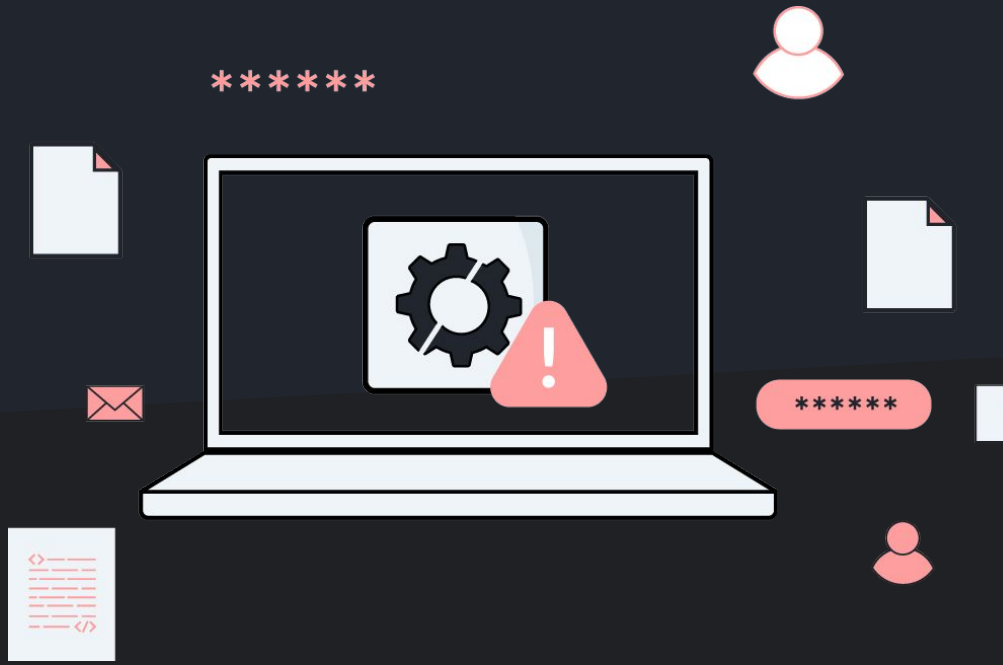
APIFORTRESS

How QA Gaps Can Lead to Dangerous Bugs in Production



With Patrick Poulin
CEO & Co-founder API Fortress

Who Are You?

API Fortress has been in the API testing and monitoring space since 2014 when we started building the platform's SaaS version. Now we have thousands of users spread all over the world, and offer both a SaaS and Self-Hosted version.

Famous Breaches

Data breaches are big and scary, and they should be. The problem is that the word "breach" implies sophisticated hacking, but the reality is most causes are related to basic problems.

The Internet runs on APIs, and most API failures are due simply to human error. That's so disappointing because of how easily many of these issues could be solved.

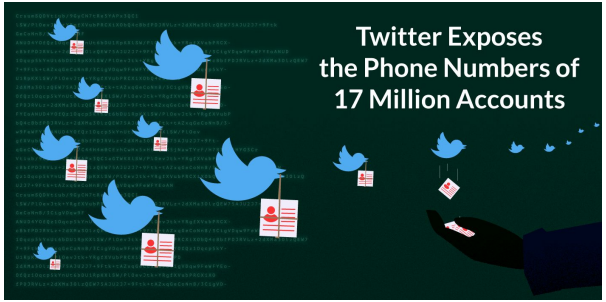
Akamai's State of the Internet report found "83% of all web traffic in 2018 was in API traffic."

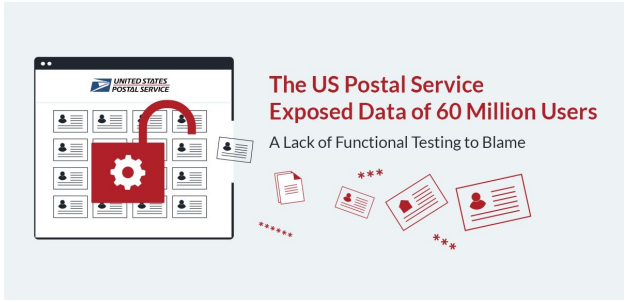
"Through 2022, at least 95% of cloud security failures will be the customer's fault."

Jay Heiser, VP at Gartner

Security

On our blog, we cover three recent API vulnerabilities that went live due to a simple lack of proper testing and monitoring. Failures to take corporate responsibility by the companies and government organizations resulted in huge API breaches that exposed the private data of billions of individuals.





BLOG: https://apifortress.com/blog/

Twitter

Exposed the phone number of 17 million accounts

Twitter's breach may have been due to a simple design flaw. During registration, you could connect with people in your Contacts. The problem was a lack of rate limiting. This led to bad actor countries guessing phone numbers and connecting people's personal numbers with their Twitter accounts. Not everyone wants their Twitter identities tied to them personally and professionally for reasons that may include blackmail.

India & Public Utilities

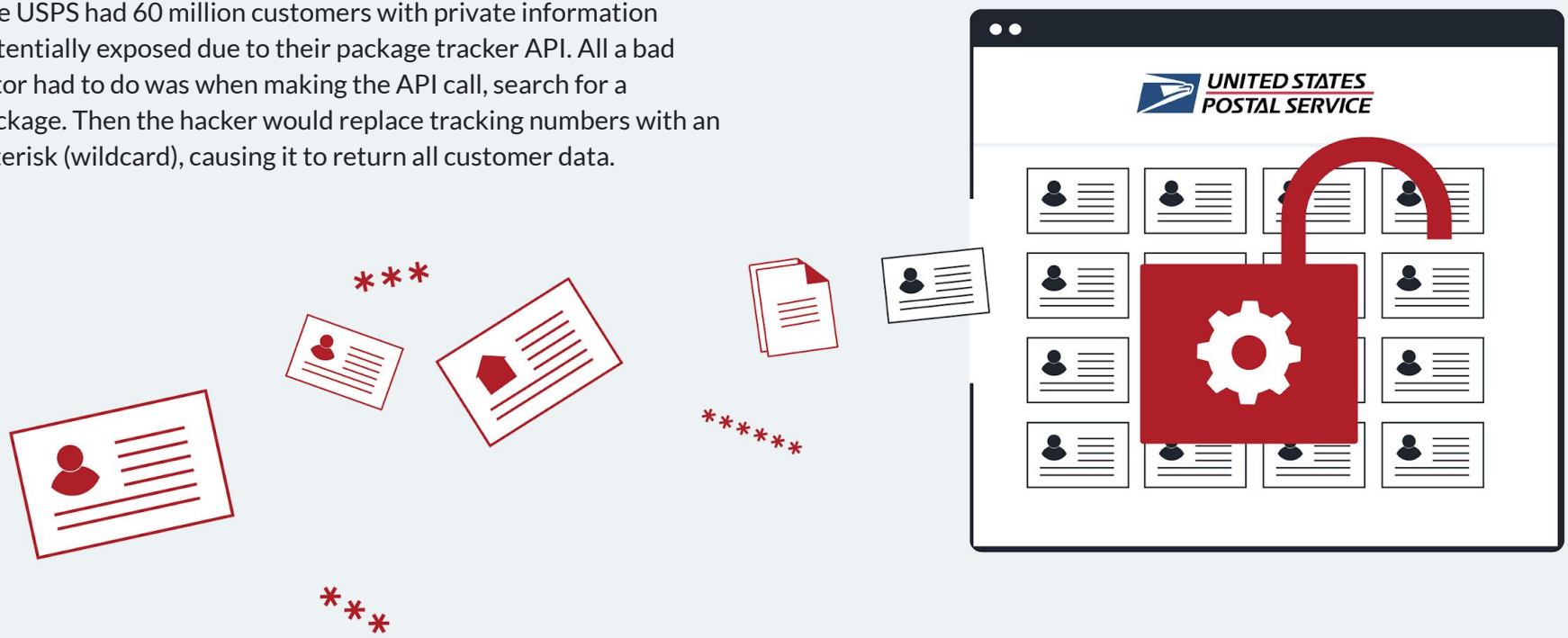
1.1 Billion customer identities exposed

In India, there was a huge breach with a company named Indane, a large utility company in India. When creating an account, they would verify your personal information using the Aadhar API. The problem was that it was publicly accessible and had no rate limiting. This meant that anyone could guess millions of “social security” numbers, and if they were right, they got someone’s personal information. Aadhar exposed 1.1 billion customers as part of the breach.

United States Postal Service

Exposed data of 60 million users

The USPS had 60 million customers with private information potentially exposed due to their package tracker API. All a bad actor had to do was when making the API call, search for a package. Then the hacker would replace tracking numbers with an asterisk (wildcard), causing it to return all customer data.



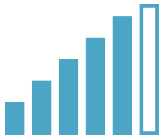
What Is Continuous Quality

The thing we've come to learn in our time with API testing and monitoring is that continuous quality requires a mixture of three things - Functional Quality, Functional Uptime, and Security.

Quality

"Through 2022, at least 95% of cloud security failures will be the customer's fault."

Jay Heiser, VP at Gartner



Reliability

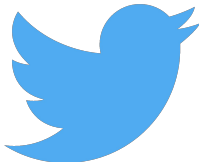
SLA's Uptime vs Functional Uptime

Security

Akamai's State of the Internet report (pdf) found "83% of all web traffic in 2018 was in API traffic."

Security

Not focusing on all three can lead to huge vulnerabilities.



Twitter

17 million accounts

India

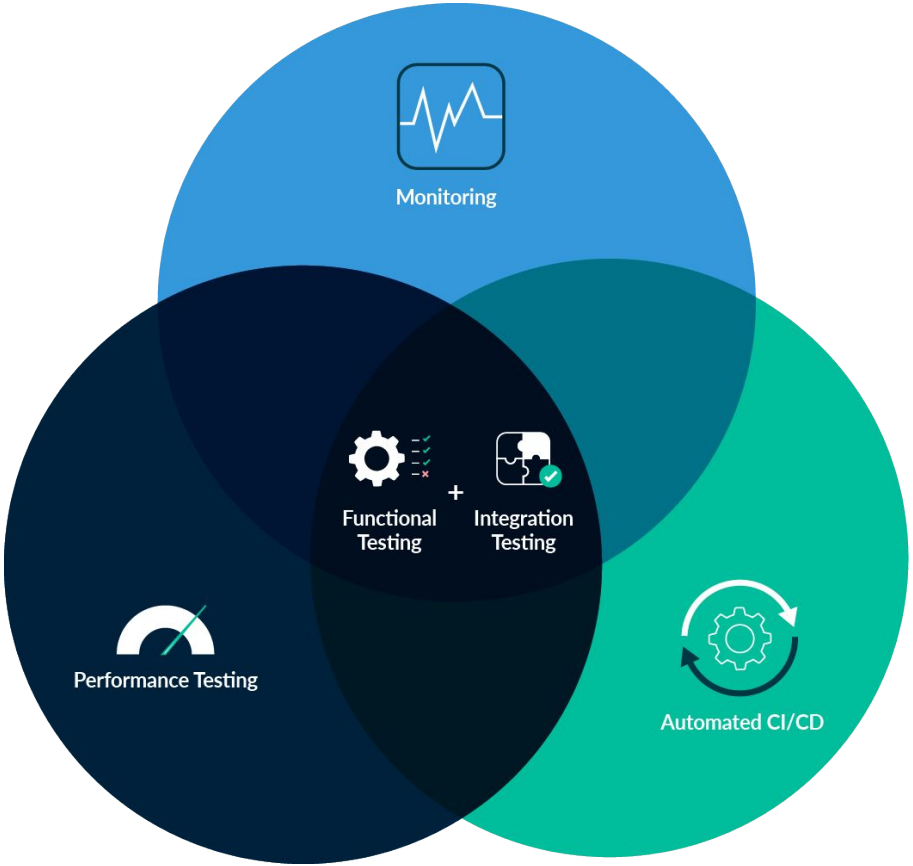
1.1 billion identities

USPS

60 million personal account details

Continuous Quality Testing

The first key to API quality is doing proper testing throughout the lifecycle. The core of your API testing should be a series of very detailed and dynamic functional and integration tests. Then using those as the basis of your automated testing, which is common, but also as part of your load testing and monitoring. A load test and monitor isn't valuable if they aren't reproducing a normal consumer flow.

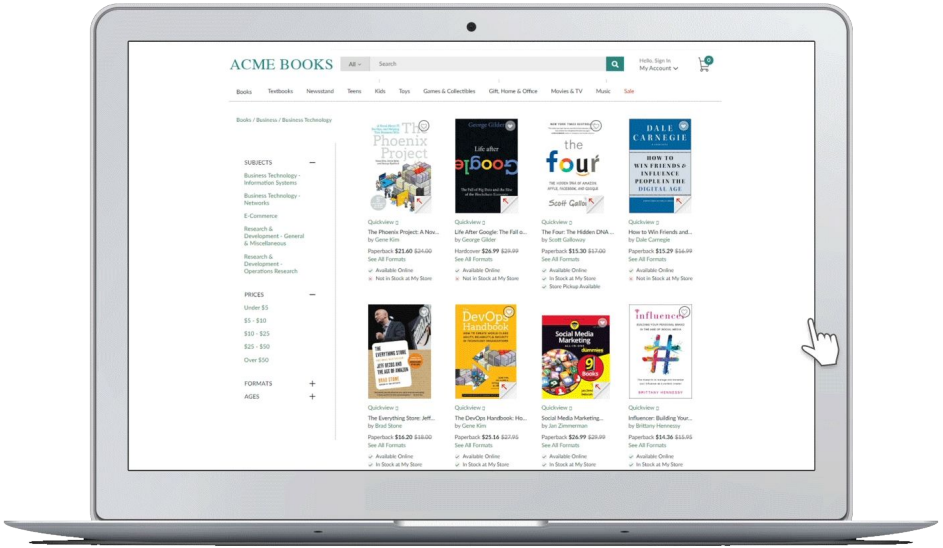


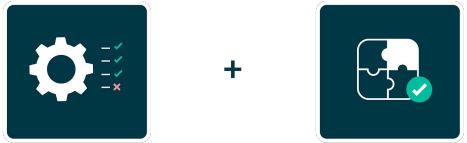
What's a Good Functional API Test?

Many companies we work with set up tests that simply make a call and look for a Status 200 OK. An API is very verbose and detailed. So a proper monitor should be analyzing the entire payload, every object, and the data associated.

We saw the quality gap or false-positive created by insufficient coverage with an early customer. They were a book publisher with a partner API that listed all current books in print with associated ISBNs; they also offered another API that gave you product details like MSRP on specific ISBNs. They had partners complaining about unexpected errors, but the publisher's tests and monitors had showed no issues.

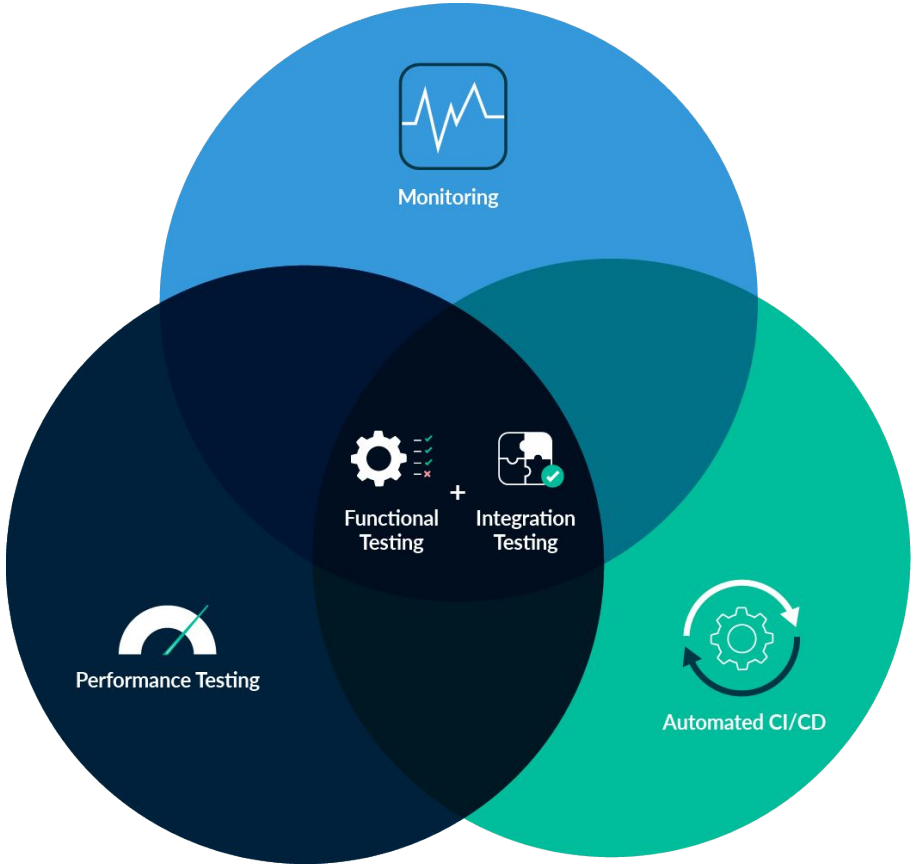
So we connected the two API tests, meaning running a call of the product listings endpoint and then using that endpoint as the data for the product details test. That's when we detected and diagnosed the issue that no one could find. Basically, they updated their products database every two weeks, but forgot to clear the API manager's cache, since it was automatically caching the products endpoint. So for hours until it cleared naturally, hundreds of out of print books were listed as "In Print."





Functional Testing + Integration Testing

Going forward, the book publisher as with our other customers have benefited from unifying functional and integration testing as part of the automated testing driving their agile or CI/CD workflows. Additionally, many of our customers have connected their unified functional/integration tests with load tests to run a true end-to-end automated API test that expands coverage and reduces false-positives. These E2E API tests on our platform can be deployed as monitors that catch huge problems that might otherwise exist in production for weeks or longer.



Show Me How

APIFORTRESS

Functional Testing

First, the core question is: What Is a Good API Test? It all starts with creating a good functional test. What's a functional test? Imagine you are an ecommerce company. A typical user journey can start in many places, and one of them is with search. So for a search endpoint, you want to call that search API, and then analyze the entire result.

http://demoapi.apifortress.com/api/retail/product?q=red


```
[
        {
                quantity: 5,
                color: ["white","red"],
                price: 29.99,
                imageURL: http://apif.com/baseball_cap.jpg,
                name: "Baseball Cap",
                description: "Boston Red Sox Baseball Cap",
                id: 1,
                category: "head",
        },
        {
                quantity: 7,
                color: ["blue","yellow","red"],
                price: 39.99,
                imageURL: http://apif.com/long_sleev_shirt.jpg,
                name: "Long Sleeve Shirt",
                description: "A wonderful long sleeve shirt",
                id: 2,
                category: "body",
        },
        {
                quantity: 50,
                color: ["red","gray"],
                price: 49.99,
                imageURL: http://apif.com/earmuffs.jpg,
                name: "Earmuffs",
                description: "Keep those ears warm in the winter!",
                id: 3,
                category: "ears",
        },
]
```

End to End Testing

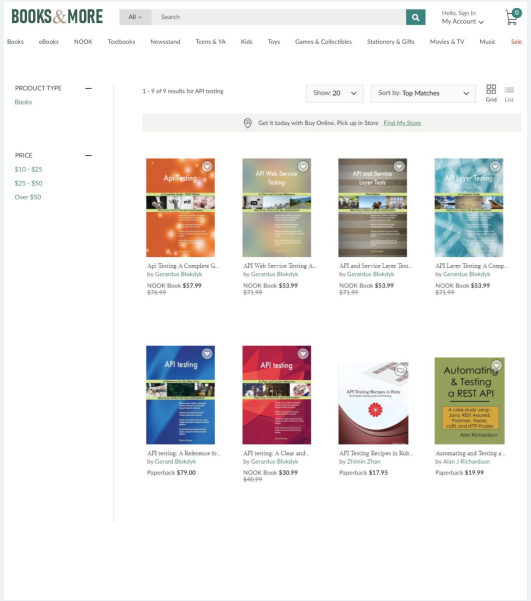
So, first you start with a search for red, then you use that search as your data for the next call, which randomly dives into a handful of products. But for this example, we'll just choose one.

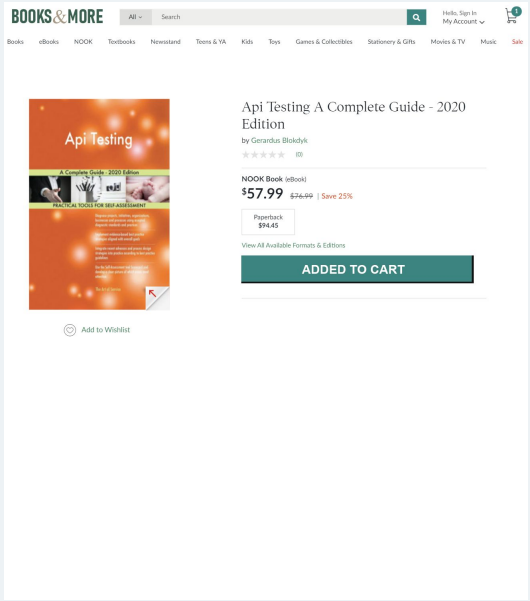
http://demoapi.apifortress.com/api/retail/product?q=red

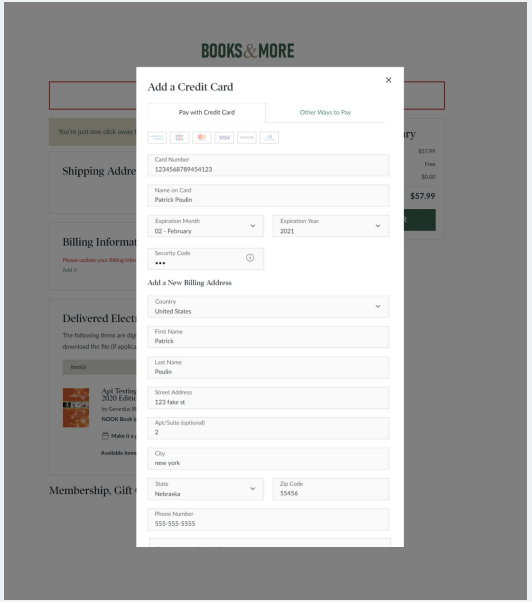
http://demoapi.apifortress.com/api/retail/product/3


```
[
        {
                quantity: 50,
                color: ["red","gray"],
                price: 49.99,
                imageURL: http://apif.com/earmuffs.jpg,
                name: "Earmuffs",
                description: "Keep those ears warm in the winter!",
                id: 3,
                category: "ears",
        },
]
```

So when I'm talking about an integration test, I'm talking about a single test that recreates a regular API consumer flow. Testing of the full flow should be a single test.

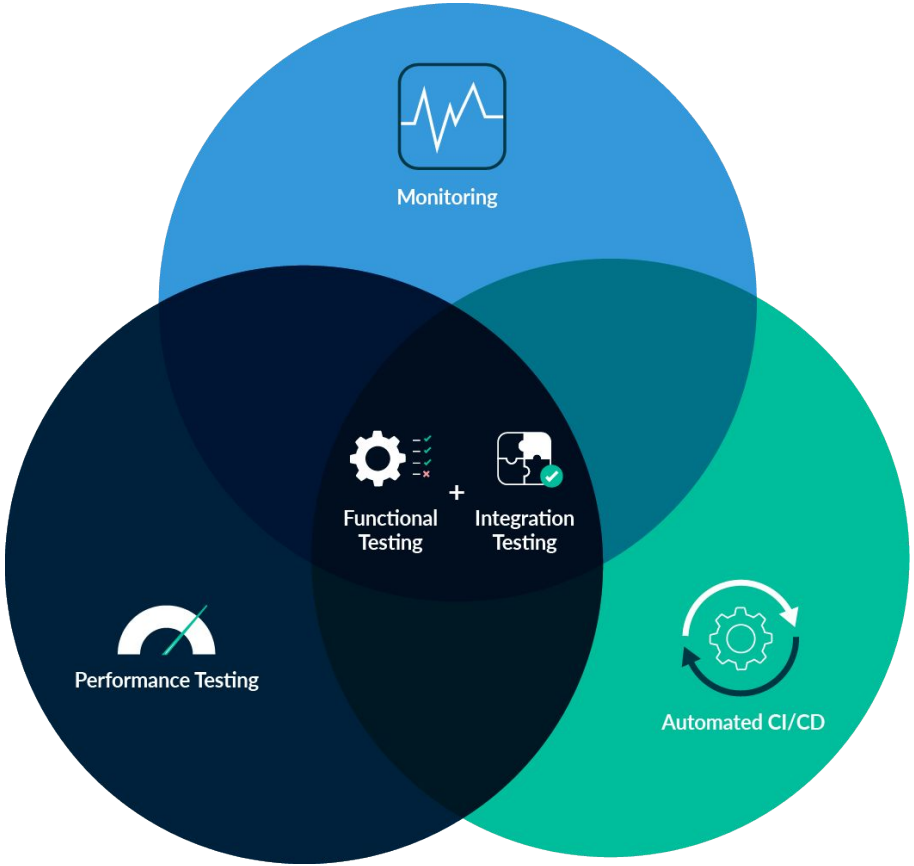






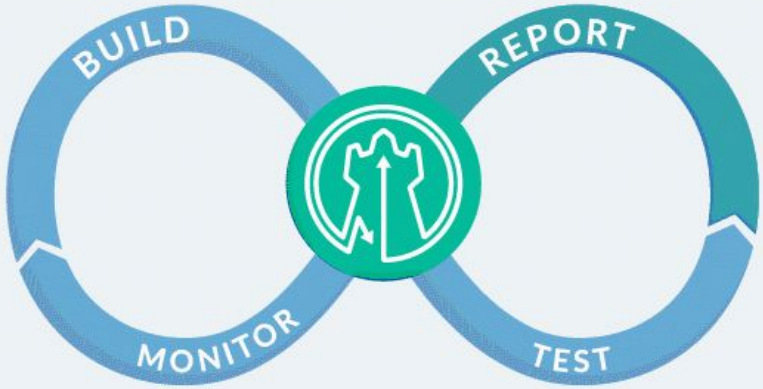
In Summary

Testing and monitoring are connected, and both require the same level of detail. At a minimum, monitors should, like proper end-to-end API tests, reproduce entire common consumer flows. This might seem obvious, but few companies implement this strategy. Many companies have completely separate functional testing teams, load testing teams, and monitoring teams. That might be okay for websites, but API quality can't be broken down that way. It's one thing - data - that needs to be tested in all three ways. A failure to recognize the difference in testing APIs results in poor quality, but also in vulnerabilities going live.

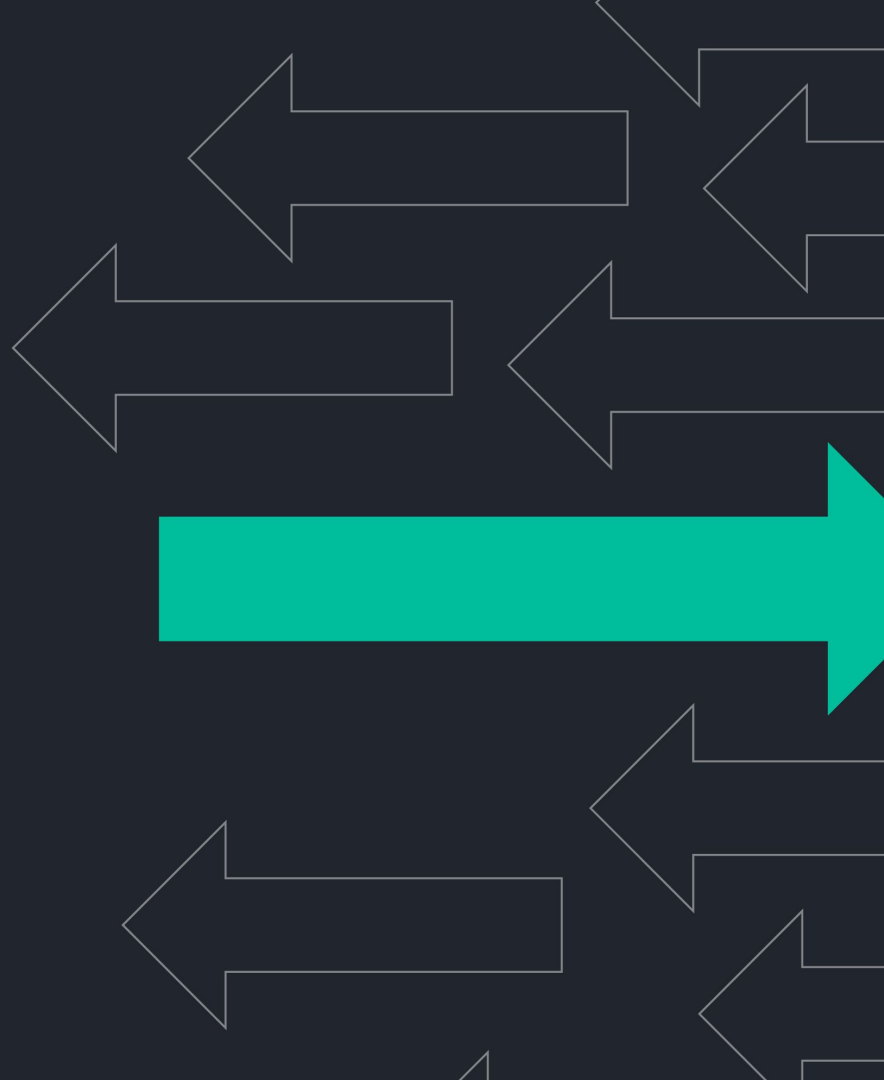


One Test Suite to Rule Them All

Use your functional and end-to-end tests as everything: from automated testing on releases to monitoring and performance testing. It all has to be combined into one, and work across all teams. Create a good process, standardize it across all teams, and deliver reliable APIs that your customers and partners will love to use.



Shift API Testing Right with Monitoring



The Internet runs on APIs, and most API failures are due simply to human error. That's so disappointing because of how easily many of these issues could be solved. One of the larger issues we see is that people use "uptime" but that can mean many different things at different levels of API quality ownership. So we prefer to call proper API uptime monitoring: "**Functional Uptime**."

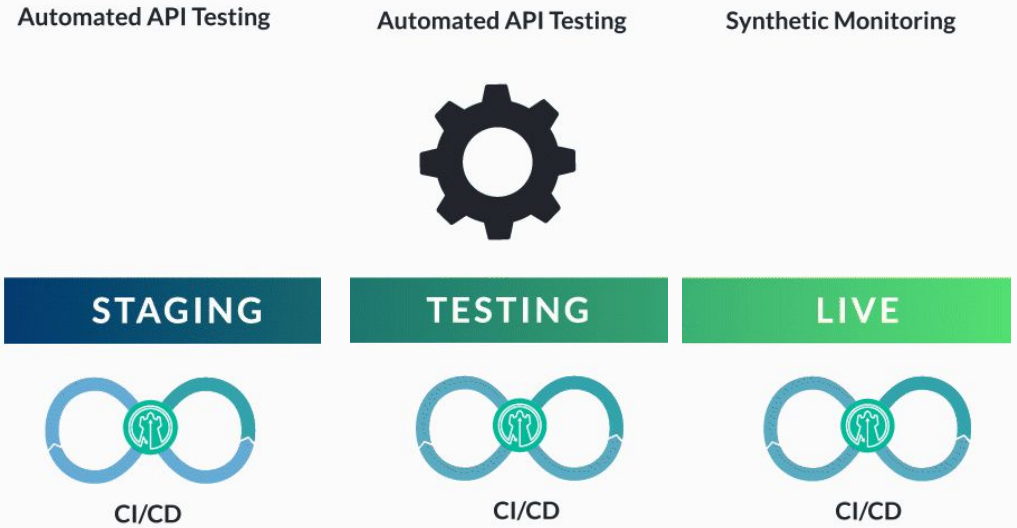
Akamai's State of the Internet report found "83% of all web traffic was in API traffic."

"Through 2022, at least 95% of cloud security failures will be the customer's fault."

Jay Heiser, VP at Gartner

The Market

We believe the gaps holding companies back from continuous quality largely stem from a lack of unified End-to-End API testing and monitoring tools in the market. So companies continue to stick to old ways of doing things: Strong API tests are run as part of an automated suite at every stage before production, but then when it goes live that usually ends. Suddenly, live monitoring is pushed to another product or team entirely - the quality gap persists on a continuous loop.



Test in Production!

If you agree that powerful end-to-end API tests are needed to deploy new APIs… why stop using them again in production?

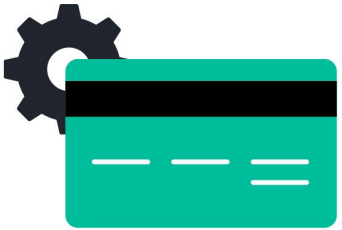
Production Testing = Monitoring
What's A Good API Monitor?

One of the larger issues we see is that people use "uptime" but can mean many different things, at different levels. So we prefer to call proper API uptime monitoring "Functional Uptime."

Many companies we work with set up monitors that simply make a call and look for a Status 200 OK. An API is very verbose and detailed. So a proper monitor should be analyzing the entire payload, every object, and the data associated.

One important difference between traditional and functional uptime monitors is in the data. In pre-production, you usually use a totally different set of data such as CSVs or stagnant data. That's not how production environments work: they require fresh and dynamic data.

While you learn about consumer flows in production, why only use tests that reproduce the flows in staging? Consider the many variables that change in production with today's services and apps, and the advantages of continuous shift-left and shift-right testing (monitoring) become clear.



What's a Good API?

Passes Automated Tests
99.8% Functional Uptime

Functional Errors & Uncaught Downtime

An API that is “up” but functionally down is the most expensive problem a company can deal with: a false-positive. It may cause you to lose customers, reputation, and revenues, as well as potentially expose sensitive data for months to cybercriminals.

How Is This Happening?

Who Owns API Functional Uptime?

One of the key reasons that so many API defects and vulnerabilities slip past today's monitoring platforms is because these platforms are flawed at their very foundation when it comes to monitoring ***APIs***. Most QA, architecture, DevOps, and product leaders can't escape the assumption that uptime is synonymous with monitoring. Consequently, no stakeholder with significant API domain knowledge seems to own an API's uptime.

There are many popular monitoring and analytics platforms today. Most claim to extend monitoring coverage to API monitoring. But these platforms all suffer from critical shortcomings.

Monitoring Platform Shortcomings:

1. "Synthetic Testing"

A euphemism for minimal API testing capabilities

2 User Profile

People in charge of the monitors aren't testing professionals.

3. Using a Different Platform

Tests are written outside of QA by stakeholders with little/no domain knowledge

4. Test Intelligence

Tests must capture real consumer flows or user behavior

Problem in a Nutshell

An API that is “up” but functionally down is the most expensive problem a company can deal with: a false-positive. It may cause you to lose customers, reputation, and revenues, as well as potentially expose sensitive data for months to cybercriminals.

Functional API Monitoring

Automated Testing Team

Monitoring Team

Let Testing Experts Monitor APIs

REDUCE RISK

- They are specialists with domain knowledge
- Catch a wider range of API bugs and vulnerabilities
- 95% of API breaches caused by human error
- Catch and fix functional errors before going live with no rollbacks

INCREASE OPS EFFICIENCY

- Increase regression testing without increasing IT resources
- Accelerate time to market
- Improve cross-team collaboration

Evaluation Checklist:

http://bit.ly/apievalchecklist

Thank You!

Questions: Will@APIFortress.com

Chat / Free Trials: API Fortress.com

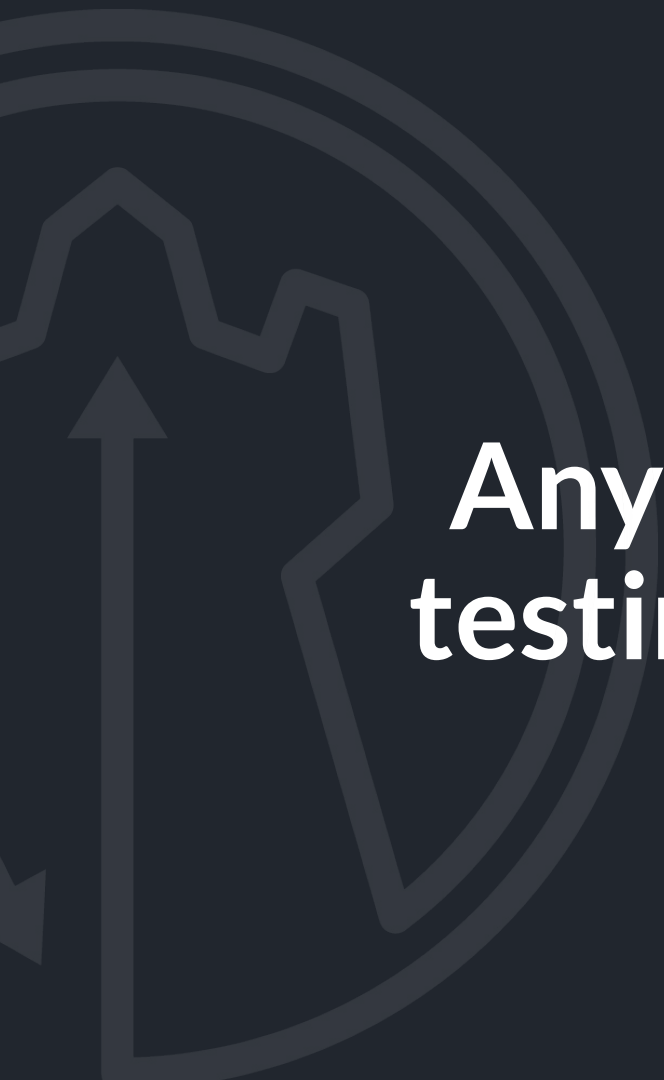
FAQ

See if you can answer these FAQ with what you've learned

What if the data API is behind a firewall?

How do I convince management to run tests on production?

What if we don't have a spec?



Any suggestions on testing status codes?

Do you have some tricks for testing the security of an API?

Can 3-legged OAuth be tested automatically?