Feb 20 · 7 min read

API monitoring with New Relic and ELK stack

If you ever had to deal with supporting partners using your API, you know just how difficult it might be to track down a bug. The client makes a complaint, you try to replicate the situation and… it works just fine. You need more details so the e-mail thread continues, time passes, the client gets frustrated and so on and so forth.

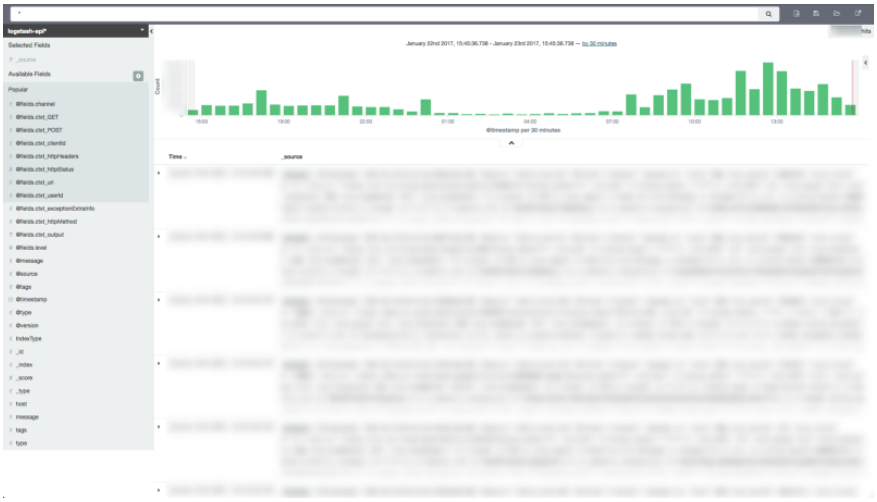
Have you ever tried to find a specific problematic client request among millions of other API calls?

We had this problem at OLX EU and we looked for a solution tailored to our needs. It had to be configurable, hosted locally and it had to be free.

ELK Stack

The ELK acronym stands for Elasticsearch, Logstash and Kibana.

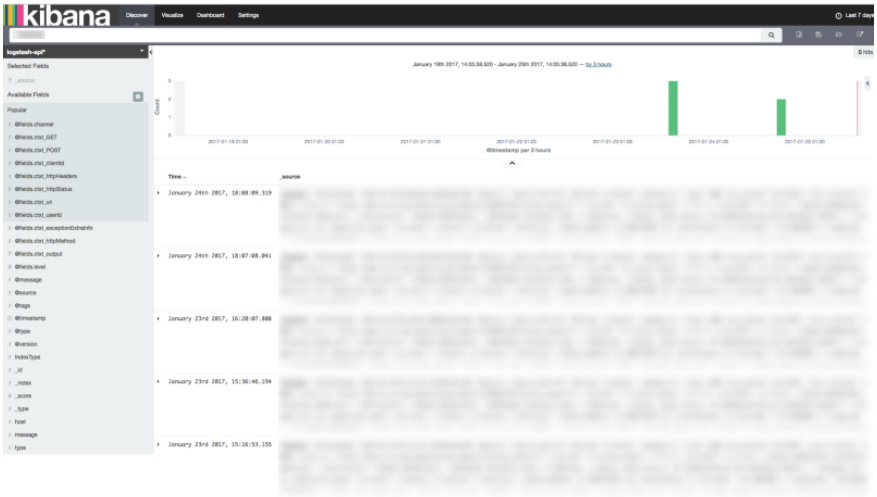
What we basically do is that we log all information pertaining to the customer's request. And I do mean ALL information—the URL that clients use, the sent payload and the API request output (for unsuccessful requests).



Kibana: overall view

What are the perks of this solution? You see exactly what your API client sent your way in just a few seconds. No need for lengthy e-mails and countless questions, and it all happens instantly!

Just select the correct date range and use any unique identifier (at OLX we use either the internal advert id or the client's id). Everything you might need shows up in a matter of seconds.



Kibana: "show me requests for this certain advert"

Obviously Kibana can do a lot more than that, everything depends on your specific requirements. You need charts based on the logged requests? Simply go to *"Visualise"* section and visualize what you want!

So, how do you set up your ELK stack?

I will focus solely on the application side of the configuration. For this purpose let us assume that there is an already running machine with ELK.

First of all you have to configure Logstash so that it takes in incoming messages and passes them to ElasticSearch. The application file could look as follows:

```
input {
  udp {
    port => 5000
    type => "api-requests"
  }
}
filter {
    if [type] == "api-requests" {
        mutate {
           add_field => { "IndexType" => "api_requests" }
        }
        json {
           "source" => "message"
        }
    }
}
```

Logstash is told to look out for incoming connections on UDP port 5000 and to add the field “type” with the “api-requests” value to each request. The following section specifies ElasticSearch index prefix and states that the message is in JSON format.

We use UDP to log data to Logstash as we want to avoid any issues related to logging. In general Logstash is a fast and stable service but we don’t want our users to experience any delays or see error pages in case of network issues or crashes of the logging machine. This makes our solution reliable—we have logs which are entirely transparent for users.

Part two—application

We use Monolog with a ready-made plugin for Logstash. The only thing we had to add was a simple custom Monolog Handler that connects to Logstash and push data.

Besides that, you obviously need some logic in order to prepare an event to log. We decided to log everything—GET and POST data, headers and all data relating to the API clients. Sensitive data such as access tokens and passwords was the only thing that was excluded from logging.

There is one catch: the UDP protocol has a limit of 65kB of data, which means that you need to handle this manually (truncate the data or even skip such a record) or otherwise language errors will occur.

The extensive logging provides us with a lot of data—a couple of gigabytes of data in ES daily. There is a script cleaning ES from indexes older than 14 days that uses the simple ES REST API.

And that’s it! It is that simple to get a tool that is easy to use and enables you to browse through almost all API requests. It is a great help in everyday work.

New Relic

When will you know that API integration has failed? Waiting for the client to call is one way to go about it, but what if you want to be proactive?

Here enters the New Relic.
(If you never have seen NewRelic in action, I strongly recommend to take a quick tour here).

I will skip basic installation and configuration, as there are many sources that will explain how to do it. Let's focus on the application side.

First of all we had to separate the API part of the traffic from the rest of the application. Even if API is still a part of the application the statistics need to be separated. We were able to achieve this extraction by explicitly calling newrelic module functions in the code responsible for API.

```
newrelic_set_appname('OLX API xxx')
```

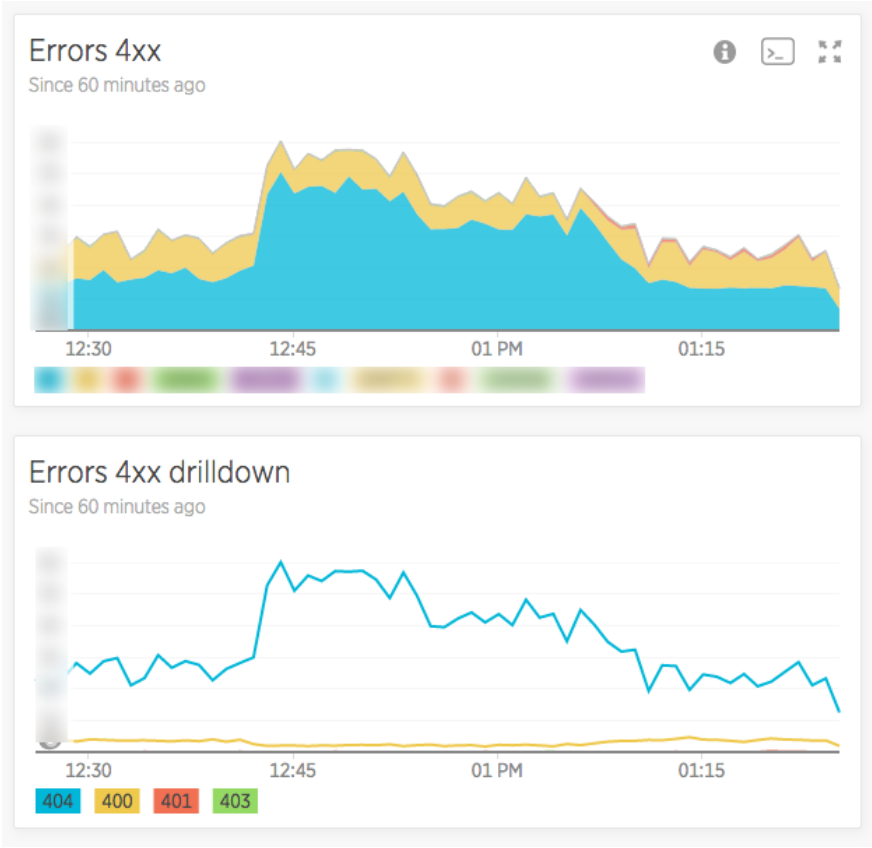
Once it's been released, all of the API requests started to accumulate in the application so we experienced traffic, performance and error analysis (together with the remaining NewRelic features).

However this only covers the technical side of monitoring, while we still need more business monitoring.

New Relic Insights

A couple of months ago New Relic has introduced "New Relic Insights" —a powerful data analysis tool based on the data gathered by Newrelic agents. Even with a new tool there already is a lot of data to be analyzed.

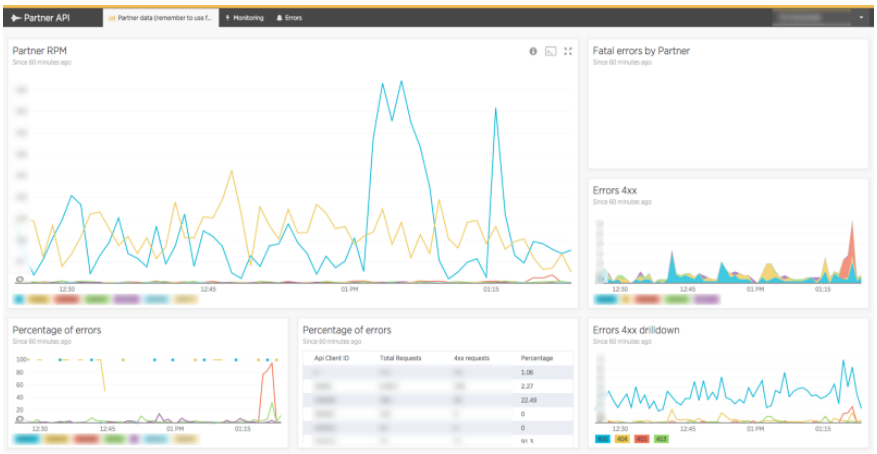
For that reason we created a dashboard monitoring traffic of errors separated by HTTP code—4xx errors have a different meaning than 5xx, don't they?



New Relic insights: on top Errors by partner, bottom: errors by type

But… what if you want to monitor one particular API client?

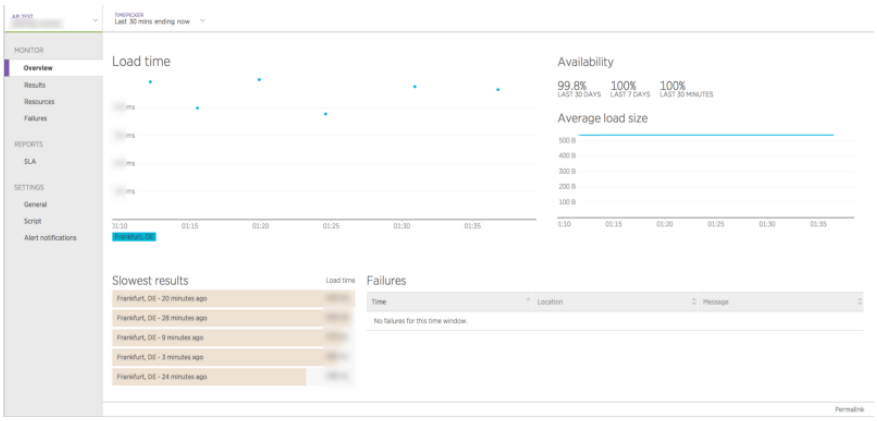
NR allows you to tailor requests (transactions) with custom parameters. As all of our endpoints require OAuth authorization, we added the information about the client id from the token to NewRelic. It was that easy—with a single parameter we gained access to “personalized” statistics of requests. The way you will use the data is entirely up to you and your needs.



New Relic Insights: one of the dashboards

New Relic Synthetics

Alright, all of the above pertains to passive monitoring. But let's do some active tests. Let's assume you want to be sure that your API main entry point (OAuth authorization) works fine—if the client provides correct credentials, he/she will receive a valid access token. This is the point where "Synthetics" comes to the rescue.



Results of Synthetics monitor

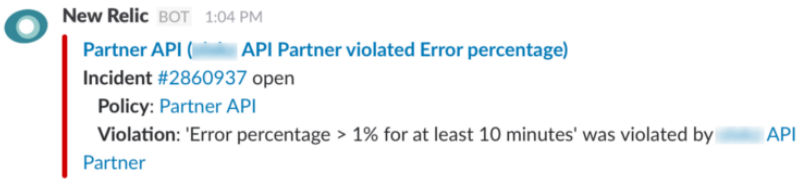
In simple words, it's a kind of scripted monitoring. You write a script and assertions, pick locations across the world that you want to run it from and… wait. That's it.

In our case, I wrote a simple script that uses OAuth authorization endpoint and checks results. If a token is valid—it's a pass. If not—error is triggered and NR sends a notification to Slack.

That is all. Myself, I used it out of curiosity. We don't write any more scripts here because we have other types of tests. However, it might be an interesting option for some of you.

New Relic Alerts

With this kind of metrics it's easy to create a custom check. The error rate is too high? Not a problem, let NewRelic send a Slack notification.



OAuth authorisation returns 503 instead of tokens? Send a notification, let the people see that there is something wrong.

Summary

The use of ELK stack allows us to quickly respond to issues reported to us by consumers. Now, it is enough to copy the unique advert ID (or any other type of unique data the client provides you with) and paste it to Kibana in order to get the full history of requests regarding the specific item in a blink of an eye.

Additionally, New Relic allows us to take a look at API as a whole. You can see the detailed performance, stability and errors. Depending on your needs you can also access other data. You decide what to do with New Relic.

This is the very beginning of our journey of monitoring with NewRelic and there are still many ideas to be tested. However, I am confident saying that **API debugging has never been easier than it is now** :-)