

# BLACKFIRE & KITCHENRESTOCK

ANTICIPATING PROBLEMS BEFORE THEY OCCUR



blackfire.io

## INTRODUCTION

### 121ECOMMERCE AND KITCHENRESTOCK – ENSURING A SEAMLESS BUYER EXPERIENCE

For nearly a decade, 121eCommerce has helped hundreds of businesses flourish in eCommerce. Our focus is to provide Magento expertise coupled with constant communication and transparent reporting that has earned us a stellar reputation and proven track record.

This case study will cover how our developers used Blackfire while working with one of our clients – KitchenRestock. With customer satisfaction as the end goal of their operation, KitchenRestock has solidified its leadership position in providing quality restaurant equipment across North America.

In a market where 75% of consumers abandon their cart before completing checkout, creating a seamless buying experience is paramount. If a web page takes more than 3 seconds to load, it will lose more than 50% of its traffic. What's more, malfunctions in the shopping cart account for 27% of cart abandonment rates.

To keep page load times in check and the shopping cart operational, our development team uses Blackfire in tandem with New Relic. With New Relic, our team already had an efficient way to detect performance issues. However, they required more concrete information in emergency cases. They knew where the performance issue came from, but needed to answer why.

That's where Blackfire came in.

## WHY BLACKFIRE?

Blackfire is a Performance Management Solution geared towards optimizing an application's performance at every step of its life cycle: Development, QA, staging, and production. Blackfire helps our team identify performance bottlenecks in our codebase and why they happened.

Blackfire suits our development team for the following reasons:

- **Hassle-free testing and benchmarking using YAML.** Blackfire helps our development team define performance expectations for how our code should behave. Configuration is simple using the YAML syntax, and supports a variety of assertions such as load speed, memory consumption, and network output.
- **Clear UI.** With Blackfire, our team can easily visualize a profile either in graph mode or list mode. The call graph view affords the team high-level oversight of the interrelations between processes, while the sidebar view allows for thorough analysis of each function call.
- **View resource consumption at every step of execution.** A key part of reducing page load times is reducing resource consumption. Blackfire.io provides a breakdown of all resources utilized by every instruction in the codebase, including memory, network, IO and database resources.
- **No extra overhead.** We use New Relic to monitor client applications in real-time. To do this, it has to keep instrumentation at a bare minimum, which compromises data breadth. By focusing exclusively on profiling requests, Blackfire offers more specific information without compromising the production environment.



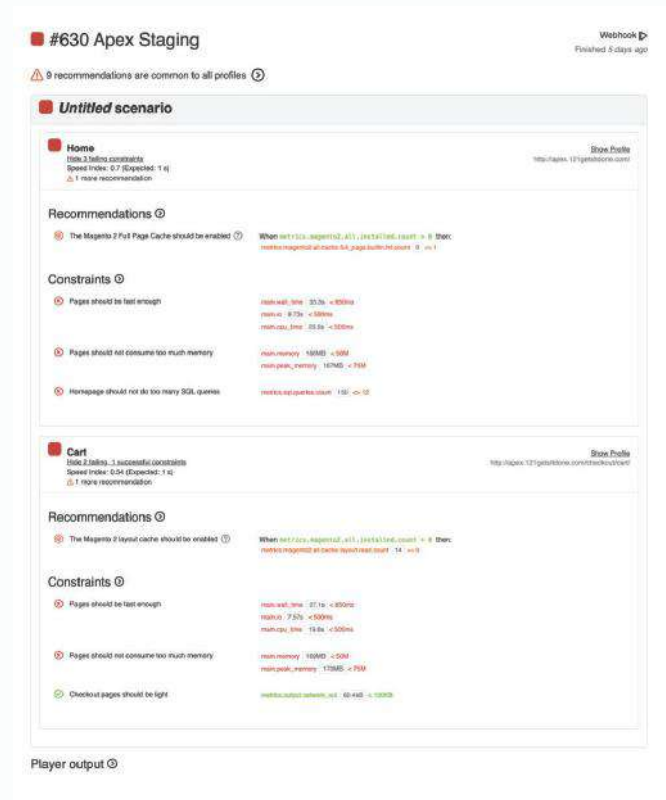
# INTEGRATING BLACKFIRE INTO OUR WORKFLOW

## SETUP

For our needs, a default setup of Blackfire was sufficient. In our development environment server, the team installed and configured the blackfire-agent file with the minimum required parameters such as Server IDs, user tokens and Log Files.

The blackfire.yaml file is configured with the team's desired performance benchmarks, which mainly relate to a client site's catalog and checkout page. We can then see these assertions enforced in the blackfire.io interface:

```
1 tests:
2   Pages should be fast enough:
3     path: /*
4     assertions:
5       - main.wall_time < 850ms
6       - main.io < 500ms
7       - main.cpu_time < 500ms
8   Pages should not consume too much memory:
9     path: /*
10    assertions:
11      - main.memory < 50M
12      - main.peak_memory < 75M
13   Homepage should not do too many SQL queries:
14     path: /
15     assertions:
16       - metrics.sql.queries.count <= 12
17   Checkout pages should be light:
18     path: /checkout/*
19     assertions:
20       - metrics.output.network_out < 100KB
21
22 scenarios:
23   Home:
24     - /
25
26   Cart:
27     - /checkout/cart/
```



## WORKFLOW

Our development team is geographically distributed between the Americas and Europe, collaborating using Git and GitHub repositories. When they need to deploy, they use a variety of unix-based (Debian, Ubuntu) servers that cater to different environments -- staging, development, and production. Our team adheres to TDD principles, and all code is first approved by the lead developer before it passes into production.

Our team uses New Relic as an APM to monitor performance in real-time. Although New Relic is sufficient for them to resolve most performance bottlenecks, some issues necessitate a more thorough examination. These are issues with a client's shopping cart slowing down dramatically, which impacts user experience. In these emergency situations, Blackfire comes into play.

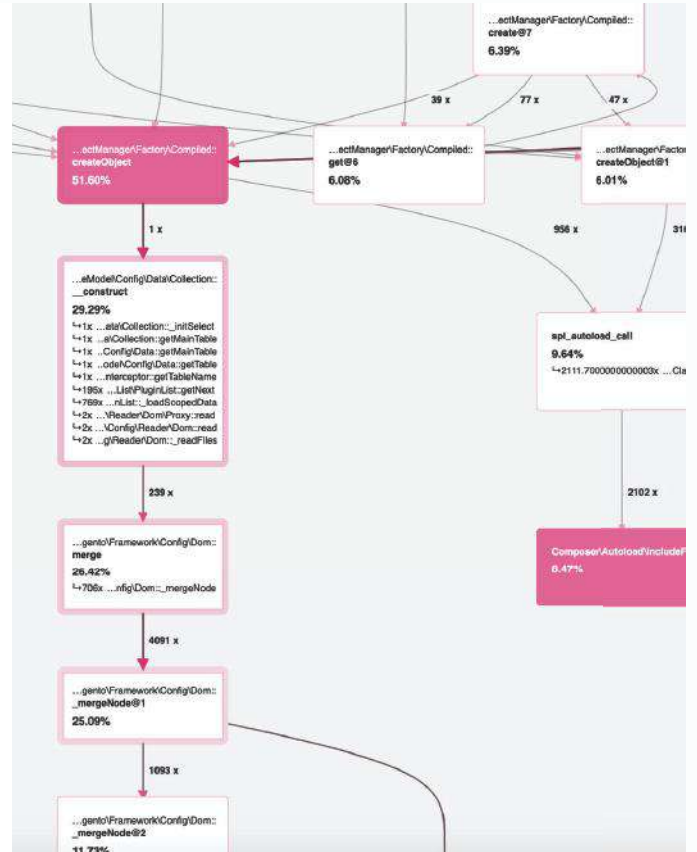
While New Relic does an excellent job at pinpointing where a bottleneck happened, Blackfire helps our team answer why the bottleneck occurred in the first place. Using Blackfire, our developers can navigate to the exact point where New Relic detected an issue, and see the processes that happened before and afterwards.

What's more, the exact consumption of computing resources for each process is shown. This allows our team to quickly diagnose whether the bottleneck is due to a network or database issue. This determines where to start resolving the bottleneck, which helps our team better estimate solution requirements.

Our team can easily share profile graphs with anyone. When a new developer joins our team, they're sent the URL of the Blackfire profile, which is enough for them to get started working.

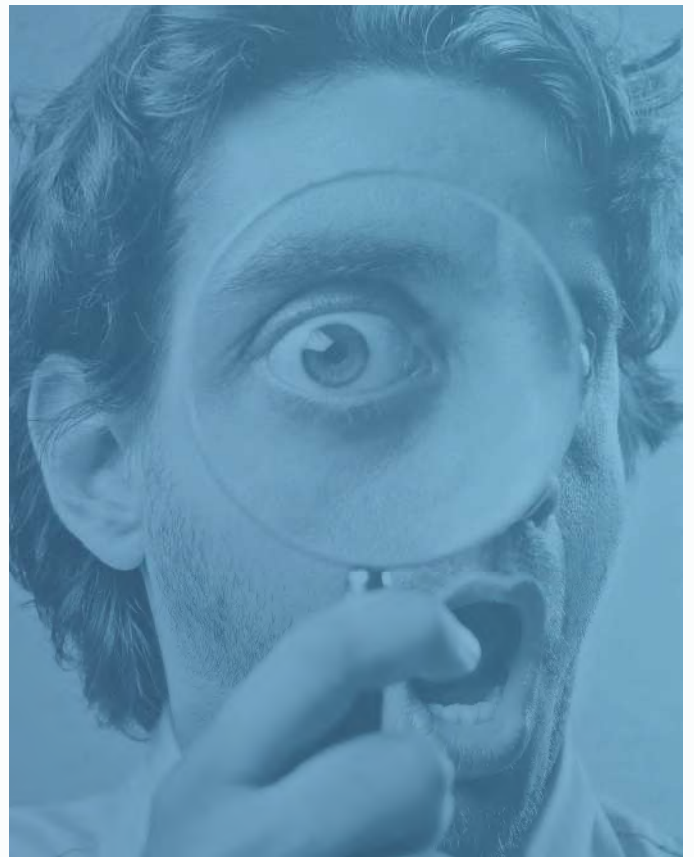
## MANAGING CHANGING REQUIREMENTS USING BLACKFIRE

There are many stakeholders involved in our business, both from our side and our clients' side. Multiple stakeholder involvement means the requirements for a client's project can change over time. Blackfire helps the development team implement requested project changes without impacting production code. Utilizing Blackfire's Call Graph view, our developers can detect and isolate performance issues in our development servers, which mitigates problems in production-level code.



KitchenRestock requested our team to extend the number of shopping cart checkout steps from 2 to 5. KitchenRestock has a catalog of more than 250,000 products and handles thousands of daily transactions. For such a large operation, a performance issue in the buyer experience could spell significant financial losses.

While implementing the requested changes, the team used Blackfire to establish expected performance thresholds for each new step by making changes to the `blackfire.yaml` file. They then began developing each new functionality while simultaneously testing/profiling -- all in an isolated development server and without impacting the production servers. As a result, the team was able to comply with KitchenRestock's request without impacting the user experience.

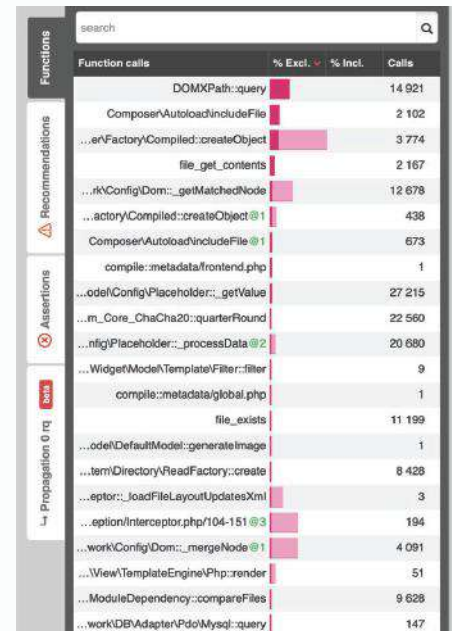


## EASING THE MIGRATION TO MAGENTO 2

For our clients, the pressures to migrate from Magento 1 to Magento 2 are twofold. On the one hand, support for Magento 1 will end in June of 2020. This will leave Magento 1 patchless, bringing a swathe of security vulnerabilities. On the other hand, Magento 2 introduces a series of optimizations – such as streamlined checkout and elastic search – that will increase ease of purchase and overall site speed.

However, migrations are as complex as they are lengthy. Migrating to Magento 2 requires our development team to rewrite all custom modules from Magento 1. During this process, our team frequently uses Blackfire to screen for possible performance issues. They specifically look for suspicious, albeit not erroneous, misuse of computing resources for every new custom module build.

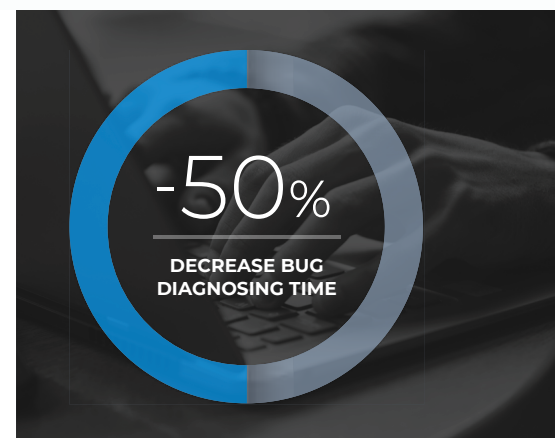
In this case, Blackfire's sidebar view and resource breakdown functionality proves critical. The sidebar view allows our team to detect function calls that are consuming more resources than normal. By using the breakdown functionality, developers can see whether the excess resource consumption was in memory, network, IO or database.



Function calls	% Excl.	% Incl.	Calls
DOMXPath::query			14 921
ComposerAutoloadIncludeFile			2 102
...enFactoryCompiled::createObject			3 774
file_get_contents			2 167
...rkConfig\Dom...getMatchedNode			12 678
...actoryCompiled::createObject			438
ComposerAutoloadIncludeFile			673
compile::metadata/frontend.php			1
...odel\Config\Placeholder::getValue			27 215
...m_Core_ChaCha20::quarterRound			22 560
...nfig\Placeholder::processData			20 680
WidgetModelTemplateFilter::filter			9
compile::metadata/global.php			1
file_exists			11 199
...odelDefaultModel::generateImage			1
...tem\DirectoryReadFactory::create			8 428
...ptor::loadFileLayoutUpdatesXml			3
...option\Interceptor.php/104-151			194
...work\Config\Dom...mergeNode			4 091
...ViewTemplateEnginePhp::render			51
ModuleDependency::compareFiles			9 628
work\DBAdapter\PdoMysql::query			147



In the event of a performance issue reaching production code, using Blackfire has cut down on bug diagnosing time by 50%. When a performance issue arises, 70% of the time is dedicated to diagnosing the cause, and 30% to resolving the issue. A 50% drop in diagnosing not only saves our clients' money, it also allows our developers to focus on more productive endeavors for our business.



## AUTOMATING PROFILE GENERATION

Our team doesn't make use of any custom metrics or integrations. However, they've leveraged the Blackfire REST API to help generate automatic profiles after each code deploy.

```
sh "curl -X POST --user \"a4bb6d6aa5c0efee:5da2f99bc306a025\" https://blackfire.io/api/v2/builds/env/979f80c6-752a-445d-83a1-33525fd98411/webhook -d \"endpoint=${CHECK_URL}\" -d \"title=${JOB_NAME}\" |  
python -c \"import json,sys;obj=json.load(sys.stdin);print obj['_links']['report']['href'];\" > build/logs/blackfire.log"
```

Every time code is deployed to one of the development servers, a cURL command accesses the Blackfire API by passing the URL and user tokens as parameters. In this specific command, a blackfire.log file is automatically created which contains the full profile for that particular deployment.

## CONCLUSIONS

In summary, Blackfire allows our development team to:

- Detect performance issues before they make it to production.
- Effectively respond to changing project requirements and increased complexity.
- Visualize resource consumption at the function call level to determine an issue's root cause.

## NEXT STEPS

Our team could definitely see themselves making use of the Blackfire/New Relic integration in the future. The core of application monitoring is done by New Relic, which is sufficient for now. However, as our team becomes increasingly reliant on Blackfire to provide critical information under critical situations, having a more coupled Blackfire integration would be a logical next step.

