about and maintain these assets: if you're not careful, you'll end up with old, unhelpful documentation and a box ticked somewhere on the API-as-a-Product checklist.

5. Think API Governance

Rahul's last piece of advice is to think about how you organize and govern your suite of APIs. He draws attention to Conway's Law, which suggests that we build systems that reflect our internal organizational or communication structures. The traditional approach to combating this — i.e., building consistent APIs, despite having different teams work on them — is to use some kind of API governance system. In theory, by subjecting all API product teams to the same standards, the APIs should feel consistent.

In practice, Rahul believes that there will always be little nuances between APIs built by different teams, which adds complexity for developers who are integrating two or more of them. To combat this, Rahul suggests using something along the lines of the Inverse Conway Maneuver (explained in the article above): if you're always going to have two related API products, have a single team be

responsible for both developer interfaces. That way, the APIs are bound to be consistent.

Final Thoughts

The best API products are built when you do things properly. That starts with knowing who your target developers are and how they'll use your API. Later, it means sticking to unnegotiable, tried and tested processes and organization-wide governance standards and getting names right the first time around. Finally, there's more to an API ecosystem than just the API itself, and maintaining supporting assets like docs, SDKs, and sandboxes is crucial.

How To Treat Your API as a Product

by **Kristopher Sandoval**

Though we tend to think of APIs in a very technical way, they are increasingly combined within the context of **business** offerings and values. Accordingly, many are recontextualizing things to treat APIs more like **products**.

An API is essentially a traditional service, delivering capability to the end user without sharing the risks and costs associated with the service. Appropriately, even if the API provider does not function as a business per se, **we should treat the API as a business asset** rather than an amorphous codebase. When the API provider is in fact a business, there's only a greater reason to do so.

To help understand **why** this is valuable, and to leverage the results of such a shift in thinking for better systems and technology, let's define some business terms within the context of the API industry.

Consumer

When someone says consumer, the first thing that pops into most people's minds is a person buying a product, exchanging currency or services for that product. In the world of APIs, a consumer is simply anyone who utilizes a service, regardless of whether it is paid or not.

There are two types of consumers — **external**, and **internal**. The **external consumer** is exactly what it sounds like, a third party that exists outside of the business. This can be the end user utilizing the free, public endpoint, or another business utilizing a private, paid endpoint. An **internal consumer**, however, is someone that exists within the business unit. This might seem strange to non-business people — after all, how can a business purchase their own products?

From a business perspective, there is value behind each transaction, regardless if the transfer of value is simply in generating data, integrating with third party solutions, or driving revenue directly. Thus, by treating the API as a core business value generator integrating with both consumer types, you can leverage the revenue streams that you do have, and improve the consumer experience.

Investment

People tend to think of coding as creating something out of nothing, but the fact is that **development is not a zero cost process**. For every hour spent working on the codebase, developing new integrations, crafting new endpoints, there is a cost. This cost can be in worker hours, or in the very real cost of spinning up additional virtual or physical servers to support the API functions.

This **cost** is something that must be managed for better return on value. Running out of money when half the codebase has been created means you have nothing to offer the consumer, but

being stuck perpetually creating revenue driving endpoints while ignoring core value to the user can be just as damaging and just as useless.

Investment and funding management processes can better guide development and production to at least ensure a minimum viable product. Iterating and expanding upon this demand ensures that the product can evolve to changing business needs, as long as it's constructed around user feedback and real-world analytics.

The key here is to **ensure return on investment** not only for added value to the various stakeholders and owners of the API, but for the very real amount of increased assets and resources that can be poured directly back into development. This in turn can result in development of long-term features that facilitate long-term funding.

Roadmaps and Lifecycles

If we're going to treat the API as a product, it makes sense then to use**roadmaps** and **product lifecycles**. Defining what the minimally viable product is, and then developing roadmaps and implementing other lifecycle management tools (such as the API Model Canvas) to govern iteration upon it has several massive benefits.

First and foremost, this process prevents **feature creep**. By limiting development to a set course of known functions, values, or additions, creeping scope can be limited, making better use of limited resources and ensuring that the consumers get what they need before anything else. This prevention of creep also typically results in a leaner codebase, which is easier to iterate upon, document, and fix.

Additionally, by limiting development to that which is on the road map, you prevent extra, unused endpoints, features that are never

completed, etc. You're essentially preventing bloat, which in turns prevents feature fatigue and reduces your security threat footprint.

Additionally, implementing a lifecycle management process that mirrors processes such as those found in **ITIL** or other management guidelines can ensure that long-term value is created and limited resources are optimally managed. This will have a long-term impact on your organization, and can ensure that the API sticks around for a very long time.

Reversing the Developer-Consumer Relationship

The classic relationship between the developer and consumer might look like this — "we built this awesome thing, now how can we get people to want it?" Unfortunately, that's not always effective when it comes to APIs. An API by its very nature is typically built to do something very specific. By treating the API as a product, you're essentially reversing this approach. Instead, it becomes "people really want this feature and codebase, how do we build this awesome thing?"

Iteration and development should be based entirely on the foundational concept of doing what the **consumer needs**. Iterating on feedback and real-world use cases will always deliver better, more highly-adopted codebases, and result in a more well-supported project. The consumer in this approach comes first, and then the tech follows.

This is of course not always the case, such as with speciality APIs or APIs implementing experimental systems, but in those cases, the API isn't really being framed as a product — in other words, that is a very particular exception to what should be seen as a general, fundamental rule.

Consumer Friendly

Any business will tell you that success goes hand-in-hand with being **consumer friendly**. Unless you're the only option for a very specific use case, if your product is less user friendly, it will not be widely adopted against competition. That being said, there are certain factors that must be considered to be truly "user friendly".

- **The API must have a great developer experience**: This means the API must be easy to onboard (setting up an account and managing personal data), describe (the API must have a core competency and function that can be explained), and consumed (documentation must be adequate, and understanding systems in place). Failure to ensure the API can be consumed means that you will have an overly complex product that users will eschew in the face of easier alternatives.
- **The API must be marketed properly and evangelized within the correct context**: A secure banking API that utilizes existing systems and technologies in a better way is not "disruptive", and marketing it as such would cause many consumers who want a stable, proven system to look elsewhere. The same is true of users who want something new — if your API is branded as "same old same old" and yet marketed towards early adopters, you will see low adoption and low success.
- **The API must be unique**: The market is vast, and for every implementation, there is an alternate one vying for the top spot. Simply put, to be a viable API product, it must be unique in some part of its implementation. Whether this means the UI is designed for a particular workflow or the solution is truly novel, the API must have a unique element that makes it intrinsically valuable to the internal and external consumers.
- **The API must be secure**: No user is going to consider a solution friendly if its data is unsecured, exchanged in the

clear, and unencrypted. Secure your API, and the value will be readily apparent to the average user.

Define and Adopt Business Roles

While this will likely make some anti-business types groan, perhaps the best way to integrate this concept of "business as an API" is to adopt some **common business roles** within the product development team. An API that integrates these roles should have a few basic roles to help guide form, function, development, and marketing.

- An API Product Manager can help guide the API Product Manager can help guide the API to adhere to the business guidelines and roadmaps as set forth.
- API Sales Managers can help define the evangelizing and marketing that must be done to support the external user and drive discoverability.
- API Process Managers and Process Owners can help guide the functions of the API itself as well as the resources they govern.

While there are many roles, like evangelist, advocate, etc, these three are a very good start, and adopting them can help reframe an API into a product.

Assume Your API Will Become an Public API

While many API developers develop within their current and given business status, this can be harmful. Assuming that your API

will always remain private, or always remain limited to business interactions, only limits the potential for growth and new revenue streams.

By assuming from the start that eventually everyone will be using your API, you can start developing systems in such a way as to support this possibility.

Simply put, assume that at the end of the day, you're not going to be there to walk the future public user through your API, regardless of whether you intend to ever make it an open API. Develop in such a way that this pivoting can be done easily, with minimal cost, and with efficient application – at the very least, you are enabling success, rather than preventing it.

Final Thoughts

To fully embrace an "API as a product" mentality, assume your API will become public. This will result in better defined endpoints, more efficient routing, better systems of scalability, and interfaces that are well-documented and defined, even if they may never open to the public.

Not every API is going to fit into the traditional idea of what a "business" is – some APIs are just for internal use or for very limited, custom audiences. In many cases, there may not even be a direct revenue source. Nonetheless, every single API has its own consumers, and as such, treating the API as it if were a vital business system can have huge benefits to your users.

Assume regardless of the final product that your API will someday be a public product. At best, you will be thinking far ahead, and at worst, you will simply engage in more efficient, valuable, and impactful development to create a better product.

Why Your API Needs a Dedicated Developer Experience Team

by **James Messinger**

In the 2019 State of API report, surprisingly, only 37% of API providers viewed documentation as a top priority. When API consumers were asked to vote on the most important characteristic of an API, 60% earmarked "ease-of-use" as their primary desire when integrating, with documentation trailing in 3rd place. While documentation can contribute to overall ease of use, these numbers reveal that it is not the only element that plays into a good developer experience.

So, the question is: How can API companies improve their overall process and deliver the high-quality experience their users want? One of the best answers to that question is: Focus on creating a

dedicated developer experience team that can empower your users by making it easier to understand, easier to build, and easier to integrate (particularly if your company develops customer-facing APIs).

Understanding the Difference: DevRel and DX

What exactly is Developer Experience (DX)? DX is all about understanding developers, their needs, their abilities, their values, what they're trying to accomplish, what tools and technologies they're using, the integration points, and how they feel while using a product.

Developer relations (DevRel) is a vital component of a comprehensive DX strategy. Some companies are large enough to have a dedicated DevRel team, perhaps even with multiple distinct roles, including evangelists, advocates, and sometimes even tech writers and growth hackers. These roles are all aimed at inspiring positive relationships with developer users, through sharing knowledge that fills the gap between the creators and consumers of tech.

Whether your company has a dedicated DevRel team or a DX team that includes DevRel responsibilities, it would be remiss not to acknowledge the role developer relations plays under the larger umbrella of developer experience.

Why Developer Experience?

Software companies and SaaS providers that sell user interface (UI) products have recognized the importance of good user experience (UX) for decades. A great UX can be the key differentiator that

makes your product successful. It's how Apple won the cell phone market and how Nest made thermostats sexy.

DX is to APIs as UX is to UIs. APIs are products, and developers use those products. Those developers have come to expect a high level of quality, ease-of-use, onboarding, and support thanks to companies like Stripe, Nexmo, and HelloSign, who are continually raising the bar.

> "DX is the acquisition of knowledge needed to implement an API. Make the acquisition easier; knowledge more digestible; the journey of implementing it simpler; lives of developers better" — Anthony Tran, creator of the Luna design system

Why the Shift?

So why are companies suddenly starting to realize that they need a DX team? After all, they've scraped by without DX for decades. What changed?

In the early 90s and into the 21st century, businesses typically invested millions of dollars in on-premise software packages such as CRMs, ERPs, and databases. They then relied on an army of expensive contractors to customize these products to meet their needs. Integrating in decades past was a big undertaking — in terms of skill, labor, and capital.

However, as companies have moved into the cloud, they've shifted away from monolithic software platforms and toward smaller, micropayment-based SaaS products. This propagation of SaaS products has led to the need for standardized integrations between them, which in turn has fueled the rise of the API economy. By 2011, REST was an industry standard, and in just under a decade, we've seen nearly a 1,000% increase in the number of APIs on the market.

This Cambrian explosion has virtually eliminated the need for expensive consultants who understand the intricacies of million-dollar software packages. It's now possible for any developer to integrate with these APIs to link disparate systems and automate their companies' workflows.

The API economy has created a culture of expectation for APIs. It's now assumed that an API will be readily available, and in many cases unmetered, for consumers or developers looking to "connect" to your application. In fact, for many customers, your API is more important than your UI. Whether you have an API and how easy that API is to use may be the differentiators that make the customer choose your product over your competitor's.

So, how do you ensure they get the best, most well-rounded API?

You Need a Dedicated Developer Experience Team

Developers aren't the best at incorporating UI into their designs. That's why many companies employ UI specialists who are responsible for putting in a friendly interface on top of the components a dev team has built.

The same holds for APIs. Your dev team shouldn't be solely responsible for an API's developer experience, because that's not the dev team's specialty.

During ShipEngine's early days, one of the defining moments for our development team was recognizing that to increase adoption we had to provide more focus on creating a product that developers loved enough to justify building out a new integration. We weren't the first shipping API on the market, and we won't be the last, so we knew we needed a way to stand out.

We started to look at how API companies in other industries navigated around their competition.

Take popular payment processing platforms like PayPal and Stripe, for example. Many may recognize PayPal as one of the leaders in the industry — and, as one of the first on the market, they deserve a seat at the table. But, historically, their API has been clunky and awkward to use. When Stripe was first introduced, they knew they were offering a product that developers would love, but also knew gaining a loyal following would require a lot of legwork.

How were they able to do it?

By building an API with a good DX.

They designed a killer API with an emphasis on consistency and quality standards, wrote user-friendly documentation, provided useful code samples and powerful SDK libraries, wireframed their website to prioritize developers' needs, and they employed a great developer relations team that attended conferences and wrote knowledge-based articles. They hacked their way into a tight market by creating a product that developers loved, and experience they would want to share with others.

> "Happy developers are chatty developers, and when we talk to each other to recommend products, the ones with the best DX are at the top of the list." — Sam Jarman, DX speaker and writer

Stripe capitalized on their ease-of-use, knowing they could lean on developers to sell their product for them so long as they could show them how pleasant the integration experience could be. Their success was even enough to make PayPal jealous.

So, what's the takeaway?

Stripe is not the only company to quickly take over market share through improved developer experience. So how were they able to

corner the market in such a short amount of time? I believe it was by employing a diverse multi-disciplinary team dedicated to the four primary elements of developer experience.

4 Main Responsibilities of a Developer Experience Team

At ShipEngine, our Developer Experience team exists to ensure an exceptional experience for the developers and customers using our API products. By focusing on these four primary areas of responsibility, we've been able to design better features, champion the interests of developers, translate feedback, and advise other internal departments on how to better empathize with and design for developers.

The four primary areas are:

1. API Design

While product development is largely left up to engineers and product teams, a Developer Experience team should maintain responsibility for providing the guidance and standards engineers need to create a product that is received well by others. This includes every part of its interface, including the protocol, style, naming, models, operations, authentication, status codes, headers, errors, paging, sorting, querying, and more. It may also include some aspects of the behavior and implementation details of the API as well. Types of Deliverables:

- Design guidance
- Design review
- Style guides
- Specifications / definitions

2. Quality Assurance

With APIs, you always want to aim for quality through consistency! To ensure an API product and developer tooling meet a high standard of quality, the DX team must become responsible for employing automated tests, linters, and processes that verify compliance with designs, schemas, and style guides. Quality assurance also involves the propagation of a culture of quality throughout the design, product, and engineering teams. Types of Deliverables:

- Contract testing
- Specification testing
- Style guide compliance testing
- Verifying accuracy and clarity of docs and tooling

3. Developer Tooling

You want to give developers a chance to test out your API before investing in full integration. So, providing a robust library of developer tools is a great way to show not tell them how great you are. Developers want and need schemas, code samples, reference implementations, SDKs, and a variety of other resources to help guide them through the build-out. Types of Deliverables:

- Code samples
- Demos / reference implementations
- SDKs and libraries
- Specifications, definitions, and schemas
- Internal tooling and automation
- Integrations with developer tools and services

4. Developer Relations

And finally, the role we (and users) are most familiar with. All communications and interactions with developer customers, such as documentation, training, release notes, community events, and user feedback studies fall underneath Developer Relations. Deliverables:

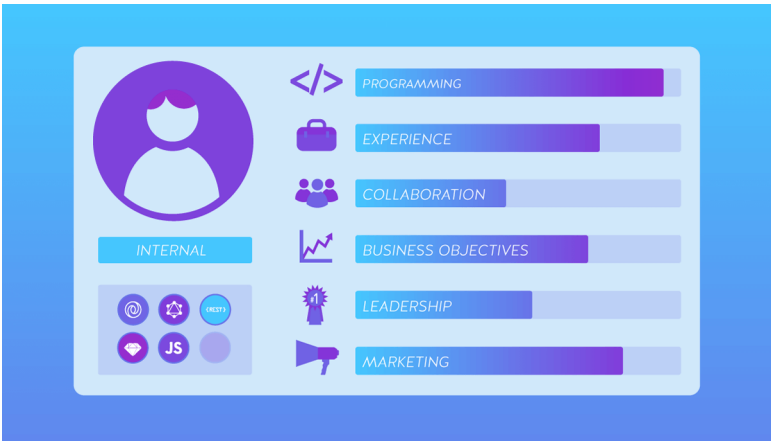
- Documentation / Tutorials / FAQs
- Release notes / changelogs
- System status info (downtime, bugs, performance)
- Media content (blogs, videos, etc.)
- Training and materials
- User research studies
- Events/community engagement (meetups, hackathons, conferences, etc.)

Final Thoughts

Just as UI/UX has been a key differentiator for Graphical User Interface products, Developer Experience is a key differentiator for API products. And, a good DX strategy extends beyond the roles and responsibilities of DevRel.

What Qualities Make a Great API Product Owner?

by **Kristopher Sandoval**



The role of a web API **product owner** is still pretty nebulous, largely due to the fact that it's a relatively new position. That's interesting, because anyone keen on **API business development** knows just how valuable such a position is. A great API product owner can be hugely beneficial, and can leverage the product's strengths to greater heights.

In this piece, we're going to discuss **what qualities make a great API product owner**, and what these qualities mean to the development process as a whole. Once you've finished this piece, you should have a solid understanding of these qualities, and a basic rubric upon which they can be compared and contrasted with your

candidate of choice.

Why API Product Ownership is Important

API product ownership is an important facet of modern web API team organizational structures. Despite this importance, the relative newness of the role is such that, even when the value is recognized, what makes a good placement is not as obvious. Having someone who can both manage a **team** and a **product** is powerful — having someone who can do that while also **evangelizing** your project internally and externally is exponentially more so.

There's a difference between "getting something done" and "owning a project," however, and it is this distinction that makes a position placement either a big win or a massive failure. Ownership of a project and all the elements of that project have a sort of cascading effect that can be more powerful than the sum of its parts, boosting productivity, improving organizational well-being, and creating a culture of personal accountability.

But what specifically does an API product owner position even entail? And more importantly, what elements are to be expected of a person in such a position?

Developer vs. Product Manager vs. Evangelist — When Worlds Collide

The classic paradigm of digital product creation is a battle between developer and manager. Developers try to create something they believe in that is functional and high quality. Product managers, on the other hand, are trying to meet business goals, trying to ascertain

the true value of the product that others would be willing to invest in it. Evangelists occupy an entirely different, unique space. They come from a technical background but are often more concerned with generating awareness, both internal and external.

This outlook is old-fashioned, but it worked in past eras of development. The new developmental situation, however, has rejected traditional roles in many cases, with small studios and groups having more tools and better funding than ever before to create unique, personal, and often niche solutions.

In this new paradigm, the old-fashioned approach simply does not always work. The new development ecosystem often entails product managers as developers, evangelists as developers, and so forth — and as those lines have blurred, so too have the positions that occupied those spaces. With developers having to wear so many hats by the very nature of the development cycle, API product ownership — as a concept and as a "position" within a company — can entail many things.

With that in mind, let's look at some of those important attributes that define this role.

True Leadership

> "Focus on constant iteration of your product or service. Never hold too closely to your idea but be open to change and innovation." - Candace Carpenter

An API product owner is not just selling an API, but everything behind it. For developer consumers, value is the sum of the support offered, the experience of the developer team, the developer support, the documentation, and other developer portal materials. All of this extra value, however, is directly powered by the leadership of that team — documentation is worthless, for instance, if leadership does not allow for it to be vetted and reviewed.

Accordingly, the value of a product is magnified by the quality of **leadership** in the API product ownership space. Product owners must be leaders, and a strong ones at that. They need to take the initiative to develop new projects, be willing to take risks for new solutions, and do all of this while maintaining **value** for the company and to the user. In short, the old adage rings true — *head in the clouds, feet on the ground.*

Part of this also means, of course, that the product owner needs to be accountable for their decisions, and able to rationalize these choices in product and project movements to not only the company and the team, but to the user base as well. **Clear communication** plays a huge role in this, and effective documentation is a big part of this approach.

Proficiency with Language vs. Familiarity with Solution

The balance between proficiency in a given language and familiarity with the implementation type is a delicate one, and one often overlooked. Let's imagine you are building an API in Go. Now, is your choice of product manager driven more by someone who is a master of Go, or by one who has heavy experience with distributed microservices, but not necessarily experienced in Go?

This is the balancing act that needs to be considered carefully. It might be tempting to look at what your product is built in, and then hire based on language, but this might result in a candidate who is an expert in the language but can barely understand the distributed structure of your microservice family.

Understanding the specific **design requirements** of an API is a must here, but always consider the attached attributes. If you are designing RESTful applications, you need to find someone versed in JSON. If working with SOAP, then XML is definitely an added

benefit. Knowledge of open-source API standards and solutions is important if that is your final intent.

These considerations are vital, and if properly considered, allow for movement if your API ever needs to pivot and change. Having the ability to adjust to the ever-changing tides of web development makes a powerful API product owner even more so.

Marketing vs. Development

> "Brand is just a perception, and perception will match reality over time. Sometimes it will be ahead, other times it will be behind. But brand is simply a collective impression some have about a product." - Elon Musk, CEO of SpaceX and Tesla

A chief consideration in the API product ownership mindset is whether or not the product owner should be positioned more in terms of marketing or in terms of development. While this somewhat ties into our earlier comparison of development, management, and evangelizing, this is much more a mindset consideration than a job duty one — for reasons that will soon become apparent.

Bill Gates innovated the workplace computer, Steve Jobs innovated the way we consume mobile media, Elon Musk innovated the way we process online payments (not to mention space exploration, and renewable-energy driven vehicles). When people talk about innovators, those names are often attached to the product. The point is that, in the tech space, the product owner is as much a brand as what they are trying to sell.

Accordingly, there's no such thing as "just" a developer or "just" a marketer in the modern workspace. With the modern ecosystem of development and innovation, the product owner should be just as recognizable as the product itself.

Accordingly, the product owner should have qualities that reflect and enable this. Being able to **bask in the spotlight**, while redirecting that spotlight to the product, is immensely valuable. Framing the solution with specific business critical **use cases** is likewise valuable. Being able to **oversee development**, stand up and say what it specifically is and does, and why it's needed — that makes a good API product owner.

Team Management

> "I have learned that nothing is certain except for the need to have strong risk management, a lot of cash, the willingness to invest even when the future is unclear, and great people." - Jeff Immelt, CEO of GE

Team management is a huge part of any organization, but within the API-fueled web economy, this is even more important. With so many remote workers, sometimes located half a world apart, the ability to inspire, lead, and most importantly **organize** defines successful leaders, and thus successful organizations.

Likewise, being able to **identify weaknesses** within the organization allows for culling and for making leaner work groups. Restructuring teams, being able to managerially outline working group responsibilities, and delegating the right tasks to the right people are all huge elements that must be done correctly in order to leverage any team's strengths.

No manager can function by themselves. Behind every Jobs, there is a Wozniak — and in the modern space, it's more often a requirement that an effective API product owner be both.

In terms of skill set considerations, knowledge and experience with toolsets that enable team collaboration in this space are key. Knowledge of messaging solutions like Slack can be hugely beneficial, and being experienced with task distribution systems like Trello equally

so. Even adopting online collaboration IDEs like Squad can mean the difference between team failure or team success.

Experience

> "There are pros and cons of experience. A con is that you can't look at the business with a fresh pair of eyes and as objectively as if you were a new CEO. Fire yourself on a Friday night and come in on Monday morning as if a search firm put you there as a turn-around leader. Can you be objective and make the bold change?" - Andrea Jung, CEO of Avon

Experience is a yet another nebulous point to consider in balance with other qualities. While it's tempting to think that experience means success, this is somewhat of a misattribution — experience can also just mean **time spent**, and sometimes, you may want a greenhorn rather than an expert.

This is very much a balancing game. Too much experience can lead to bias for a given solution or choice, and can result in rote implementations that just only slightly change from version to version. Having too much experience can also lead to complacency, and in some personality types, arrogance, both of which allow for consistent failures to go unnoticed.

Likewise, having too little experience can have damaging effects as well. Innovative approaches from a newcomer may not be grounded in reality, and can lead to some expensive failures. Additionally, while a greenhorn may be more creatively oriented, newer developers or managers can also lack crucial experience, making learned lessons come at a price.

What you really want here is a mixture. A good product manager can be highly experienced, but they must have the mind of a newbie,

willing to take risks and do things outside the box. They must be willing to think big, and most of all, be willing to fail.

Meeting Business Objectives

This is not a nebulous concept, and in fact, might be the most concrete of all advice in this piece. Your API product owner must **own the product**, and thereby, be responsible for and cognizant of the business objectives that have been given to them.

A product owner has a unique position in that they are straddling the world of development and business. While they can see the intimate details of both sides, they must be able to see the larger picture, and where the product fits in that picture. They must be able to guide the project in such a way as to promote the attainment of business objectives, balanced of course by all the other attributes on this list.

Seeks to Understand Developer Audience

Finally, and perhaps most importantly, a product owner must **intimately understand their audience**. A product is worthless without a user base, use case, and no desire for implementation — thus, an API product owner must be familiar with all of them.

As a point, here, the API owner must cater the API, its marketing, and its approach to promote both the **best developer and user experiences possible**. This includes everything from beautiful documentation to basic outreach, from error codes to code distribution — everything should be designed to help build a community.

Simply put, the API product owner must act as the **unifying node** between several diverse points within the community, addressing analytics, the development team processes, marketing and sales, and user experience, with the end goal being at any cost to satisfy the customer base and achieve the goals they desire.

Hiring Internally vs. Externally

While it might seem easy to consider a candidate based on the concepts introduced here, there's one variable that's not quite as easy to answer — the question between hiring **internally** or **externally**.

On the one hand, there's a definite value to hiring from **internal staff**. Internal staff are already intimately familiar with the solution and the language — this saves development time, and definitely helps when it comes to documentation and other practices. On the other hand, this also leads to the issues we talked about earlier when it comes to experience bias — and in this case, new blood can be just as powerful as old.

Outsourcing to Jump-Start

One remaining topic is the prospect of **outsourcing** the position as a temporary method to jump-start API development or marketing. This is becoming a much more common tactic as years pass, and has been used to great effect for API development. This strategy can allow for rapid initial development, getting the project off the ground. That being said, there are some issues.

Chief of those issues is outsourcing ignores the **value of veterans**. Having a long-term, proven manager is very important, and cannot be ignored for the sake of a quick fix. While having too much experience can be a problem as previously stated, when it comes to getting a program off the ground, having a proven manager can help give predictable insights into the response you can expect from the community.

With all of this in mind, our advice would be this — jump-starting through outsourcing is a valid approach, but it should be used sparingly. When used, whenever possible, an agreement should

come with the caveat that the manager, having proved themselves through successful iteration and launch, becomes a permanent fixture on the team.

Conclusion

We hope we've helped illuminate all of the various qualities that make a **great API product owner** — or at the very least, given a strong rubric with which to judge possible candidates. While there's no possible way to cover every single possible skill, caveat, and quality, those we've discussed in this piece should serve as a solid basis to start.

6 Ways to Market Your Niche API

by **Thomas Bush**

So you've just finished building the perfect API: it's well-designed and solves particular problems that everyone is having — what now? You could just tell a few colleagues about it and let word-of-mouth do the rest for you, but if you want to grow your user base fast, you'll have to get your hands dirty with some marketing.

Thankfully, there are a lot of options for marketing APIs; what's more none of them are particularly complicated, and they definitely don't feel "sleazy" like the widespread marketing and sales stigma might have you believe. Here are six of our favorite ideas for how you can market a niche API, complete with actionable pointers and a few case studies…

1. List Your API on Directories

One of the most effective, yet straightforward ways you can promote an API is by listing it on online API directories. The majority of directories have some kind of search function, which allows users to find your API by name (if you chose a descriptive one), as well as incredibly useful categorization and keyword features. Listing your API on these directories is as easy as pie, and since every directory is a little bit different, we won't go through process here. If you're looking for some suggestions on which directory to start with, take a look at ProgrammableWeb, RapidAPI, and APIs.io.

Here are some pointers on how to make the most of these directories…

DOs:

* Make use of all the major directories
* Look for smaller, more specific directories in your niche
* Provide as much information about your API as possible

DON'Ts:

* Choose irrelevant keywords and categories
* Forget to update listings and let information grow stale

2. Create Valuable Content

Content marketing is a fantastic way to market any product — and the humble API is no exception. Creating helpful (or even entertaining) content that appeals to prospective users is a surefire way to have the right people stumble upon your API. Truth be told, you can create whatever type of content you think is appropriate, but text (e.g. blog posts or free ebooks) and video (e.g. walkthroughs

or challenges) are popular for good reason. On that same note, you probably want to focus on creating informative content, unless you think the entertaining stuff will really hit a note with potential users.

There's also the tangential approach of creating a mashup: use your API to build an awesome service that demonstrates the functionality of your API. This always gets people talking – and will help programmers both run into and fall in love with your API. As an example, nViso used their facial analysis API to build a web app that offers financial advice based on your facial reaction to certain questions. Time for some hints on creating great content…

DOs:

* Explore different mediums of content creation
* Market your content (take SEO into account at all times)
* Enjoy the process so consumers will too

DON'Ts:

Make content that isn't relevant to your product or brand Create subpar content just for the sake of it

Case Study: Ipinfo.io

ipinfo.io provides an API for looking up IP addresses. Their content strategy involves long, well-developed bimonthly blog posts which cater to existing users, potential users, and the programming ecosphere as a whole. These articles ultimately market themselves, giving ipinfo a constant influx of curious visitors who want to learn more. Many niche APIs use a content strategy to spread awareness.

3. Make the Most of Social Media

Social media is all the craze right now, or at least it was five years ago. However, it's still a powerful tool for creating relationships with prospective users — and is just as effective for maintaining them. Ultimately, what you share on social media is "content", so you can decide whether you'll inform (especially by sharing pertinent news) or entertain along your journey of customer acquisition.

In terms of traditional social media, Facebook and Twitter are the most universal options, but you should definitely explore the possibility of contributing to Q&A sites (although it's a stretch to call that social media) — like Quora or Stack Overflow — where you can create organic product interest just by helping out. For social media, here's what you should and shouldn't do...

DOs:

* Keep your followers updated on a regular basis
* Share anything you find fun, exciting, or helpful
* Answer any relevant questions you can

DON'Ts:

* Blatantly shill your own API
* Spam your followers

Case Study: Twilio

Twilio's API products are all about cloud communication. They take a communicative approach to marketing, too, with full time developer evangelist Phil Nash answering Stack Overflow questions on a weekly basis. Most of the answers help out existing Twilio users, creating a powerful and trusted social presence for the

company. Forum participation is one way to passively evangelize a service.

4. Host and Attend Events

One of the more expensive, but definitely more conversant ways to market a niche API is by hosting, as well as attending, relevant events. In-person events are quite frankly unmatched when it comes to building meaningful relationships with your user base; the trade-off is not being able to attend them from behind the computer screen. Attending events in your niche is a much more affordable approach, but if none of them seem to riff with your product then you might have to host one. Examples of such events include conventions, conferences, and hackathons. This is how to make the most of these events…

DOs:

* Contribute to the space in a natural way
* Make as many connections as possible
* If hosting, offer incentives (e.g. prizes or just fun) for attending

DONT's:

* Blow your marketing budget out of the water by mistake
* Assume that every attendee will become a user

Case Study: Shopify

Shopify has to be the world's fastest growing eCommerce company, with numerous APIs that developers can use to work with their platform. Since 2016, Shopify has hosted "Unite", their annual conference for both partners and developers, where they announce

new releases and give talks. Not only does this improve brand awareness, but it also skyrockets credibility. Shopify arranges their own developer conference to encourage community formation.

5. Use Launch Announcements

It's the most obvious way to tell people about your new API: use launch announcements to countdown to release day! While you could keep this internal and announce your release only to an existing user base, it's a good idea to let the public know too. You can use ProductHunt for public launch announcements, but don't be scared to hunt around and find thought leaders in your space who might give your upcoming API a shoutout. If you have a blog or email newsletter, you should include a notice there too. Here are some suggestions…

DOs:

* Publish more than one launch announcements (just in case anyone misses the first)
* Tell users what to expect

DON'Ts:

- Spam your launch on every page you can find

6. Reach Out Directly

The final, and perhaps the least creative way to market your API is with direct marketing. If you already have an email list or a social media following, this might mean announcing the release of your API (just as above) or reminding people of it, but otherwise

you could still consider targeted advertising as an option. This method could cost as little as nothing, but is capable of killing your budget if you go too heavy on the advertising suite. That's why it's essential you consider what audience you'll be reaching out to directly. Reaching out is easier with these pointers…

DOs:

* Show genuine enthusiasm for your product (hopefully you've created an awesome API)
* Offer incentives for those who take action

DON'Ts:

* Devalue your brand with "spaminess" or "pushiness"

Don't Forget to Make Devs Happy!

While that's it for our "proper" marketing methods, it goes without saying that your API should work great, and just as importantly: your developers should love using it. Creating a fantastic developer experience will augment these marketing methods by encouraging programmers to use and talk about your API. On the other hand, a poor developer experience will hurt user acquisition — all those marketing efforts will be in vain if users click away as soon as they see your developer portal. To wrap things up, here are some of the essentials for a solid developer experience:

* A clean and intuitive developer portal
* Plenty of code samples and a sandbox area (for easy testing)
* SDKs and reference docs

Nordic APIs Resources

Related API-as-a-Product Sessions

Visit our Youtube Channel for many presentations on API business.

More eBooks by Nordic APIs:

Visit our eBook page to download all the following eBooks for free!

How to Successfully Market an API: The bible for project managers, technical evangelists, or marketing aficionados in the process of promoting an API program.

Identity and APIs: Discover the techniques to secure platform access and delegate access throughout a mature API ecosystem.

API Strategy for Open Banking: Banking infrastructure is decomposing into reusable, API-first components. Discover the API side of open banking, with best practices and case studies from some of the worldâ€™s leading open banking initiatives.

GraphQL or Bust: Everything GraphQL! Explore the benefits of GraphQL, differences between it and REST, nuanced security concerns, extending GraphQL with additional tooling, GraphQL-specific consoles, and more.

The API Economy: APIs have given birth to a variety of unprecedented services. Learn how to get the most out of this new economy.

API Driven-DevOps: One important development in recent years has been the emergence of DevOps, a discipline at the crossroads between application development and system administration.

Securing the API Stronghold: The most comprehensive freely available deep dive into the core tenants of modern web API security, identity control, and access management.

Developing The API Mindset: Distinguishes Public, Private, and Partner API business strategies with use cases from Nordic APIs events.

Create With Us

At Nordic APIs, we are striving to inspire API practitioners with thought-provoking content. By sharing compelling stories, we aim to show that everyone can benefit from using APIs.

Write: Our blog is open for submissions from the community. If you have an API story to share, please read our guidelines and pitch a topic here.

Speak: If you would like to speak at a future Nordic APIs event, please visit our call for speakers page.

About Nordic APIs

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.

Facebook | Twitter | Linkedin | YouTube

Blog | Home | Newsletter