# The Essential Guide to API Monitoring

AlertSite UXM
by SMARTBEAR

# The Essential Guide to API Monitoring

## Contents

## Why API Monitoring Matters

We've all heard the stories about the overnight sensation that took decades to achieve. The "API Revolution" has that same feel to me. Some version of APIs has been at the heart of application development for many years, but as the web application world grew more and more connected and developers learned to leverage code built outside their own application, the API industry began to boom and flourish.

### APIs are the core of your application

Most of us can now say that we have built our web applications on a foundation of APIs, both internal and external. It is a common conversation in planning meetings to talk about which APIs we need to expose to the outside world in order to drive business and, in some cases, to build APIs into the product roadmap. With so much emphasis on the tiny API, it's important to recognize its inherent power and make sure you build the appropriate safeguards to protect it.


### API failure can be one of the most impactful failures

Because one API can affect so many applications/components, changing or deprecating that API can cause widespread failures that you may not be able to predict, particularly if that API has been made public. Once you've made your API available to other developers, either in a controlled fashion to trusted partners or in a public way to anyone with a developer/production key, you take on a responsibility to ensure that nothing affects the API's performance. Factors like server load,

the amount of data coming down (you always want paged results, for example), level of encryption, the quality of the controller code, etc., all affect the quality of the API and its performance.

Of course, you should test all of that before putting your API into production, but just like any other important feature, you should also monitor those factors after you deploy. It's when your customers are banging on it in new and different ways that problems surface. That's where API monitoring comes in.

## Industry sees the need

As we all become more reliant on our own and third-party APIs, the industry as a whole is beginning to see the importance of monitoring APIs in production. An API failure can be more catastrophic than an application-level failure because it can bring down so many other dependent applications. As time goes on, the API management platforms are all recognizing the need for some kind of analytics dashboard that goes beyond just showing usage statistics. Those are valuable as well but what about performance data for the APIs you rely on?

Companies like Apiphany and Mashape are starting to put dashboards together for their solutions, which solves a small piece of the problem. But they only provide generic data for the APIs they are specifically managing.

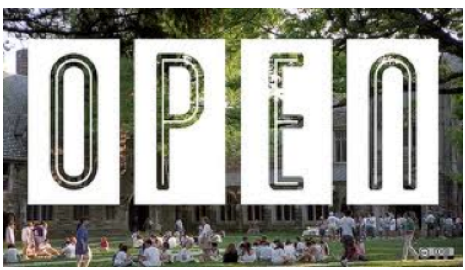
## Only you know what's most important to you

That's not a philosophical statement. That's reality. You have adopted or built an API for a specific purpose within your application. You know what data matters to you, which transactions are most important to you, and therefore which monitors make the most sense for you. Building an API monitor is similar to building an API test but it requires an even more distilled viewpoint. An API test can be very complex and multi-stepped. An efficient monitor should run frequently, at least every hour,

and check only the most important thing(s). You need to be able to identify that essential response that must perform well for your users and set a threshold for acceptable response time. Creating a monitor is an exercise in refinement – distilling your API tests to the single most critical action that signals continued success.

This eBook will lead you through the basics of ensuring your API quality meets the needs of your customers.

## How Open Data and APIs Fuel Innovation




Open Data is gaining constant traction nowadays, perhaps most notably with the recent White House initiative. At its core, it's about making data collected by governments and organizations available to everyone. Instead of providing the pre-molded pieces, the actual building blocks are put out there and, as such, Open Data aims to be the building blocks used by crafty entrepreneurs and technologists out there for building new businesses for us to marvel at.

APIs, on the other hand, are the glue that makes this work. Just like pieces of Lego have a universal way of adhering to each other thanks to those small plastic "bumps and holes", APIs provide a standardized and (fairly) simple way to connect sources of Open Data to each other. By utilizing standardized and platform-agnostic technologies like HTTP, JSON and XML, they are the ultimate facilitators for integrating all those open data sources into innovative applications and solutions.

This is where things get really exciting.

Instead of data providers assembling their data to tell the stories they want to tell, they are giving us the data to tell whatever story we find relevant. You want to build a web app about hats for the benefit of transportation? Awesome! Or perhaps you should build a news app about air for the benefit of the environment? Go for it! (Zany API/data integration ideas by courtesy of the APIRandomizer at http://apirandomizer.com/). The point being: ultimately you and your users will be the judge of what is right for them and businesses will adapt accordingly by changing data sources, adapting usage scenarios, etc.

So what about quality? Quality of all the APIs that are used to glue these Open Data sources together are pivotal to their success. And the great thing is - not only are most of these Open Data sources available for free but so are the tools needed to build and test them. Not only can an innovative API or application idea be built using freely available Open Data sources, but the actual tools for ensuring the functionality, performance, availability and security of these apps and the APIs they consume are out there as well – for free. And be sure to note - some of these are well-crafted tools with large user communities whose vendors and providers realize and understand the value of being part of the whole Open Data and API movement.

And of course there are already a number of cool apps out there harvesting these Open Data sources using API technologies (and presumably being built and tested using free API quality tools) – check out the open-data-applications page at Pinterest to dig into some examples.

## Testing Your APIs

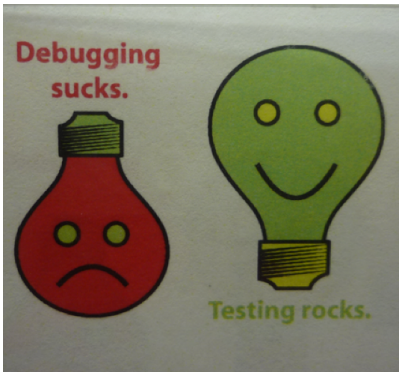
It has become more and more common for software quality processes to span the entire lifecycle from development through production monitoring. The job of building and maintaining quality software cannot end at user acceptance or the hand off to operations. It has to extend to reliability, availability, and performance in operations. This is especially true for the APIs you are using.

The turbos have really kicked in for the API economy. And it's no wonder. APIs are the central technology for applications built in the cloud, for mobile apps, and for third-party applications. If your application doesn't have some reliance on third party APIs, it almost certainly is relying on your own APIs. As the API industry continues to boom, there are more conversations starting about API testing, governance, stability… Recent acquisitions of Mashery by Intel and Layer7 by CA indicate the new corporate need for API governance.

So what's needed? We need to start treating our APIs as features because, in many ways, they are. We rely on them to power our applications, extend our business, engage our partners, and drive customer adoption. Here are some approaches to make sure that the APIs that are most important to you are reliable.

### Performance planning and testing

Spend the time to map out your expectations around up-time and response time. Even if you have not exposed your APIs outside your own corporate infrastructure, their performance greatly affects your application's performance (and, by extension, your user's experience). Know how much load you are expecting, how many concurrent requests you have to handle, and how much data will be passed. Planning ahead for your performance needs ensures that you will do the right kind of testing and monitoring. Having a specific tool that targets API load testing can be more effective than using an application load testing tool that just tests the APIs as a by-product of normal load testing.




## Testing methodologies for functional testing

There are a lot of alternatives for testing your APIs, just as there are a lot of alternatives for testing your application. Understanding the API's definition and exercising its capabilities is as important as if it were any other piece of functionality in your application. There is no right or wrong way to approach testing so you need to determine whether it makes sense to automate your API tests in order to provide clean and repeatable regression tests during future development or perform manual testing of your APIs in order to attempt a variety of conditions and scenarios. In many cases, the right answer is yes. ☺ Automating regression tests allows you to focus more time and resources on exploratory testing – this is often the best way to approach feature testing so there's no reason why it wouldn't be equally valuable to test your APIs using the same approach.

## Production analysis

Production analysis is quickly becoming another arm of Quality Assurance. Many organizations have realized the benefit of not only reporting on production failures but also using production data to perform routine quality analysis of their software. And it's no different with APIs. While there aren't many options out there yet, this is a growing focus in the API community who are beginning to see the value in establishing SLAs and watching any production trends around usage and performance as an important extension of their testing efforts. Building an API production monitor is similar to building an API test, although you probably want to limit the number of steps you perform in the test in order to reduce

cost and improve reliability. Your biggest concern is probably how well does that API perform its basic tasks and how fast does it respond. AlertSite for API Monitoring allows you to take the same tests you built with soapUI and upload them to use as production monitors. This gives you the added assurance that the tests that passed in your lab are also successful in production.

Of course, we're at the early stages of this API boom and things will likely change as we progress, with more emphasis on governance and security over time. But even now, it's an important responsibility for any API provider, whether internal or external, to ensure that their APIs are well-tested and monitored so the business can depend on the functionality they provide.

## Defensive Coding with APIs

Why put coding after testing? A good programmer tests their code and thinks about testers when building it. A great programmer takes the results of testing and monitoring, and works it back into their code. When it comes to integrating with, and depending on, 3rd party APIs in your business solution, you had better think twice of the implications. There are a number of aspects related to 3rd party APIs that you need to consider and handle pro-actively to avoid looking bad to your users and customers when those APIs fail to deliver as required. Let's have a little deeper look at what you could (or should) be doing at the code level to ensure that the volatility of depending on 3rd party APIs is kept to a minimum.

"Defensive coding" is a practice that is well described in the programming literature, for example in Defensive Programming: Being Just-Enough Paranoid, Jim Bird outlines some of the basic principles and underlying reasons for defensive coding. And in the just as informative and entertaining AGGRESSIVELY DEFENSIVE PROGRAMMING,

Terence Eden gives some pretty awesome and unexpected examples of how defensive coding can help track down the strangest of issues.

So how about APIs? Are there some specific defensive coding techniques developers should put to work to shield them from unexpected events? Sure there are, many derived from those already mentioned in the above references. Let's have a look at a few obvious (and thus important) ones:

1. **Code for errors** – There is no denying it; errors will happen, especially if you are integrating with someone else's code over a network. Thus, error handling should be part of your general application logic and not solely as a "try-catch" statement surrounding your entire block of code. Handling errors as an "expected" application event forces you to create a non-disruptive experience for your users (instead of just an error message) and greatly increases the perceived stability of your application.

2. **Handle timeouts** – No matter how fast those 3rd party APIs are guaranteed to be, there will be situations when they aren't. This could be caused by network conditions, 3rd party API overload or a DOS attack – whichever, you need to handle it gracefully. Make sure your API calls have reasonable timeouts, and perhaps even run them in a separate thread so you can use the main thread to provide feedback to your users while the call is waiting for its response.

3. **Validate data proactively** – the JSON/XML/whatever data returned by a 3rd party API (hopefully) follows some kind of data definition – either verbal or machine-based (JSON-Schema, XML-Schema, etc). Be defensive and validate that the response actually complies. Date formats, timestamps, decimal numbers, etc can all be subject to "format changes" when system components are updated at the 3rd Party API site. This can introduce subtle changes not detected by their regression tests (if they have any), or not expected

to make any difference. For example, your 3rd party API might add another 4 decimals to the map coordinates it returns, but your code might be parsing these as strings and thus not handle a change in length. Or they might change from lower to upper case in a field containing weekdays, which your code didn't expect to handle.

4. **Honor caching** – One of the core principles of REST is to make use of the HTTP caching mechanism, and many REST APIs provide caching information with response representations. Unfortunately, most HTTP client libraries don't have any built-in handling, which results in unnecessary HTTP calls to the 3rd party APIs and in turn decreases performance of both your service and the 3rd Party API. Make sure that your code caches resources locally as indicated by these HTTP headers and uses this cache as much as possible. The advantage is two-fold - it will greatly decrease your dependency on the availability of the 3rd Party APIs and it might decrease the load on those APIs considerably,making them more responsive when you actually need them.

5. **Throttle performance** – While all the above recommendations have been about functionality, you can be defensive in regard to performance as well. If you design your code to "notice" when it is under heavy load it can take precautionary actions, for example by throttling requests to 3rd party APIs and redirecting users to failover servers or simply letting the user know that performance may be suffering.

Obviously you shouldn't overdo any defensive coding, but the above suggestions aren't very hard to implement if you keep them in mind from the start – and the consequences of neglecting the underlying issues can be grave for you and your users.

As a side note, it strikes me that coding defensively is a little like preparing your first dinner for your to-be in-laws:

- Sample fresh ingredients when buying (instead of trusting the shop)
- Try the food while preparing it (instead of just trusting the recipe)
- Have a backup plan (instead of trusting your power/water supply)

Knowing that you've minimized the risk for disaster like this it is good to know there is only one thing that can go wrong in this situation. You.

## Creating API Monitors

Creating monitors for APIs you either provide or consume (or both) should be a cornerstone in your online quality strategy. One of those last excuses stopping you from setting up your first API monitor might be that you don't really know what to consider and where to start when setting it up – so let's get you going with some hands-on tips for creating your first API monitor.

### Re-use functional Tests from development

If you're providing your own APIs then re-using functional API tests created during development and testing for monitoring has several advantages:

- Instead of just checking availability, "real" functionality of your API will be scrutinized continuously, providing you with a safety net for continuous deployment practices and infrastructure changes

- Given that your functional API tests are set up to assert and provide relevant error messages, the corresponding API monitors will have the ability to give you much more detailed error information for root cause analysis than regular availability monitors.
- Given that your functional API monitors mimic expected usage scenarios, their actual structure can tell Ops how your APIs are expected to be used, and help them set up the API infrastructure accordingly.
- Obviously, using one tool for creating tests and monitors is lower overhead in maintenance, learning, cost, etc.

### Create tests that mimic your use cases

If you are mainly integrating with 3rd party APIs (for example Twitter, Google Maps, etc) you need to make sure that you know about their failures before your users notice. Here it is essential that the monitors you create actually mimic how you use that API; for example, if you are using the Flickr API to get the latest photos for a certain group on Flickr, make sure your API monitor does the same thing, and not get a list of popular cameras (or anything else that is easier to set up). Also, make sure you monitor the entire flow of your use cases; don't just monitor the first API request; monitor them all, in sequence – just like your application uses them.

### Be prepared for changing data

This is a tricky one when it comes to monitoring 3rd party APIs. Often your monitor will want to validate some kind of output based on your input; for example you might validate the coordinates or route-plan you get back from one of the Google Maps APIs to give you the expected results every time. Unfortunately though, Google updates coordinates on the 4th decimal rather frequently, so if your monitor doesn't take that "volatility" into account – it might fail unnecessarily. The same goes for

route-planning; perhaps a traffic jam is making Google return a different "unexpected" result for a limited period of time – something that you need to be prepared for (and that you could actually use to your advantage – I'll save that for another blog-post).

On the other hand; if you're not going to validate the returned data in an API monitor, what are you going to validate to make sure it's working as required? This is a tough call to make, I recommend you at least be defensive with data-validations; don't do too many and try to focus on those you think won't ever change.

## Use a dedicated account

Many (most) APIs require you to specify some kind of credentials or access key in your requests; make sure you are using a dedicated account(s) for your monitoring, both for your own APIs and 3rd party ones. There are several reasons for this, including:

- for your own APIs, it makes you run your monitors with the same access rights as your users, so you can detect problems that might not affect "super-user" accounts
- for 3rd party APIs, it allows you to plan billing and utilize bandwidth separately from your "production" API usage

## Don't overdo it

As much as we want you to use our products (free and commercial), I urge you to not overdo your API monitoring. You probably don't need to monitor from every location in the world, or exercise all operations in all your APIs with every possible input. Instead, make sure your API monitors are a safety net for catching problems within the "hot-spots" of your API and make sure to cover those areas that are most vital to your API business. Overdoing things will just result in poorly maintained API monitors, which start to fail, which you will start to ignore, which in the end won't provide any value to anyone (not even us).

## Creating API Monitors from your Open Data Project

At SmartBear, we are totally jazzed by so many of the cool software initiatives going on right now. One that particularly strikes our fancy is the Open Data initiative and how the US Government is jumping into the fray with both feet. If you are using any of the government APIs in your project, we recommend you treat them as you would any third-party API. What does that mean? Well, for us, it means that the quality of your application is only as good as the quality of the foundation it's built upon… and that includes the APIs you use to retrieve data.

To repeat our mantra about third-party APIs:

### Research

Know what your choices are. There are a lot of APIs out there, even in the Federal space. Make sure that the API you've chosen to use is the one that gives you access to the data you expect.

### Defend

If you are depending on that API for your application's core functionality, best to be defensive. If that API doesn't respond or is modified (or worse, abandoned), you should make sure that you have coded defensively so your application doesn't fail completely.

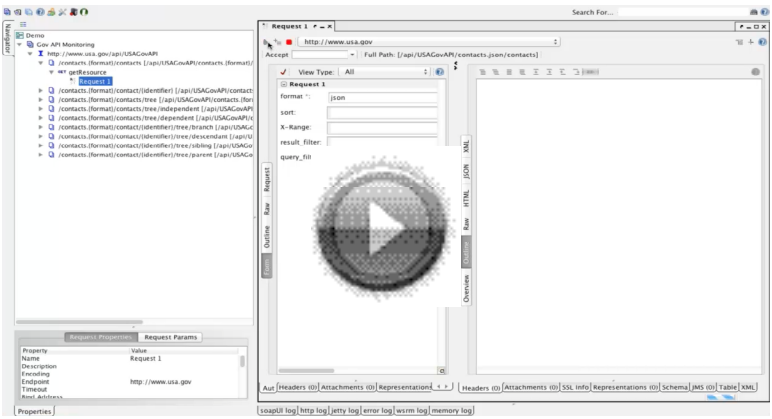
### Test

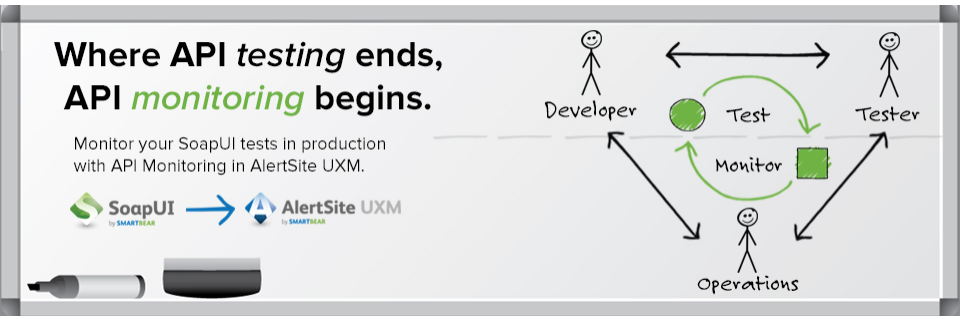
Your testing plan should include third-party APIs as well as the APIs you build yourself. Your consumers assume that the quality of your application is your responsibility so it is a good investment of your time to test all of the APIs you rely on.

### Monitor

Even after your APIs pass your tests, you should create production monitors that exercise those APIs regularly. This will make sure that you get notified whenever something unexpected occurs with those APIs.

We've put together a little example for you so you can see how to create a test and monitor for one of the Federal APIs, using our Swagger plug-in for SoapUI and our AlertSite for API Monitoring offering.






## About SmartBear Software

More than one million developers, testers and operations professionals use SmartBear tools to ensure the quality and performance of their APIs, desktop, mobile, Web and cloud-based applications. SmartBear products are easy to use and deploy, are affordable and available for trial at the website. Learn more about the company's award-winning tools or join the active user community at http://www.smartbear.com, on Facebook or follow us on Twitter @smartbear and Google+.

SMARTBEAR

SmartBear Software, Inc. 100 Cummings Center, Suite 234N Beverly, MA 01915
+1 978.236.7900 www.smartbear.com

SB-AS-102814-WEB