SPONSORED BY

IBM

GEEK GUIDE

An API Marketplace Primer

for Mobile, Web and IoT

LINUX JOURNAL

Table of Contents

TED SCHMIDT is the Senior Project Manager and Product Owner of Digital Products for a consumer products development company. Ted has worked in Project and Product Management since before the agile movement began in 2001. He has managed project and product delivery for consumer goods, medical devices, electronics and telecommunication manufacturers for more than 20 years. When he is not immersed in product development, Ted writes novels and runs a small graphic design practice at http://floatingOrange.com. Ted has spoken at PMI conferences, and he blogs at http://floatingOrangeDesign.Tumblr.com.

IBM Cloud
Route 100
Somers, NY 10589

Produced in the United States of America

July 2016

About the Sponsor

IBM API Connect

IBM is a globally integrated technology and consulting company headquartered in Armonk, New York. With operations in more than 170 countries, IBM Attracts and retains some of the world's most talented people to help solve problems and provide an edge for businesses, governments and non-profits.

Innovation is at the core of IBM's strategy. The API economy is the source of much of today's innovation as companies adopt new ways of managing their information assets and technology, and adjust the ways they interact with partners and customers. Business leaders now have a dynamic environment in which they can transform their organizations; build new ecosystems; and monetize core assets, services and products.

API Connect is the only comprehensive solution that addresses all stages of the API lifecycle, no matter where you are in your journey. As part of the integrated portfolio of IBM products, services and tools, it helps companies achieve all the benefits of the API economy and become cognitive enterprises.

To learn more about IBM API Connect, contact your IBM representative or IBM Business Partner, or visit: http://IBM.biz/apiconnect.

An API Marketplace Primer for Mobile, Web and IoT

TED SCHMIDT

Introduction

Pick up any e-commerce web or mobile app today, and you'll be holding a mashup of interconnected applications and services from a variety of different providers. For instance, when you connect to Amazon's e-commerce app, cookies, tags and pixels that are monitored by solutions

like Exact Target, BazaarVoice, Bing, Shopzilla, Liveramp and Google Tag Manager track every action you take. You're presented with special offers and coupons based on your viewing and buying patterns. If you find something you want for your birthday, a third party manages your wish list, which you can share through multiple social-media outlets or email to a friend. When you select something to buy, you find yourself presented with similar items as kind suggestions. And when you finally check out, you're offered the ability to pay with promo codes, gifts cards, PayPal or a variety of credit cards.

The glue that holds all of those services and applications together in a single, coherent package is the application programming interface (API). APIs are bits of code that allow two other programs to talk to each other by defining the method for a programmer to use to request something from a service or application. In the case of an e-commerce application, dozens of APIs must come together in harmony for the app to work properly.

APIs not only allow an app to consume a service, but they also allow that service to distribute itself much more widely. For instance, Google Maps APIs allow multiple retail brands to implement store locator functionality in their e-commerce apps, all while reinforcing the Google brand itself.

According to ProgrammableWeb, almost 16,000 public APIs are currently available, up from only 105 that were available in 2005 (http://www.programmableweb.com/news/api-growth-doubles-2010-social-and-mobile-are-trends/2011/01/03). This new era of delivering

consumable solutions in the form of API mashups presents a whole new set of challenges and opportunities for IT solution architects and leaders.

In this ebook, I consider some of those challenges and opportunities and look at how they can be addressed by leveraging the power of Node.js. I begin by describing what APIs are and some different ways of thinking about them that may affect the way you think about how your business uses APIs. To do this, I also look at the API economy and how it's changing your enterprise API strategy, whether you know it or not. Finally, I discuss the API lifecycle along with why and how to manage it.

Whether you're building mobile, web, IoT or cloud solutions, chances are you'll be using APIs to leverage some existing service. How you use and manage your APIs will be critical to the success of your technology solutions as well as your selling channels. Let's get started.

Current State of API Development: Mobile, IoT, Cloud and Web App Challenges and Opportunities

What Exactly Is an API? In a sense, APIs have existed since companies decided to exchange electronic data instead of paper. Electronic Data Interchange (EDI) was one of the first standardized formats for the electronic interchange of data between businesses. APIs really came into existence because of the internet when Salesforce.com created the first API in 2000. As the internet has grown and changed, the architecture model for APIs also has changed. Service-Oriented

Architecture (SOA) helps better manage the relationships between providers and consumers of information in an environment that's constantly changing, which allows APIs be more robust in handling requests, because it creates a service layer with a standard communication protocol that reduces the complexity of integrating between discrete and changing systems.

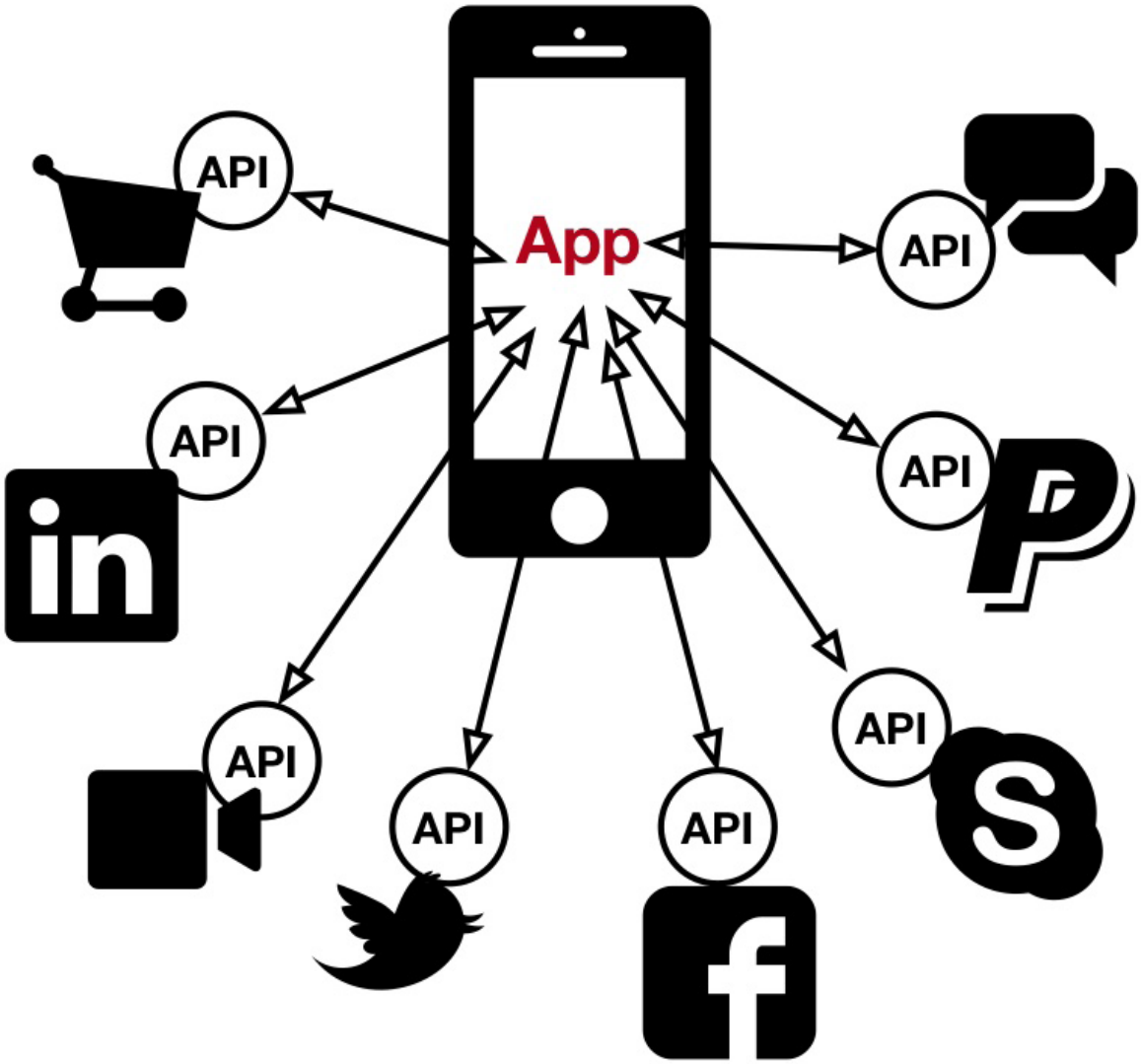

FIGURE 1. What Is an API?

Fundamentally, there are two SOA Strategies: Simple Object Access Protocol (SOAP) and REpresentational State Transfer (REST). There has been a lot of discussion about which is the better approach, so it's good to understand the fundamentals of each strategy and to learn why REST has become the preferred approach for mobile, web and cloud applications.

SOAP uses Extensible Markup Language (XML) to allow programs running on different operating systems to communicate with each other over different transports, including HTTP, SMTP and FTP, although it typically uses HTTP. SOAP uses a formal structure and very well defined messages, and as a result, those messages tend to be big. SOAP uses operations, like procedure calls, and sends fewer (but larger) messages than REST (which I'll discuss in a bit). SOAP also requires two programs to be written: one for the client to request data and one for the server to provide it.

REST also uses XML, but it also can use JavaScript Object Notation (JSON) to script messages over HTTP and only HTTP, so it's often used to develop web services. Because it uses lighter-weight messages, it also uses less bandwidth, making it perfect for internet APIs, which are called RESTful APIs. RESTful APIs are easier to use, and they're the method preferred by cloud service providers, such as Amazon. RESTful APIS also scale easier and are simpler to learn, which make REST a much more likely choice for mobile, web and cloud applications.

REST is a better choice than SOAP any time the scalability of an API is the primary concern, or when there are a lot of consumers of the API. In other words, a billion mobile phones accessing Google Maps is a good time to make use

Although an API is essentially an interface between two programs, the importance of APIs is increasing as enterprises continue to shift toward a more collaborative model that uses multiple partners and service providers to create an effective, familiar user experience that still maintains brand uniqueness.

of a RESTful approach. For such an application, the large, structured format of SOAP provides no benefit. This is not to say that SOAP has no benefit. To the contrary, any time you're in an environment that needs the structure of SOAP or that might be audited, SOAP is the definite choice. The bottom line for the discussion here, and the point you should take from all of this, is that for APIs involving regulated (financial, medical and so forth) transactions, SOAP is probably the best approach. For public APIs that experience high traffic, a RESTful approach is more appropriate.

Although an API is essentially an interface between two programs, the importance of APIs is increasing as enterprises continue to shift toward a more collaborative model that uses multiple partners and service providers to create an effective, familiar user experience that still maintains brand uniqueness. An API is how any internet-connected application consumes a service. Usually, when people talk about APIs, they are thinking about web APIs, like they think

about web services. It's simply a documented method and format for an application to get information from a separate service or application—store locations, coupons and product catalogs are all examples of data that can be served by APIs.

Since APIs are instructions and formats for getting and receiving data from a service, they tend to remain more constant than, say, an actual website, which is continually being updated and refreshed. That's not to say that APIs never change, but they change with the same types of controls and discipline as a program itself. And when they do change, all the consumers of that API need to be aware, so that programs using the APIs can make any necessary changes as well.

Types of APIs One way to look at APIs is from the perspective of where they are used and who makes use of them. Following this perspective, there are basically three types of API: private, partner and public. Some enterprises roll their own private, or "enterprise", APIs. Typically, they do this for web or mobile apps that are consuming internal resources. For instance, customer account data can be delivered to a consumer mobile app through an internally developed API. This makes sense if you don't want to get tied to the release cycle of an external partner, if you're efficient in providing the service needed by your web or mobile app, or if you're really just creating an interface between two enterprise systems, such as an e-commerce web app and a back-end fulfillment system. However, this tends to be a more expensive approach to API management, because a lot of overhead is associated with maintaining a library of APIs. Certainly, there is salary cost, and the cost associated with

configuration management, but there are other inherent risks, depending on the data being stored and delivered through the API. For instance, PII (Personally Identifiable Information), such as medical history or credit-card

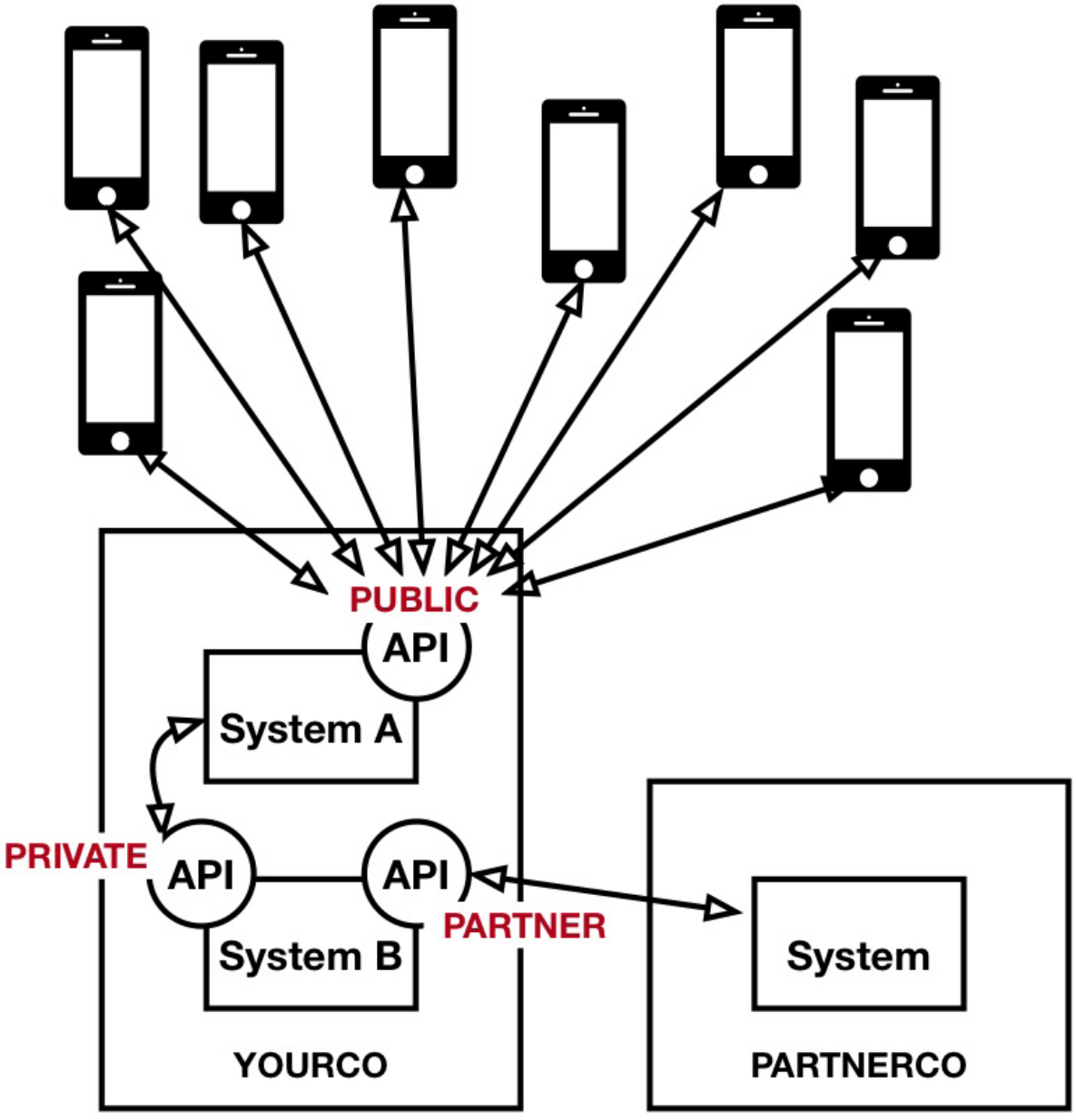

FIGURE 2. API Types

information, must be managed very carefully through encryption and other security techniques. These increasingly important security issues actually lead some enterprises to look externally for services that will secure and manage sensitive data, and simply provide access to it through APIs.

This leads me to the second two types of API, both being provided by an external service provider. The first, to which I alluded, can be referred to as a "partner" API. Companies that provide these types of services, like credit-card processors, employee payroll processors, Mail Chimp or SalesForce, tend to be strategic business partners that provide a key service, but not a service your enterprise wants or can afford to develop an expertise in itself. The challenge here is that your enterprise becomes dependent on changes made to APIs by the service provider. The nice part about using this approach is that when your business strategy changes and the enterprise wants to move to a different provider, you simply can "unbolt" from its APIs and connect to another service provider. This is actually a bit more complicated than I'm making it sound, but changing the APIs between your employee time-card system and an external payroll service is probably simpler than implementing a whole new payroll system.

It's important to remember that investing in a partner relationship is no small matter. Partner APIs can include banking or financial data, medical and health data and other PII. These types of data typically have some external governing, oversight or certification body. Using a partner API can subject your enterprise to audits and controls that are dependent on an external partner's capabilities, which in

turn introduces risk into your organization. Imagine having your credit-card processor lose its PCI compliance. How quickly do you think you could find a new one and change out all your APIs? This is an extreme example, but it's made to illustrate that using a vendor API may save costs in one place, but could introduce risk in another.

When considering a partner API, implement an approval and review process even if there isn't industry oversight, understand the liability your enterprise faces when using the API from a partner, consider the financial impact as it relates to risk, and read the contracts carefully. You'll need to be a little more structured here, ensuring that data is protected or even encrypted. Make sure you have access at any time to what's going on behind the curtain, and arrange policy and compliance reviews up front.

This leaves the third type of API, the "open", or "public", API. In developing mobile, web and IoT apps, this is likely the type of API you will use most often. Google provides multiple APIs for Maps, Adsense, Adwords, Earth, YouTube and so on. Similarly, Facebook, The New York Times, Yahoo!, USA.gov, Twitter, OpenTable, LinkedIn and Amazon are just a few among almost 16,000 public APIs that can be found at ProgrammableWeb (http://www.programmableweb.com/apis/directory). These public APIs allow an enterprise to access a plethora of data that adds value to a mobile, web or IoT app without incurring the increased cost that would come from developing and managing those data sources internally.

Typically, information that's provided through public APIs is already publicly accessible and not sensitive or subject

to any regulatory oversight, like weather and traffic data. Public APIs also are typically commodity services that are provided without a fee. The advantage to consuming these APIs is that they are low-cost (if any), require little security (if any), and they usually are easy to use and have a low cost to switch if the service you are consuming doesn't quite meet your needs.

It's in this third type of API that things get really interesting because of several factors. First, since its advent in the 1990s, the internet has exploded in terms of speed, usage, reliability and the sheer number of connected devices. In 2015, there were more than 13 billion devices connected to the internet. And, according to Juniper Research, that number should triple to almost 39 billion by the year 2020 (http://www.juniperresearch.com/press/press-releases/iot-connected-devices-to-triple-to-38-bn-by-2020). And these devices are not just laptops and mobile phones; they include security cameras, refrigerators, automobiles and even medical monitoring devices.

All of these devices, and the people who use them, are expecting access to data in an instant. The solution is for data to be served by its provider through a single point of contact to every device. It's called the "API First" approach, and it ensures and simplifies maintenance and versioning controls, as well as security, by forcing this myriad of devices to conform to a single API in order to access data. In other words, before building the apps and devices that consume the data, build the API to the data.

The API Marketplace: Opportunities and Challenges

Up to this point, I've been talking about APIs pretty much

In other words, before building the apps and devices that consume the data, build the API to the data.

from the perspective of the API consumer. What about the perspective of the service or data provider? In other words, what if your enterprise has access to data or provides a service that you've been consuming internally, and sees a potential market for that data or service delivered in mobile, web or IoT?

For instance, if you were a title or real-estate company that had access to new home sales data, there might be remodeling, insurance or landscaping companies that would pay you for access to your data via a common API. Suddenly, your API becomes a product that you need to think about marketing and selling. But, you're in the business of selling real estate; you're not a technology company. Rather than trying to build a mobile app that provides new home sales data, all you need to do to get that data into the market is provide an API. Let someone else incur the cost and complexity of building an app that has to consider multiple platforms and devices. All you really need to think about is your competition. Who else could provide that data?

Earlier, I wrote that most public APIs are free or very low cost. So what is the benefit to creating a public API? Well, depending on your enterprise's strategy, you may want to make your data available only to specific business

partners, thereby sticking with a partner API strategy. Or, you may want to expose APIs to the public as a means of advertising or creating brand recognition. If you don't share that data, chances are a competitor will. It used to be that the last player with a website was the loser. Soon, it will be the last player with an API that will lose the race. As more internet-connected devices (including smartphones, tablets, thermostats, televisions, automobiles and so on) require more data from APIs, the more demand will increase for APIs, and the more enterprises will need an API strategy.

The API Lifecycle Like any software, APIs have a lifecycle. And like any lifecycle, the API lifecycle begins with planning. Call it planning, envisioning or analysis, whether your enterprise is concerned with private, partner

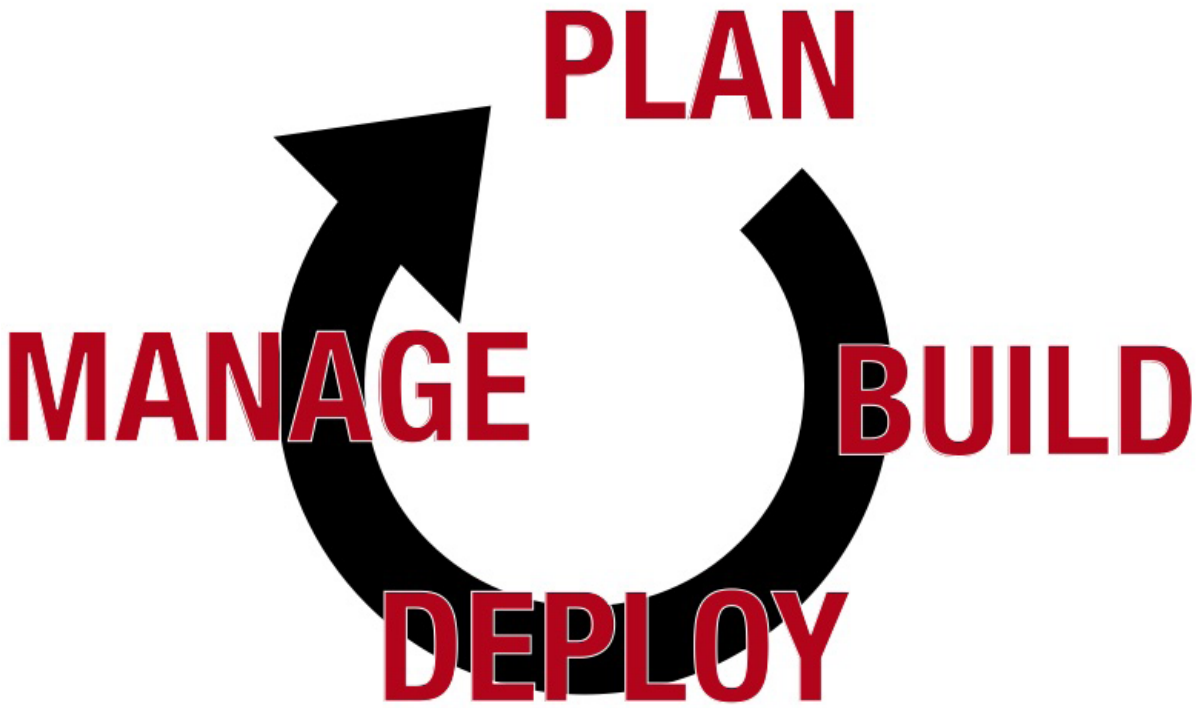

FIGURE 3. API Lifecycle

or public APIs, it's a good idea to decide what your business objectives are in terms of APIs. Are you going to enter the burgeoning API marketplace or just pursue an RYO agenda? Either way, you need a plan that covers entry and exit.

If your strategy is focused on private APIs, it's going to be more focused on keeping costs down by creating efficiencies and stability into your organization. If your strategy involves the use of external partners, planning gets a bit more complex as you consider the viability of long-term relationships and the ease of replacing those relationships that are no longer effective. Finally, a strategy built around public APIs is going to be much more market-driven. In this case, organizational agility and experimentation trump standardization and stability.

After the planning phase of the API lifecycle comes the construction or development phase. Again, just like any software development lifecycle, you'll need to translate the API strategy that was defined in the planning phase into business requirements and a usable product. Versioning, hosting, management and distribution all come into play during the construction of APIs. The platform on which you choose to build your APIs is critical at this point, especially if you are building APIs for mobile, web or IoT. It's important to understand whom you are serving and what you are serving. When serving the web, the ability to handle multiple I/O threads efficiently becomes the defining characteristic of your API.

Once you've built your APIs, you look to distribution to your customer base. If your enterprise is pursuing a private or partner API strategy, then distribution becomes more

focused on ensuring a secure developer portal, entering into partner agreements and performing what auditing may be required by your industry. For instance, if you are handling credit-card transactions, you need to ensure that each of your partners is PCI-compliant before allowing access to your API developer portal. And, you may need to think about re-certification as well.

If you are pursuing a public API strategy, distribution becomes more focused on things like marketing, training, ease of use and support for all those developers in the wild who you want to use your API. Because a public strategy pits your API against any college drop-out who stays up all night developing open APIs while living off Mountain Dew and a burning passion to be the next Zuckerberg, you have to be agile, customer-focused, and have a product that is extremely easy to get to and use.

Now that you've sent your API baby into the world, it's time to start managing the onslaught of traffic. For private and partner APIs, managing is more about security, control, workload optimization and analytics. For public APIs, management includes all those concerns, albeit in a simpler format (that is, security is typically limited to user name, password and some form of client authentication). However, for public APIs, the distribution and volume of traffic will be astronomically bigger than it is for private and partner strategies. This is when scalability becomes critical.

At some point, you'll want to bill your customer for the use of your API and underlying service, especially if you're pursuing a partner strategy. This also can be done

during the management phase, because it's during the management phase that you're monitoring the usage of your API. That being said, if your API is public, it is going to be fairly difficult to take a fee for it, unless you have a corner on some service or data. Remember that developer jacked up on the Dew? The moment you start charging a fee for a public API, she's going to develop an open-source version and put yours out of business.

It's at this point that the lifecycle iterates back on itself. Earlier, during my discussion about planning, I mentioned that your API plan needs to include both entry and exit considerations. During the management phase, you watched the usage stats to determine how much your API was being used and by whom. As you continue your ongoing API planning, it's important to call something done and deprecate an API any time you see things like limited use, absence of third-party innovation (the Mountain Dew developer moves on to something shinier), poor financial performance or other factors that tell you the party's over.

At this point, several things should be obvious about competing in the API economy:

1. You need an API strategy and plan.

2. You need to manage and secure your APIs according to your plan.

3. You need to assess your systems to determine where APIs add value.

Managing the API lifecycle doesn't have to be complicated. But like any good discipline, it will help your enterprise be successful when it comes to an API strategy.

4. You need to provide simple, secure access to your APIs for developers and partners.

5. You need to ensure API controls and security as appropriate across your API inventory.

Managing the API lifecycle doesn't have to be complicated. But like any good discipline, it will help your enterprise be successful when it comes to an API strategy. Tools like API Connect (https://developer.ibm.com/apiconnect) are tremendous benefits in managing the API lifecycle, especially when your enterprise is grappling with an API strategy that is impacted by the demands of billions of devices on the internet, all hitting your API at the same time. But before looking at API Connect, let's take a brief look at the open-source runtime that it's built on: Node.js.

Why Node.js Is the Solution for IO Scaling Problems: Event-Driven, Non-Blocking IO, the Largest Open-Source Ecosystem

Used by organizations like GE, Microsoft and PayPal, Node.js is an open-source runtime environment and

JavaScript library that was developed in 2009 by Ryan Dahl and built on Google Chrome's JavaScript runtime. According to https://nodejs.org/en, using an event-driven, non-blocking I/O model, Node.js was built for the purpose of building scalable network applications that would run fast. So, it's perfect for the real-time data demands of applications running across a distributed network of devices, like web apps, mobile, IoT and so on.

In a distributed world, Node.js is perfect for applications that are I/O-bound or require data streaming. It's also perfect for applications that are based on JSON APIs or have real-time data-intensive demands. What Node.js is not good for is anything that demands a lot of compute cycles. Node.js lends itself to RESTful APIs, which are lightweight.

Let's take a quick look at the defining characteristics of Node.js. First, Node.js is event-driven and asynchronous, which means that Node.js servers don't wait for APIs to return data. They just move on to the subsequent request after making an API call. So, it's non-blocking, because it's doing other things while you're waiting on the API to return data. When the initial API does return something, an events notification mechanism in Node.js lets the server know. This allows a Node.js server to handle thousands of database requests from distributed devices. Unlike most servers that create multiple threads to handle requests, resulting in memory being maxed out, Node.js servers use a single thread with an event-looping mechanism that makes the server very scalable. In addition, Node.js doesn't buffer data. Finally, because Node.js is built on the Chrome V8 engine, it executes code fast.

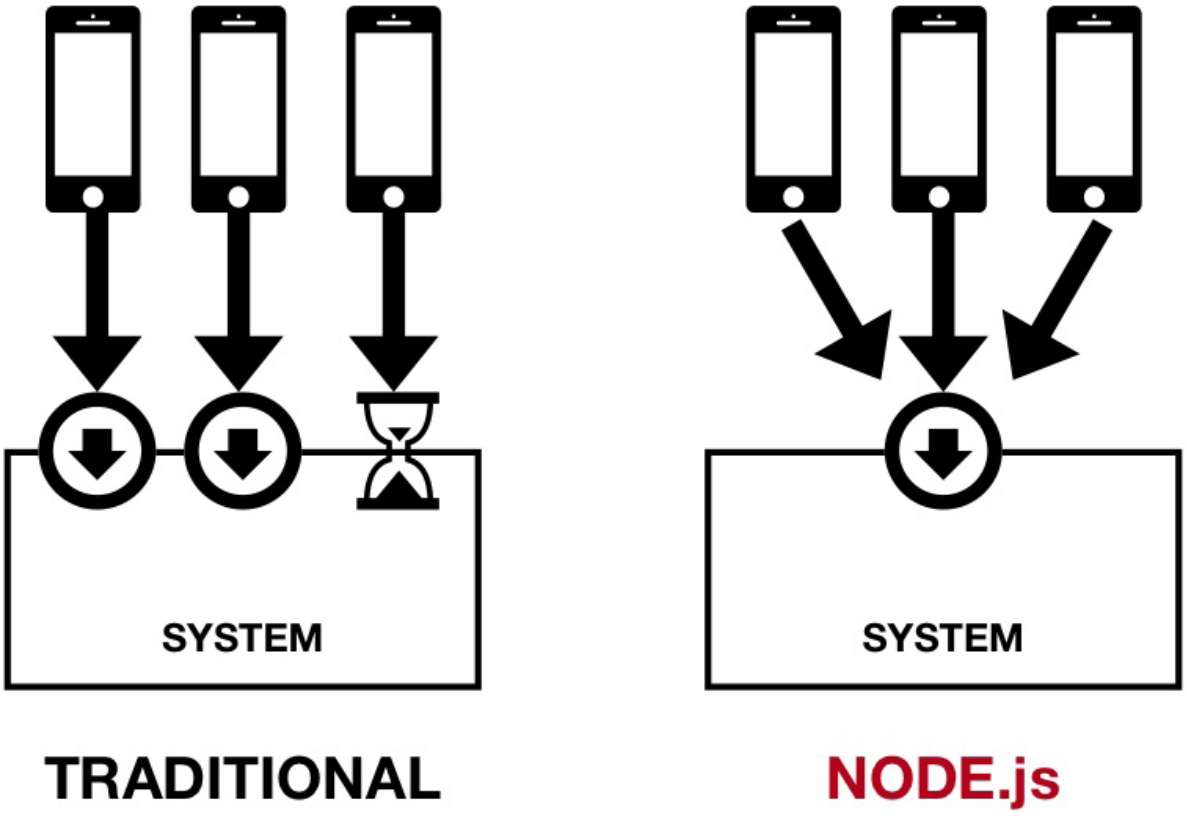

FIGURE 4. Node.js

As I alluded to at the beginning of this section, Node.js is not a web server; it's a runtime environment and JavaScript library, and you have to build your own HTTP servers. Node.js runs on Linux, OS X and Windows, and it is simple to install. You can download it from https://nodejs.org/en/download.

Once installed, you can execute raw JavaScript by using the `node` command without any arguments, or you can provide Node.js with a JavaScript file to execute. The latter is the usual method for using it, and that's a topic better left for another ebook.

All this being said, it's important to remember that Node.js

is not a solution for applications that use a lot of compute cycles; it was created specifically to address the problems associated with having a highly distributed network of devices and applications with high I/O needs, like a mobile app looking for location on a map, or a thermostat trying to locate its owner, or a chat application on an e-commerce website. In short, Node.js was built for mobile, web and IoT applications.

How API Connect APIs (Using Node.js) Are Becoming de Facto Standards and Why

So far, I've discussed APIs and the emerging API economy, and I've made a case for why RESTful APIs, built with Node.js, are the way to go if an enterprise wants to be successful in the API economy. I've also shown why effective management of an enterprise's API lifecycle is so important to that success. Let's look now at what one tool in particular, API Connect, offers in the way of API lifecycle management.

To begin with, API Connect makes use of open-source frameworks for Node.js to increase the speed of development of Node.js APIs. This is great because time to market is critical in the API economy. Whether your strategy is private, partner or public, you need to build high-quality, scalable APIs fast to be competitive. Because it's built on a platform that ensures scalability, API Connect builds APIs that are standard for mobile, web and IoT by default.

But API Connect does more. Not only does it offer a complete suite of management capabilities that covers the development of de facto APIs for mobile, web and IoT, but it also helps developers distribute new APIs faster. Management capabilities are provided through multiple

But API Connect does more. Not only does it offer a complete suite of management capabilities that covers the development of de facto APIs for mobile, web and IoT, but it also helps developers distribute new APIs faster.

portals that make API consumption easy and secure. Add to this version control, governance and administrative security, and you have a complete API lifecycle management tool for private, partner and public APIs.

The bottom line is this: IoT, mobile and web apps demand the scalability that comes from using a RESTful approach with Node.js. API Connect is built to develop on Node.js and to provide complete API management capabilities. In an exploding API economy, it's the model for what an enterprise is looking for.

Conclusion

The API economy is here. With 13 billion (and growing) interconnected devices accessing almost 16,000 (and growing) public APIs, you have to ask yourself whether you're ready to compete. By planning your strategy and building lightweight, RESTful APIs on Node.js, by making those APIs easy to learn and access, and by managing the complete lifecycle of all your APIs, you can be ready to compete in the new API economy.■