DZone

THE 2018 DZONE GUIDE TO

Performance

TESTING AND TUNING

VOLUME IV

BROUGHT TO YOU IN PARTNERSHIP WITH

catchpoint™ SENTRY (x) matters®

Dear Reader,

For those of you who know how tough it is hunting down the cause of a spike that shouldn't be there, this guide is for you. I've worked many years with performance tuning in environments like Big Data and NoSQL, and at a much lower level than messing with the XMeters or XLogics of the universe. Albeit, most of the fun is in the upper stack, as it makes even the smallest anomaly a challenge.

Tell me if this sounds familiar: the team tells the product manager they need to push the release. When the PM asks "why," they explain that they found a performance bug that requires an architectural change. Of course, the PM asks, "And we didn't see this coming?" and the team can only respond with silence.

Been there, done that! And that, my friends, is the abyss called non-functional requirements, which is where the spotlight is pointing. The industry rode the DevOps train hard and now we are getting some attention. Organizations are finally understanding that in order to increase profits they must look at not only the data, but at the entire system. Ultimately, they must back the culture to survive.

There are many good solutions out there and fewer still that are great. This is where AI is going to be the differentiator. The advent of machine learning is cool, but the use of it in performance tuning and testing is awesome. And let's not forget about quality and high availability. So, monitoring these hybrid environments will be essential.

Another big change in development is making performance a priority in the SDLC. Gone are the days of waiting until the 11th hour to assess performance. It must be architected such that it is assessed early in development, in a way that can scale and be sustained over the maintenance cycles. The number of high profile outages and breaches over the years cements the requirement for performance to be part of the "definition of done."

There is also a shift in quality engineering such that it's almost like DevOps and QA will eventually merge into a seamless entity. This will open the door for enterprises to come to grips with the world of performance engineering and provide a path to a deeper understanding of their solutions architecture as compared to their vision.

This guide will touch on several ideas ranging from root cause analysis for escalations to the history of performance engineering, and more. Whether you're just reading to pass time or looking to step up your game, you'll find plenty of perspective to help guide you through the maze of performance.

By Rick Garee
SR. QA ENGINEER, DZONE

DZONE IS...

PRODUCTION
CHRIS SMITH
DIR. OF PRODUCTION

ANDRE POWELL
SR. PRODUCTION COORD.

G. RYAN SPAIN
PRODUCTION COORD.

ASHLEY SLATE
DESIGN DIR.

BILLY DAVIS
PRODUCTION ASSISSTANT

MARKETING
LAUREN CURATOLA
MARKETING SPECIALIST

KRISTEN PAGÀN
MARKETING SPECIALIST

JULIAN MORRIS
MARKETING ASSOCIATE

SALES
MATT O'BRIAN
DIR. OF SALES

CHRIS BRUMFIELD
SALES MANAGER

FERAS ABDEL
SALES MANAGER

ALEX CRAFTS
DIR. OF MAJOR ACCOUNTS

JIM HOWARD
SR. ACCOUNT EXECUTIVE

JIM DYER
SR. ACCOUNT EXECUTIVE

ANDREW BARKER
SR. ACCOUNT EXECUTIVE

BRIAN ANDERSON
ACCOUNT EXECUTIVE

RYAN McCOOK
ACCOUNT EXECUTIVE

TOM MARTIN
SR. ACCOUNT MGR.

JASON BUDDAY
ACCOUNT MGR.

MICHAELA LICARI
ACCOUNT MGR.

BUSINESS AND PRODUCT
RICK ROSS, CEO

MATT SCHMIDT, PRESIDENT

JESSE DAVIS, EVP, TECHNOLOGY

KELLET ATKINSON
MEDIA PRODUCT MANAGER

EDITORIAL
CAITLIN CANDELMO
DIR. OF CONTENT & COMMUNITY

MATT WERNER
PUBLICATIONS COORD.

SARAH DAVIS
PUBLICATIONS ASSOCIATE

MICHAEL THARRINGTON
CONTENT & COMMUNITY MGR. II

KARA PHELPS
CONTENT & COMMUNITY MGR.

TOM SMITH
RESEARCH ANALYST

MIKE GATES
SR. CONTENT COORD.

JORDAN BAKER
CONTENT COORD.

ANNE MARIE GLEN
CONTENT COORD.

ANDRE LEE-MOYE
CONTENT COORD.

Executive Summary

BY MATT WERNER
PUBLICATIONS COORDINATOR, DZONE

INTRO

Performance issues are an everyday, unpleasant part of software development. They threaten to scare users away, lose money, and eat up time that could otherwise be devoted to new products or new functionality. But exactly what issues give developers the most trouble, and how are developers fixing them? To find out, we asked 473 DZone members to answer questions about their most frequent problems, tools of choice, and how they approached software development.

DEVELOPERS KEEPING TO THE RIGHT

Data: 56% of respondents build application functionality before taking performance into account, while 44% build with performance in mind. However, 85% of DZone users reported that they are encouraged to build performance into their applications from the start.

Implications: Organizations are encouraging developers to think about performance, but most developers are unwilling or unable to do so.

Recommendations: The wide disparity in what developers are encouraged to do and what they actually do may suggest an issue with company culture and how management instructs developers to behave in time-sensitive circumstances, such as an upcoming release or crucial bug fix. If companies are practicing a mix of agile and waterfall practices, and still insist on hard release dates, then developers may not be incentivized to think about performance and instead just focus on the application's functionality to get the product out the door. Make sure that developers have consistent instructions that don't get thrown out when the pressure is on.

SHIFTING LEFT IS THE REAL DEAL

Data: 61% of DZone users who experience frequent issues with application code and 73% who experience frequent database-related issues focus on application functionality first before performance. 52% of members who focus on performance during development solve issues in 10 hours or less, on average, compared to 38% of members who focus on functionality.

Implications: Shifting left does create real long-term benefit for development teams and their applications. The less time spent maintaining and repairing current software means there's more time to build new applications or add new functionalities.

Recommendations: It's clear that shifting performance to the left is beneficial in reducing the number of issues developers have to deal with. As mentioned in the previous section, organizations have to make sure developers are hearing a consistent message, i.e. that focusing on performance should not be abandoned even if there's pressure to release a product by a certain date. In addition, developers should be properly educated on the benefits of shifting performance to the left by checking research and pointing out the time that will be saved down the line, and stress that they can spend more time creating software and less time fixing bugs.

NAGIOS AND LOGSTASH LEAD THE PACK

Data: Usage of most monitoring tools either fell or remained steady across the board. The top four tools were Nagios (33% in 2017 vs. 25% in 2018), LogStash (27% vs. 25%), Splunk (25% vs. 20%), and Amazon CloudWatch (21% vs. 22%).

Implications: The decrease in the use of monitoring tools is related to developer habits. In 2017, 47% of DZone members used 3-5 monitoring tools, compared to 42% in 2018. However, in 2018, 32% of members reported using only 1-2 monitoring tools, vs. 15% in 2017.

Recommendations: Developers are using fewer monitoring tools in general, likely to simplify their processes by only focusing on a few tools rather than four or five. While there's certainly value in doing so, make sure that a minimalist setup isn't missing any data that would prove valuable to the organization. If you need to make a decision based on data you don't have, you'll waste a lot of time implementing a solution and collecting that data at a later date.

Key Research Findings

BY **JORDAN BAKER**, CONTENT COORDINATOR, DZONE

DEMOGRAPHICS

473 software professionals completed DZone's 2018 Performance survey. Respondent demographics are as follows:

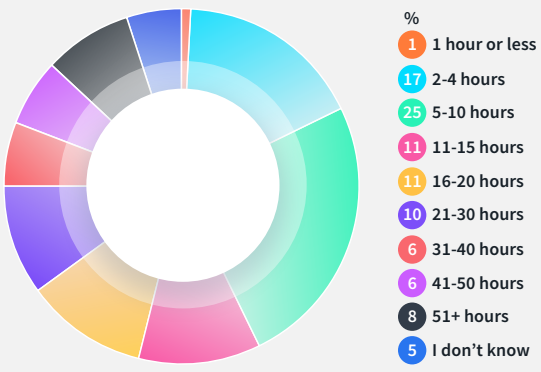
- 38% of respondents identify as developers or engineers, 22% identify as developer team leads, and 15% identify as software architects.
- The average respondent has 14 years of experience as an IT professional. 55% of respondents have 10 years of experience or more; 18% have 20 years or more.
- 33% of respondents work at companies headquartered in Europe; 36% work in companies headquartered in North America.
- 20% of respondents work at organizations with more than 10,000 employees; 17% work at organizations between 1,000 and 10,000 employees; and 25% work at organizations between 100 and 1,000 employees.
- 79% develop web applications or services; 47% develop enterprise business apps; and 22% develop native mobile applications.
- 84% work at companies using the Java ecosystem; 67% at companies that use client-side JavaScript; 36% at companies that use server-side JavaScript; and 34% at companies that use Python. 63% of respondents use Java as their primary language at work.

FREQUENCY OF BUGS

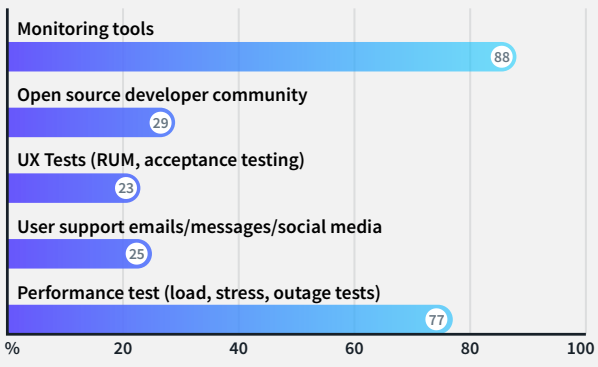
One of the biggest shifts in this year's survey was the number of respondents reporting that they needed to solve performance issues on a weekly basis. When asked, "When was the last time you had to solve a performance problem in your software?", 36% of those who took the survey responded with "this week." That is up from 28% in 2017's Performance and Monitoring Survey, and 26% from 2016. Conversely, the percentage of those who told us that their application code poses frequent performance issues (the number one issue reported in last year's survey) decreased from 36% in 2017 to 30% in 2018. Despite this reduction, however, application code still poses the biggest performance issue, by far, for our community.

The second most frequent performance issues occur in the database, with 21% stating their database often gives them performance problems (though this number is down from 24% in 2017). Not only were database issues voted the second most frequent,

GRAPH 01. How long does it take your team to solve a performance problem?



GRAPH 02. In your experience, which methods are the best ways of discovering performance issues?



they were also voted as the most challenging issue to fix (of those that can be fixed). 49% of respondents told us that database issues are the most challenging, while 46% stated that application code problems are the most difficult.

FUNCTIONALITY BEFORE PERFORMANCE

Despite the pervasive nature of the shift left movement in the software development industry, a majority of respondents still claim to worry about functionality before performance, rather than building performance into their app/software from the beginning of the SDLC. Indeed, over the past three years, this trend of building functionally before considering performance does not seem to have ebbed in the developer community. In our 2016 Performance and Monitoring survey, 56% respondents told us they build functionality then consider performance issues; in 2017, this number was 55%; in this year's survey, the same percentage of respondents as 2017 claimed to code for functionality prior to performance.

If we compare these numbers to the frequency of performance issues above, we find, unsurprisingly, that more problems arise when performance is not coded from the start. Of those who told us that application code causes frequent issues, 61% said that they build their application and then worry about performance. And, of those who reported to have frequent database-related performance issues, 73% said they worry about functionality first, then worry about performance. Finally, 52% of respondents who build performance in their application from the beginning of development solve performance issues in 10 hours or less, on average; this drops to 38% for respondents who worry about performance after functionality is built.

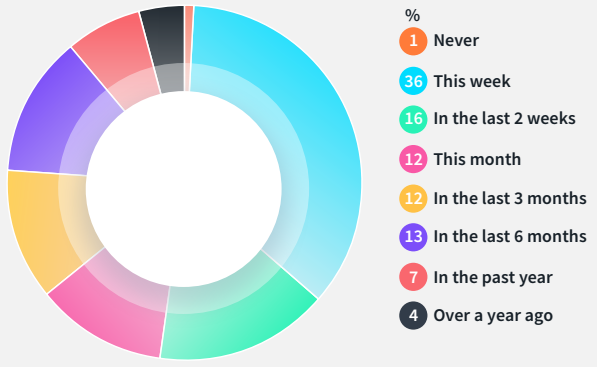
MOST POPULAR MONITORING TOOLS

When we asked community members to tell us their favorite monitoring tools, an interesting trend appeared. While the top four most popular tools remained the same as in our 2017 survey, none of these tools rates of use have grown by a significant margin over the past year. Here's a list of the top four tools reported in the 2017 survey, with the percentage of respondents who used them: Nagios (33%); LogStash (27%); Amazon CloudWatch (21%); Splunk (25%). In this year's survey, 25% of respondents stated they use Nagios and LogStash (which constituted the two most popular options), 20% reported using Splunk, and Amazon CloudWatch stayed relatively steady at 22%. While there were some increases in the percent of developers using some of the smaller tools, (jClarify, for example, rose from 3% in 2017 to 5% in 2018), the major reason for this dip in the adoption and use of these four major tools seems to be developer habits rather than an increase in competitors. In 2017, 47% of respondents used 3-5 monitoring tools, while in 2018 this number fell to 42%. More significantly, however, was the major increase in developers who use 1-2 monitoring tools. In 2017, only 15% of respondents told us that they use 1-2 monitoring tools; in 2018, this number more than doubled, with 32% of survey respondents reporting that they use 1-2 monitoring tools.

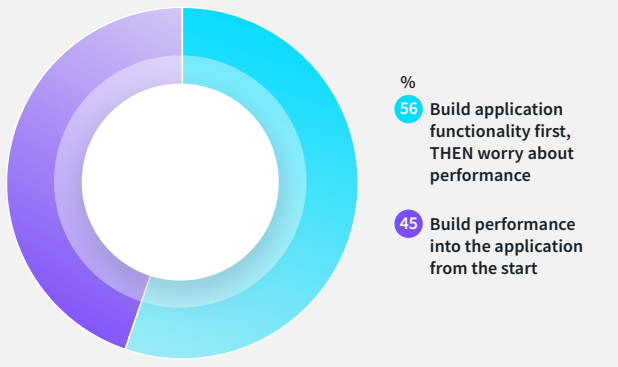
ROOT CAUSES

Given that the weekly occurrence of performance issues is on the rise, let's take a look at the problems developers experience when determining the root cause of an issue. If we recall from the "Frequency of Bugs" section, database performance issues were the second most common and the hardest to fix. Interestingly, when our audience was asked to give the most common root causes of performance hiccups, slow database queries were voted the biggest problems for daily (14%), weekly (13%), and monthly (11%) timeframes. The second largest root cause behind performance problems (again across these three timeframes) encountered by our audience was that there were too many database queries, with 14% telling us that this was a daily issue, 12% claiming it to be a weekly issue, and 10% claiming it to be a monthly issue. When compared to our data from last year's survey, these numbers appear relatively stable, with slow database queries accounting for the root cause of 12% of daily performance issues, 14% of weekly issues, and 11% of monthly issues. Similarly, in 2017, too many database queries were the root cause of 14% of daily performance issues, 13% of weekly issues, and 10% of monthly issues.

GRAPH 03. When was the last time you had to solve a performance problem in your software?



GRAPH 04. How do you prioritize in your application development process?



The 6 Most Common Performance Testing Mistakes, and How to Fix Them

QUICK VIEW

01. 15 years as a performance test consultant is plenty of time to see the patterns in how performance testing fails.

02. Regardless of the project, the company, or the tool, the same mistakes get made over and over – often without us even realizing that we made them.

03. Load testing engineer Jason Rizio takes on the six most common performance testing mistakes, with practical, actionable advice on why the issue is taking place, and how to neatly side-step each pitfall.

BY JASON RIZIO
CUSTOMER SUCCESS MANAGER, TRICENTIS

As a performance testing consultant for the last 10 years, you could say that it's second nature for me to look for performance patterns. One of the patterns I have observed over my career is that, regardless of the project size, company, or the technology being used, the same types of performance testing mistakes get made over and over and over.

This is fundamentally unsurprising, as human nature is the same regardless of the company, and every project and business environment has deadlines. Sometimes those deadlines mean testing simply *must get done*, making it easy to cut corners to get the results "over the line." Unfortunately, however, those shortcuts can lead to costly performance testing mistakes and oversights.

With a bit of self-awareness and some helpful accompanying tools however, you can often mitigate these mistakes quite easily.

1. INADEQUATE USER THINK TIME IN SCRIPTS

Hitting your application with hundreds or thousands of requests per second without any think time should only be used in rare cases where you need to simulate this type of behavior – like a Denial of Service attack. 99% of people are not testing for this scenario all the time, however; they are just trying to verify that their site can handle a target load. In that case, user think time is very important.

I've seen many people run tests without any think time with thousands of users and using 1 load injection node, then proceed to ask:

Why are my response times slow? When I manually visit the site using my browser, it seems to be responding just fine.

The cause of this is almost always the sheer volume of requests that the load injection node is tasked with, which subsequently runs into maxing out machine resources and triggering local exceptions quickly.

You can address this by simply adding some sort of think/wait time in between your virtual user's steps, effectively slowing the user down to a more realistic pace. A real-world user would never make back-to-back page requests within 1 second. Think time allows you to add a human pause, mimicking a real person who would wait a few seconds before further interacting with the site. An easy solution is to use a Gaussian Random Timer when designing your tests, which allows your users to interact in a random fashion just as they would in a real-world scenario.

2. USING AN INACCURATE WORKLOAD MODEL

A workload model is the detailed plan that you should be writing your scripts against. This workload model should outline your business processes, the steps involved, number of users, number of transactions per user, and calculated pacing for each user.

Having an accurate workload model is critical to the overall success of your testing. Often this is easier said than done — there have been many projects where business requirements simply don't exist because they weren't considered beyond, "the system should be fast."

For existing applications, it's always worthwhile to have access to production statistics, which will show a wealth of information such as:

1. What are the most common/popular transactions?
2. How many of each transaction happens on a typical business day?
3. How many of each transaction happens on a peak day?
4. What are the general times or periods that these transactions tend to occur?
5. What are the transactions that have a high business cost if they were to fail under load?

From these criteria, you can assemble a picture of the workload model you need to simulate.

For new applications, the general rule is to work as much as possible with the business representatives to ascertain realistic and agreed upon figures that are well defined, simple, and testable.

After the first testing cycle and once the application is released, on-going monitoring of usage in the production environment will help provide feedback for the next round of testing, where the workload model can be tuned further.

3. SETTING UP INADEQUATE INFRASTRUCTURE MONITORING

Your execution results like throughput, transaction response times, and error information aren't overly helpful unless you can see how your target infrastructure is coping with the scenario.

It's a common problem — I have heard many testers ask why their response times are taking minutes instead of seconds. The problem can lie either in the load generation or the target application infrastructure.

How do you solve this problem? The ideal solution is to have custom monitoring dashboards for all your on-demand load injection infrastructure. Some load testing platforms, like Tricentis Flood, provide custom monitoring dashboards upon request. This enables you to view system resource utilization while running your tests, ensuring that no bottlenecks are present on the load generation side.

On the target application side, there are quite a few tools that can be implemented to help diagnose bottlenecks or sluggish performance during a load test. There are actually at least 100+ application monitoring services available right now — all with differing price points and features. I have used many of the more popular services such as New Relic, AppDynamics, and Splunk, but these can get quite expensive for a full stack monitoring option. There are also a few open-source, free alternatives such as Nagios and InspectIT. They are not as polished, but with a little know-how and time spent setting these up, it can be worthwhile and very cost effective.

4. USING HARD CODED DATA IN EVERY REQUEST

Using the same data in your HTTP request for every user is not a realistic usage scenario. Often, smarter applications and existing database technology will recognize identical requests and automatically cache them, making the overall system appear faster than it is. This leads to an invalid performance test.

Let's take the simple act of registering a new user for a typical online shopping website. Most (if not all) sites won't let the same user be registered more than once without some unique information being specified.

Here's an example JSON payload that can be used within a JMeter request for adding a new customer:

```
{
    "firstName": "Teddy",
    "lastName": "Smith",
    "mobile": "0470123766",
    "email": "user@domain.ext",
    "currencyCode": "AUD",
    "currentMembershipStatus": "LVL 1 Member"
}
```

You would not be able to keep submitting this request into a shopping website, as there would be some verification required, particularly on the mobile and email fields.

Instead, we can make these two fields unique so we can run this request successfully each time. With JMeter, we can easily use built-in random number generators to ensure that fields like phone numbers and email addresses are unique.

```
{
    "firstName": "Teddy",
    "lastName": "Smith",
    "mobile": "${Random(0000000000,9999999999)}",
    "email":
"${RandomString(10,abcdefghijklmnopqrstuvwxyz1234567890)}@
domain.ext",
    "currencyCode": "AUD",
    "currentMembershipStatus": "LVL 1 Member"
}
```

Here, we have made two simple code changes:

1. We replaced the mobile field with a JMeter `Random` function that will generate a ten-digit random number for the mobile phone number field.
2. We replaced the email filed with a JMeter `RandomString` function that will generate a ten-character email username along with `@domain.ext`.

When we run this payload within an HTTP request in a load test, every single request will have a different mobile number and email address. This simple example of using dynamic data for all your requests can save you a lot of invalid tests.

5. IGNORING SYSTEM OR SCRIPT ERRORS BECAUSE RESPONSE TIMES AND THROUGHPUT LOOK FINE

It is quite easy to gloss over details that could have a huge impact on your test's validity. Here's common scenario:

A load test runs with a target number of users, and the tester see that response times and error rates are within acceptable ranges. Expected throughput however, is lower than anticipated. How can this be when your load testing platform is reporting very few transaction related errors?

It could be that you are encountering system-related errors that are not being reported on the transaction pass/fail counts — that is why everything looks OK to the untrained eye. A lower than anticipated number of transactions per minute or RPM value is the telltale sign of this issue.

Delving into runtime logs will reveal the issue on the scripting side, which can then be fixed. Once you have a test with transaction response times, error rates, and throughput rates all in the expected zone, are you in the clear?

Not just yet. There is almost always an overlooked area that can still impact your load tests: verification of actual system or application logs during the test.

This can be easily monitored if using a purpose built APM (Application Performance Monitoring) tool such as New Relic or AppDynamics, but a lot of the time, you will need to do this manually.

Exceptions can be caused by different factors, meaning you should carefully analyze why they were caused by your load test scenario and what impacts they have on your system's performance and stability.

Exceptions often carry an enormous performance overhead due to exception handling routines and any related logging to disk or memory operations. This might not be much of an issue with a single transaction, but multiply this over 1,000 concurrent users and it can severely impact your system's responsiveness.

6. OVERLOADING LOAD GENERATORS

The last mistake is overloading load generators due to one or more of the following:

- Too many concurrent users on a single load injection node
- The target site is very image-heavy or CSS-heavy, which impacts the number of concurrent users you can fit on a load injection node
- Load injection node hardware limitations

A site heavy in CSS and lots of images will cause a larger footprint in resource utilization than a very simple text only site or API calls. This affects the number of threads/users that can be supported per node. So how do you know what number of supported threads/users can be used comfortably per node on your load testing platform?

I would generally recommend running initial tests with a low number of users (1-100) as a scaling test. You can use a node resource dashboard that shows CPU and memory metrics to plan for node capacity.

To pinpoint if you are overloading a load injector, look for error logging, out of memory exceptions, and CPU or network utilization stats. CPU usage in the 80%+ region for a sustained period is a sign that the load injection node is being saturated. The network Rx throughput figure is also an issue in this case, as each load injection node has a limit — find out that limit and note that anything above will mean network saturation.

JASON RIZIO is from Melbourne, Australia, and has been involved in the technical QA space since studying software engineering at RMIT University. Having spent the majority of his testing career in Performance Testing consultancies, it was only a natural next step to gravitate towards helping customers in their journey of continuous performance improvement using next-generation load testing technology at Tricentis Flood.

Developing with Performance in Mind

QUICK VIEW

01. Performance is a major factor for business revenue in all aspects of programming including desktop, web, and mobile apps.

02. When working with small amounts of data, the processing time seems negligible. Once an application loops through thousands of records, the processing time can be too long and hang a user's system or cause web applications to timeout.

03. You must determine the best data types and structures to create applications with performance in mind.

BY JENNIFER MARSH
SOFTWARE DEVELOPER, PINKHATCODE.COM

When developers are faced with building a solution, they focus on "making it work." What they sometimes forget is the importance of application performance. Whether it's web, desktop or mobile device-based programming, your users want an application that responds quickly. They don't want a desktop, application that hangs or a web application that takes too long to load. When you approach a development project, it's just as important to focus on performance in addition to creating a bug-free application.

PERFORMANCE AND REVENUE

In development, businesses talk in terms of revenue. Prioritization of projects, application goals, and customer engagement are all tied with revenue. Performance metrics also tie into overall revenue for a business. Just a four-second delay in load times results in a **25% increased bounce rate on a website**. This means that if the business is making, for instance, $2 million a year in revenue with 10,000 customers, poor performance could be costing them $500,000 each year.

Desktop development may not rely on web page load times, but a hanging, buggy application results in uninstalls. Installation and downloads are the two metrics that mobile device and desktop application developers watch. When we say a "hanging" app, it's one that interferes with desktop performance and overwhelms CPU and memory resources. For a user, this means that other applications and the computer or mobile device itself runs slowly. The user can identify the culprit on their machine and eventually remove it. For a business, it means the loss of a customer and revenue.

When you approach a development project, it's just as important to focus on performance in addition to creating a bug-free application.

CODING FOR PERFORMANCE

One of the traditional statements in coding is that you only need to learn one language (preferably a C-style language), and you know them all. Learning syntax for different languages is easy. Understanding best practices for engineering and building entire platforms on any language is the difficult part.

Even if you don't know a specific language, most of them work with the same coding structures — loops, variables, functions, pointers, data types, and arrays are all examples of common structures in any language. Some languages handle memory resources and storage differently, but most of the basic performance-enhancing techniques in one language will carry over to another. For this reason, the languages used in the following examples can be applied to your own coding language of choice.

STRINGBUILDER VS. STRING FOR CONCATENATION

The `StringBuilder` class and object type is available in C# and Java. It's common for a developer to work with the basic string data type and build a string for output based on concatenation procedures. `StringBuilder` is built for better string concatenation performance and should be used in place of traditional strings when you want to manipulate values.

> Learning syntax for different languages is easy. Understanding best practices for engineering and building entire platforms on any language is the difficult part.

C#

```
public class BadStringConcatenation
{
    static void Main()
    {
        string myString = "";
        for (int i=0; i < 100000; i++)
        {
            myString += " Hello ";
        }
        Console.WriteLine ("String value: {0}", myString);
    }
}
```

Above is an example of using the string data type for concatenation. The performance-killing issue with this code is that the compiler doesn't just keep adding "Hello" to the `myString` variable. Instead, it takes a copy of the string, recreates the new string, and then adds the new value to `myString` again. The compiler has to create an entirely new string with each iteration. Since we have a 100,000-iteration loop, this function isn't optimal.

Instead, you should use `StringBuilder`.

```
public class GoodStringConcatenation
{
    static void Main()
    {
        StringBuilder myString = new StringBuilder();
        for (int i=0; i < 100000; i++)
        {
            myString.Append(" Hello ");
        }
        Console.WriteLine ("String value: {0}", myString.
ToString());
    }
}
```

With `StringBuilder`, you no longer take copies of large string variables during each iteration. Instead, you only copy the appended string and add it to the existing one. Application speed increases, and your users aren't waiting several seconds for calculations.

USE PRIMITIVE DATA TYPES INSTEAD OF OBJECTS

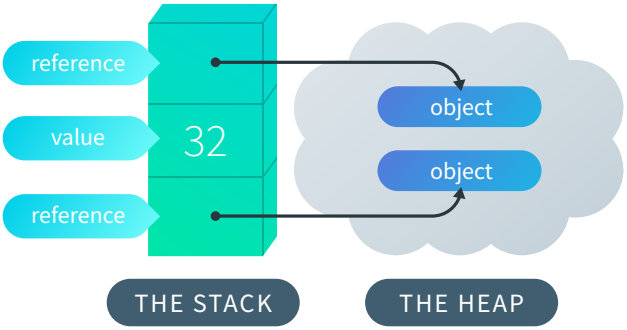
The previous example made a case for using the `StringBuilder` object, but with most procedures you want to stick with primitive data types. We'll use the Java language to illustrate why. Java has eight defined primitive types - int, double, boolean, byte, short, long, char, and float. When you can use these primitive data types, use them instead of their object data type counterpart.

The following code has two statements. The first one uses the primitive Java data type `int`. The second one uses the object `Integer`.

Java:

```
int myPrimitiveInt = 100;
Integer myObjectNumber = new Integer(100);
```

Both of these statements create an integer variable that contains the value 100. The first statement is preferred due to performance. The performance costs in the second statement stem from the way the compiler stores both of these values. Primitive data types are stored in a section of memory called the stack. Objects such as the Integer object in the second statement are stored in the stack as a reference to a second section of memory called the heap.



*Source: **Orange Coast College***

When the compiler needs to retrieve the primitive `int` variable value, it grabs it directly from the stack. However, when the compiler retrieves the second Integer object value, it grabs a reference to the heap. It then needs to go to the memory instance where the object is allocated and perform a lookup

for the value. This might seem like a subtle difference, but it plays a role in performance. When you are doing low-level programming such as gaming software, these small changes make a difference in the user's experience and your application's speed.

PRECISION COSTS YOU PERFORMANCE AND MEMORY

At a high level, you might think rounding at the 10th decimal point is not a big deal, but some applications require precision, particularly financial, engineering, or science applications. A fraction of a milligram can affect someone's health. Rounding to the wrong number could affect the structural integrity of a building. Precision and rounding are critical to these types of application. However, precision comes at a price: performance.

> The basic rule for developers is use as much precision as you need, but only use it if you need it. If you don't need a 128-bit floating point number (28-29 decimal points), then don't use it. You waste memory and CPU resources to calculate problems.

The biggest precision debate is over three primitive data types available in most languages: decimal, float, and double. The most expensive of these three is decimal, but it's the preferred data type in applications that require it.

The basic rule for developers is *use as much precision as you need, but only use it if you need it*. If you don't need a 128-bit floating point number (28-29 decimal points), then don't use it. You waste memory and CPU resources to calculate problems.

A float or double data type should be used when precision isn't important. A float is only 32 bits, and a double is 64 bits. Float is notorious for **inaccurate results**, so only use floats when you are exactly sure of what type of precision you need. This leaves double as the default for most programmers, but always consider memory usage before you determine which one to use.

USE FOR INSTEAD OF FOREACH

Similar to using `StringBuilder` instead of string for concatenation, using for instead of `foreach` also has a performance benefit. This usually becomes an issue when you have a generic list and need to iterate over each value.

C#

```
public class ListIteration
{
    static void Main()
    {
        List<Customer> customers = new List<Customer>
        foreach (var customer in customers) {
              Console.WriteLine ("Customer name: {0}",
              customer.Name);
        }
    }
}
```

In the above code, `foreach` is used to iterate through the list. A `foreach` loop uses an enumerator to "walk" through each value in the list. Enumerators contain a `Current`, `MoveNext`, and `Reset` class. Each time an iteration is made, the Enumerator must make a call to `Current` and `MoveNext` to keep track of the current and next value in the list. These calls add up when you have a large list to work through.

public class ListIteration

```
{
    static void Main()
    {
        List<Customer> customers = new List<Customer>
        for (i = 0; i < customers.Count; i++) {
              Console.WriteLine ("Customer name: {0}",
              customer[i].Name);
       }
    }
}
```

We changed the `foreach` loop to a for, and if you test the two methods you'll see that this for loop executes much faster. With a for loop, only one `get_Item` method is called, which reduces the time it takes to execute this function.

WRAP UP

Take these examples and try them out yourself. You'll notice a huge difference in execution time. Although small amounts of data might not seem like a big difference, when you are coding for businesses that rely on heavy data transactions, the difference makes a huge difference in productivity and revenue.

JENNIFER MARSH is a software developer and technical writer. She's written technical content for IBM, Rackspace, Adobe, and Udemy and continues to write articles that help people understand the life of a coder.

Stop hoping your user will report bugs

Sentry provides answers.

Get started for free at sentry.io

SENTRY

Cut Time to Resolution from 5 Hours to 5 Minutes

REDUCE UPSTREAM COST OF ERRORS BY 98%

Developers rarely have the information they need to address real problems at the source. How serious is the bug? Do we have the context to fix the right thing? Who should own resolution? How long will it take? Can we prevent regression?

By closing the insight gap, product teams reduce the upstream cost of errors by up to 98%, cutting average remediation time from five hours to five minutes. Full visibility and control means developers can remove technical overhead, address root causes, improve decision-making, and stay in their workflows to focus on what they do best: build software that makes your customers' lives better.

MOVE FAST AND FIX THINGS

Application performance relies on developers being able to fix problems before users ever encounter an error. Fixing what's broken starts with visibility and insight, which are the role of automation and aggregation. But efficiency requires ownership and control of your code from end to end.

Integrations with source code management platforms like GitHub, Bitbucket, and VSTS associate each issue with a specific author and commit to remove the typical guesswork, disruption, and time to remediation, no matter where the bug is in the application life-cycle. Errors become an opportunity to iterate and improve user experience — with zero time away from your existing workflow.

OWNERSHIP FROM END TO END

Rather than waiting until your customers churn or relying on operations to provide a partial view, developers need action-able solutions:

- Unlike logs, get alerts on specific issues in real time
- Unlike APM, fix your app where and when it breaks
- Unlike systems metrics, iterate on the code you control

WRITTEN BY RYAN GOLDMAN
VP MARKETING, SENTRY

PARTNER SPOTLIGHT

Real-Time Error Tracking by Sentry

Over a half-million developers use Sentry to see all the code-level details they need to resolve issues at every stage of the application lifecycle.

SENTRY

CATEGORY
Developer Workflow Productivity

OPEN SOURCE?
100% Open Source

RELEASE SCHEDULE
Continuous Delivery

CASE STUDY

DoorDash is a complex three-way marketplace between millions of merchants, deliverers, and customers. Scale, stability, and performance are DoorDash's primary goals, further complicated by the dependencies of multiple platforms and applications.

As the engineering team doubles every year, and deploys to production become more error-prone, Sentry helps move ownership upstream, which is essential when you're shipping as fast as DoorDash is.

Sentry is DoorDash's early-warning system for code-level changes that could break functionality. It provides enough context for the developer to fix high-priority client- or server-side issues without having to disrupt workflow before rolling back or rolling forward.

Check out our DoorDash video case study.

STRENGTHS

- See full code right in the error's stack trace and automatically apply source maps
- Filter errors by context and search custom tags relevant to your app or environment
- Expose the events and user actions that led up to each error, in context
- Track to a specific release so you know when and by whom problems were introduced
- Alert via Slack, HipChat, or PagerDuty and integrate with GitHub, Bitbucket, or Heroku

NOTABLE CUSTOMERS

- Airbnb
- PayPal
- New York Times
- Microsoft
- Dropbox
- Twitter
- Uber
- Atlassian
- Pinterest

WEBSITE sentry.io **TWITTER** @getsentry **BLOG** blog.sentry.io

Monitoring Microservices With Spring Cloud Sleuth, Elastic Stack, and Zipkin

BY PIOTR MIŃKOWSKI
IT ARCHITECT, PLAY

QUICK VIEW

01. The basic concepts around centralized log aggregation and distributed tracing in microservices-based environments.

02. Integrating Java applications with Elastic Stack using Logstash Logback Encoder.

03. Propagating context between microservices using MDC and Spring Cloud Sleuth.

04. View logs collected from the different applications in Kibana.

05. Using Zipkin as a tool for analyzing message flow and latency in communication between microservices.

One of the most frequently mentioned challenges related to migration to microservices-based architecture is monitoring. Each microservice should be run in an environment isolated from the other microservices, so they do not share any resources like data sources or log files between them.

However, the essential requirement for microservices-based architecture is that it is relatively easy to access the call history, including the ability to look through the request propagation between multiple microservices. Grepping the logs is not the right solution for that problem. There are some helpful third-party tools that can be used when creating microservices using such frameworks like Spring Boot and Spring Cloud libraries.

TOOLS

- **Spring Cloud Sleuth**: A library available as a part of the Spring Cloud project. It lets you track the progress of subsequent microservices by adding the appropriate headers to the HTTP requests. The library is based on the MDC (Mapped Diagnostic Context) concept, where you can easily extract values put to context and display them in the logs.
- **Zipkin**. A distributed tracing system that helps gather timing data for every request propagated between independent services. It has a simple management console where we can find a visualization of the time statistics generated by subsequent services.
- **Elastic Stack (ELK)**. Elasticsearch, Logstash, and Kibana — three different tools usually used together. They are used for searching, analyzing, and visualizing log data in real-time.

Many of you have probably heard about Elasticsearch and Kibana, even if you haven't done anything with Java or microservices before. For example, if you take look at **Docker Hub**, you will find the projects using the above tools among the most popular images. In our example, we will just use those images. Thanks to Docker images we will easily setup the full Elastic Stack environment on the local machine. Let's begin with running the container with Elasticsearch.

One of the most frequently mentioned challenges related to migration to microservices-based architecture is monitoring. Each microservice should be run in an environment isolated from the other microservices, so they do not share any resources like data sources or log files with them.

However, the essential requirement for microservices-based architecture is that it is relatively easy to access the call history, including the ability to look through the request propagation between multiple microservices. Grepping the logs is not the right solution for that problem. There are some helpful third-party tools that can be used when creating microservices using such frameworks like Spring Boot and Spring Cloud libraries.

```
docker run -d -it --name es -p 9200:9200 -p 9300:9300 -e
"discovery.type=single-node" docker.elastic.co/
elasticsearch/elasticsearch:6.1.1
```

Running Elasticsearch in development mode is the most convenient way, because we don't have to provide any additional configuration. If you would like to launch it in production, the vm.max_map_count Linux kernel setting needs to be set to at least 262144. The procedure of modifying it is different depending on the OS platform. For Windows with Docker Toolbox, it must be set via `docker-machine`.

```
docker-machine ssh
sudo sysctl -w vm.max_map_count=262144
```

Then, we run the Kibana container and link it to Elasticsearch.

```
docker run -d -it --name kibana --link es:elasticsearch -p
5601:5601 docker.elastic.co/kibana/kibana:6.1.1
```

At the end, we start Logstash with input and output declared. As an input, we declare TCP, which is compatible with `LogstashTcpSocketAppender` and used as a logging appender in our sample application. As an output, Elasticsearch has been declared. Each microservice will be indexed on its name with a *micro* prefix. There are many other input and output plugins available for Logstash which could be used and are listed here. Another input configuration method using RabbitMQ and Spring `AMQPAppender` is described in my post **How to ship logs with Logstash, Elasticsearch, and RabbitMQ**.

```
docker run -d -it --name logstash -p 5000:5000 logstash -e
'input { tcp { port => 5000 codec => "json" } } output {
elasticsearch { hosts => ["192.168.99.100"] index =>
"micro-%{serviceName}"} }'
```

MICROSERVICES

Now, let's take a look on sample microservices. This post is a continuation of a post on my blog, about **Creating microservices using Spring Cloud, Eureka, and Zuul**. Architecture and exposed services are the same as in the previous sample. The source code is available on **GitHub** (branch `logstash`). As I mentioned before, we will use the Logback library for sending log data to Logstash. In addition to the three Logback dependencies, we'll also add libraries for Zipkin integration and Spring Cloud Sleuth starter. Here's a fragment of the `pom.xml` for microservices:

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-sleuth-zipkin</artifactId>
</dependency>
<dependency>
   <groupId>net.logstash.logback</groupId>
   <artifactId>logstash-logback-encoder</artifactId>
   <version>4.9</version>
</dependency>
<dependency>
   <groupId>ch.qos.logback</groupId>
   <artifactId>logback-classic</artifactId>
   <version>1.2.3</version>
</dependency>
<dependency>
   <groupId>ch.qos.logback</groupId>
   <artifactId>logback-core</artifactId>
   <version>1.2.3</version>
</dependency>
```

There is also a Logback configuration file in the `src/main/resources` directory. Here's a `logback.xml` fragment. We can configure which logging fields are sent to Logstash by declaring tags like mdc, `logLevel`, message, etc. We are also appending a service name field for Elasticsearch index creation.

```
<appender name="STASH" class="net.logstash.logback
appender.LogstashTcpSocketAppender">
   <destination>192.168.99.100:5000</destination>
   <encoder class="net.logstash.logback.encoder
   LoggingEventCompositeJsonEncoder">
      <providers>
           <mdc />
           <context />
           <logLevel />
           <loggerName />
           <pattern>
               <pattern>
                    {
                    "serviceName": "account-service"
                    }
              </pattern>
           </pattern>
           <threadName />
           <message />
           <logstashMarkers />
           <stackTrace />
     </providers>
   </encoder>
</appender>
```

The configuration of Spring Cloud Sleuth is very simple. We only have to add the `spring-cloud-starter-sleuth` dependency to `pom.xml` and declare sampler `@Bean`. In the sample, I declared `AlwaysSampler`, which exports every span — but there is also another option, `PercentageBasedSampler`, which samples a fixed fraction of spans.

```
@Bean
public AlwaysSampler defaultSampler() {
  return new AlwaysSampler();
}
```

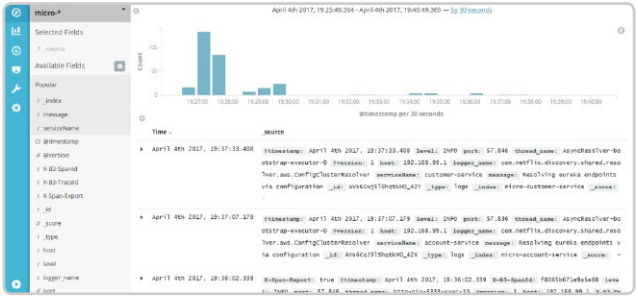
KIBANA

After starting ELK docker containers we need to run our microservices. There are five Spring Boot applications that need to be run:

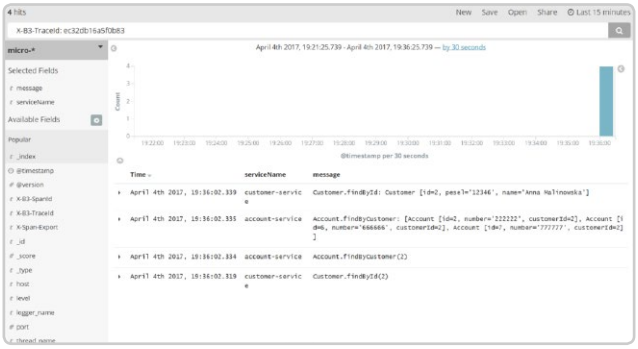
1. `discovery-service`
2. `account-service`
3. `customer-service`
4. `gateway-service`
5. `zipkin-service`

After launching all of them, we can try call some services — for example, http://localhost:8765/api/customer/customers/{id}, which causes the calling of both customer and account services. All logs will be stored in Elasticsearch with the `micro-%{serviceName}` index. They can be searched in Kibana with the `micro-*` index pattern. Index

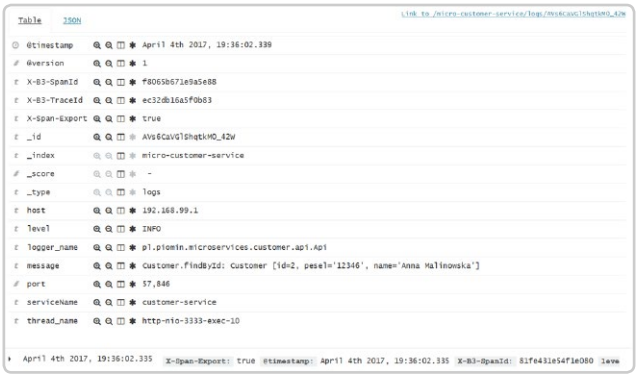
patterns are created in Kibana under section **Management > Index** patterns. Kibana is available under http://192.168.99.100:5601. After running it, we will be prompted for an index pattern, so let's type `micro-*`. Under the **Discover** section, we can take a look at all logs matching the typed pattern with a timeline visualization.



Kibana is a rather intuitive and user-friendly tool. I will not describe in detail how to use Kibana because you can easily consult the documentation or just go through the UI. The most important thing is to be able to search logs by filtering criteria. In the picture below, there is an example of searching logs by the `X-B3-TraceId` field, which is added to the request header by Spring Cloud Sleuth. Sleuth also adds `X-B3-TraceId` for marking requests for a single microservice. We can select which fields are displayed in the result list; in this sample, I selected `message` and `serviceName`, as you can see in the left pane of the picture below.



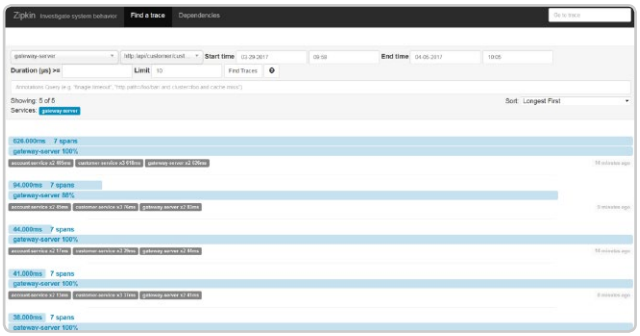
Here's a picture with single request details. It is visible after expanding each log row.



ZIPKIN

Spring Cloud Sleuth may also send tracing statistics to Zipkin. That is another kind of data than the data stored in Elastic Stack. These are timing statistics for each request. Zipkin UI is really simple. You can filter the requests by some criteria like time, service name, and endpoint name.

Below is a picture with the same requests that were visualized with Kibana (http://localhost:8765/api/customer/customers/{id}).



We can always see the details of each request by clicking on it. Then, you see the picture similar to what is visible below. In the beginning, the request has been processed on the API gateway. Then, the gateway discovers customer service on the Eureka server and calls that service. Customer service also has to discover the account service and then call it. In this view, you can easily find out which operation is the most time-consuming.

CONCLUSION

A microservices-based system is by definition a set of independent, relatively small standalone applications. There is no an upper limit for the number of microservices in your system. Their number can even reach a total of a few hundred. Considering that each for them may be launched in some instances we are talking about a couple of thousand independent applications. To successfully monitor such a large system, we have to collect and store logging and tracing data in a one, central place. With tools like Elastic Stack and Zipkin, monitoring microservices-based systems is not a difficult problem to solve. There are also some other tools — for example, Hystrix and Turbine — that provide real-time metrics for all the incoming requests.

PIOTR MIŃKOWSKI has more than 10 years of experience working as a developer and architect in the finance and telecom sectors, specializing in Java and its associated tools and frameworks. He works at Play, a mobile operator in Poland, where he is responsible for IT systems architecture. Here, he helps the organization migrate from monoliths to microservices-based architectures, as well as set up a CI/CD environment. In his free time, he publishes articles on Piotr's TechBlog, where he demonstrates the newest technologies and frameworks in the programming world.

DIVING DEEPER

INTO PERFORMANCE

#PERFORMANCE TWITTER ACCOUNTS

 @SOUDERS

 @BBINTO

 @ANDYDAVIES

 @PAUL_IRISH

 @IGRIGORIK

 @BMEURER

 @YOAVWEISS

 @LAVRTON

 @ANDREAPERNICI

 @GUYPOD

PERFORMANCE ZONES

Performance
dzone.com/performance
Scalability and optimization are constant concerns for the Developer and Operations manager. The Performance Zone focuses on all things performance, covering everything from database optimization to garbage collection to tweaks to keep your code as efficient as possible.

Mobile
dzone.com/mobile
Web professionals make up one of the largest sections of IT audiences. We collect content that helps web pros navigate in a world of quickly changing language protocol, trending frameworks, and new standards for UX. The Web Dev Zone is devoted to all things web development, including frontend UX, back-end optimization, JavaScript frameworks, and web design.

Web Dev
dzone.com/webdev
The Mobile Zone features the most current content for mobile developers. Here you'll find expert opinions on the latest mobile platforms, including Android, iOS, and Windows Phone. You can find in-depth code tutorials, editorials spotlighting the latest development trends, and insights on upcoming OS releases.

PERFORMANCE REFCARDZ

Scalability and High Availability
Provides the tools to define Scalability and High Availability, so your team can implement critical systems with well-understood performance goals.

Java Performance Optimization
This Refcard covers JVM internals, class loading (updated to reflect the new Metaspace in Java 8), garbage collection, troubleshooting, monitoring, concurrency, and more.

Getting Started with Real User Monitoring
Teaches you how to use new web standards—like W3C's Beacon API—to see how your site is performing for actual users, letting you better understand how to improve overall user experience.

PERFORMANCE RESOURCES

Performance Optimization Resources
Get tons of resources for optimizing the performance of various products, from Microsoft SQL Server to Microsoft IIS for Windows.

18 Tips for Website Performance Optimization
Get some top tips for performance optimization, regardless of the platform you're using.

Research on Performance Optimization & Implementation of Oracle Database
See exactly how database optimization can improve the performance of the database and decrease client response time.

PERFORMANCE BOOKS

Web Performance in Action: Building Fast Web Pages
Learn how to speed up the delivery of your site's assets to the user, increase rendering speed, build workflows that automate common optimization techniques, and more.

High Performance Mobile Web: Best Practices for Optimizing Mobile Web Apps
Learn about which aspects of your site or app slow down the user's experience and what you can do to achieve lightning-fast performance.

Ruby Performance Optimization: Why Ruby Is Slow, and How to Fix It
Learn how to write faster Ruby code, and see exactly what makes Ruby and Rails code slow and how to fix it.

A+APM

"Shifting left" has been a common talking point of agile and DevOps methodologies for several years now and has become common wisdom for those looking to increase productivity and improve software quality. In fact, according to the 2018 DZone Performance survey, 85% of DZone users are encouraged to develop with performance in mind. However, only 45% are actually putting that philosophy into practice. With that in mind, we wondered how prepared our audience would be to deal with common performance issues, because a pop quiz can strike at any time. We looked at how frequently they occur, how difficult they are to solve, and how long it takes to solve them.

catchpoint™

Detect and Fix Issues Fast

With Synthetic and Real User Monitoring

Sign up for a free trial: **catchpoint.com/freetrial**

Monitoring and Observability

Defining the term monitoring is a difficult task giving the evolution of the space over the years. Monitoring can include system, component, web performance, application performance monitoring, digital experience, and end-user monitoring. Lately, there has been a shift in the monitoring world and the term observability has emerged. What is the difference between monitoring and observability?

The purpose of monitoring is to detect issues and minimize the effect on end users. Can users access your application and services, are they performing within appropriate thresholds? Alerts are configured to inform teams when something is not performing optimally or is broken. But before an alert can be configured, it is critical to create a baseline and understand the current state of the environment. This is where observability comes in. Without observing a system, it is impossible to know what is normal and what is not normal.

Organizations should constantly be observing the state of their services and applications to look for outliers and anomalies that wouldn't be detected from previously configured alerts. Aggregating data points to create means, medians, percentiles, or another consumable statistic is useful in some instances, but you also need access to the unaggregated data to find the anomalies and patterns that may reveal hidden insights.

Anomalies and outliers won't always trigger an alert and they shouldn't always trigger an alert. Anomalies can lead us to deeper insight into our systems, provide additional context, and help us identify key attributes. If the only thing we are focused on is whether an alert is generated, we are missing large parts of the story.

WRITTEN BY DAWN PARZYCH
DIRECTOR OF PRODUCT MARKETING, CATCHPOINT

PARTNER SPOTLIGHT

Catchpoint

Detect and fix issues faster for complete visibility into your users' experiences from anywhere

CATEGORY
DEM - Digital Experience Monitoring / Performance Monitoring

OPEN SOURCE?
No

RELEASE SCHEDULE
6-8x per year

CASE STUDY

LinkedIn services users in every corner of the globe, which requires a wide array of third-party vendors, including three DNS resolution services and five content delivery networks (CDNs), plus their own internal CDN. But with so much reliance on third-party solutions for their digital delivery, the LinkedIn SRE team has to keep a close eye on the performance of these cloud solutions in order to ensure that their end users are getting the best digital experience possible. This can be especially challenging in certain parts of the world like China, India, other parts of Asia, and South America, where the local infrastructure on the ground is often unable to provide the level of performance that LinkedIn requires.

Continue reading Catchpoint's LinkedIn case study here.

STRENGTHS

- The industry's most extensive test types
- The industry's most comprehensive global node coverage
- Richest, most complete view of the customer's experience
- Easy to deploy and use

NOTABLE CUSTOMERS

- Google
- Verizon
- Oracle + Dyn
- Honeywell
- Priceline

WEBSITE catchpoint.com **TWITTER** @catchpoint **BLOG** blog.catchpoint.com

How to Avoid Finger-Pointing and Uncover the True Cause of Incidents

BY DAWN PARZYCH
DIRECTOR AT **CATCHPOINT**

QUICK VIEW

01. It is human nature to search for an explanation as to why something went wrong. Decisions are governed by emotion not logic.

02. The focus of an incident review should be on how processes and systems can be changed or improved the next time, not on finding who or what was at fault. Avoid blame.

03. Break discovery into discrete areas such as detection, notification, assessment and resolution.

04. Put people first when developing processes or reviewing the details of an incident.

What do the Equifax Breach, the S3 outage, and the wrong movie being named Best Picture at the 2017 Oscars have in common? They were all attributed to human error. While humans were all involved in these incidents, the answer to what actually led to these incidents is way more complicated.

We have built complex systems that break in unknown and unpredictable ways. Attributing something to human error drastically oversimplifies things. Yes, humans were involved and yes, a mistake may have been made. But chances are pretty high that it wasn't just a single person, and the processes and systems in place did not have appropriate protections to prevent such a thing.

When something goes wrong, we naturally search for an explanation or a reason why. Knowing why something happens makes us feel better and helps us put a process in place to make sure this doesn't happen again, or to us. This process goes by multiple names:

- Root cause analysis
- Post-mortem
- Lessons learned
- Post-incident review

POST-MORTEM

A post-mortem is a process to identify a chronological timeline of the events preceding the incident. The intention is to determine what happened and put processes in place to prevent a similar incident from happening again.

> Even if it is called a blameless post-mortem, the process itself is built to identify the cause of the problem.

ROOT CAUSE ANALYSIS

Root cause analysis is often included as part of a post-mortem. It's designed to identify the primary reason why an incident occurred.

The definitions of post-mortem and root cause analysis are focused on finding reasons why an incident occurred and preventing incidents from happening in the future. The challenge with post-mortems and root cause analyses is the desire to attribute an incident to a single issue or person. Even if it is called a blameless post-mortem, the process itself is built to identify the cause of the problem. Oftentimes there is never a single cause, rather a combination of events aligned in a certain way contributing to the incident.

Instead of focusing solely on why something occurred and what we can do to prevent similar events from happening in the future, we should strive to focus on what was learned from the incident. Instead of conducting a post-mortem or root cause analysis, hold a lessons learned or post-incident review.

LESSONS LEARNED

Rooted in project management, lessons learned is a formal process of reviewing a project (or incident) and documenting what was learned during it. For incident reviews, this can include a review of how effective the incident handling process was and identifying what needs to be improved or changed. Learning from the experience of others is more valuable than trying to not get blamed for a mistake. If all we are doing is identifying what went wrong and not learning from the incident, we will end up making the same mistakes.

POST-INCIDENT REVIEW

This year saw a trend in IT moving toward a new concept of post-incident reviews as opposed to root cause analysis and post-mortems. Post-incident reviews put the focus on what can be learned from outages, incidents, and failures. The goal of a post-incident review is to examine the timeline to determine what happened and whether it was handled correctly. The outcome of this is to develop better processes and strategies for future incidents.

If an organization or team is suffering from alert fatigue, it is possible that alerts are ignored, which results in longer detection.

KEY COMPONENTS OF ANY INCIDENT REVIEW

DETECTION

The amount of time it takes to detect an incident depends on the tools and processes in place. If an organization or team is suffering from alert fatigue, it is possible that alerts are ignored, which results in longer detection. If the proper monitoring and alerts are not configured, customers may be detecting issues before the company knows something is wrong.

Questions to consider:

- How was the incident detected?
- How do processes and tools need to change to detect incidents earlier?

NOTIFICATION

Receiving an alert during the detection phase is part of a notification. Oftentimes there are other stakeholders that need to be notified both inside and outside of the organization. Part of the incident review for notifications should include any relevant social media communications that occurred during the incident. Notifying customers and maintaining communication can help improve a company's reputation when services are unavailable.

Questions to consider:

- How were relevant parties notified of the incident?
- How do notification procedures need to change for more effective/timely notifications?
- How were social media relations handled?
- How do social media communications need to change?

ASSESSMENT

For triage and communication processes, incidents often need to be categorized in terms of severity. Not every incident requires an external notification or social media strategy. Assessing and tracking the severity of an incident can result in different protocols being implemented.

Questions to consider:

- How was the severity of the incident assessed?
- How did the severity change through the course of the incident?

RESOLUTION

Sometimes a fix is easy to determine, sometimes it takes a while to determine what needs to be fixed, and oftentimes there is a lot of trial and error. Understanding the dynamics involved in the resolution can identify areas to improve and reduce the overall mean time to repair.

Questions to consider:

- How was it determined what changes were needed to resolve the incident?

- How do resolution processes need to change?

It is important to phrase the question in open and non-accusatory ways. Notice the difference between asking "How was the severity of the incident assessed?" and "How do assessment procedures need to change in the future for more accurate assessments?" and "Was the severity of the incident assessed correctly from the beginning?" The latter implies blame, if the incident did not receive an accurate severity level then somebody or something is to blame. Asking a "How?" question instead makes people pause and problem solve instead of trying to assign blame. The answer to how does something need to change, can be "It doesn't." If everything ran smoothly, then nothing needs to change.

Notice the difference between asking "How was the severity of the incident assessed?" and "How do assessment procedures need to change in the future for more accurate assessments?" and "Was the severity of the incident assessed correctly from the beginning?"

THE HUMAN FACTOR

All the tools and processes put in place are meaningless unless they are designed with a focus on the people. This doesn't mean we look to assign blame to a person or attribute an incident to human error, but rather we look at how people are interacting with the tools and their frame of mind. Understand the perspective, bias, and motivation of everybody involved.

The implementation of blameless post-mortems took people into account. People want to avoid blame. The idea that somebody was to blame creates fear and anxiety. "Will I be fired?" "Will I lose the promotion I was hoping for?" Understand that when people are in a positive frame of mind they are more likely to collaborate and provide truthful answers when problem solving.

We have turned to automation as an attempt to eliminate the chances of human error causing incidents. However, we still see major incidents occurring because we are not considering the humans involved in the processes we are implementing. We like to think that we are rational beings that use logic to make decisions, but our decisions are driven by emotion. According to Chris Voss in *Never Split the Difference: Negotiating As If Your Life Depended On It*, "While we use logic to reason ourselves toward a decision the actual decision making is governed by emotion."

When we are building software, automating tasks, or developing processes we have to think about the humans involved:

- When will they be using this? If the process or tool is used during incidents, there is additional pressure and stress during this time. Given that emotions drive decisions, what can be done to lessen the emotions or reduce pressure?
- If the user is woken up in the middle of the night, how can I make it easier for them to quickly take the necessary actions? Can additional information be included in alerts to help with diagnosing a problem?
- Is information in protocols and processes conveyed in the simplest way possible, can it be understood by both junior and senior employees at 2 AM?
- What can potentially go wrong?

Technical solutions that automate tasks are good, but there is a human behind the automation tasks. We need human and technical solutions to help us resolve incidents and have to be willing to acknowledge when a process is broken or we didn't have the appropriate safeguards in place. Strive to avoid the blame or pinning an incident on a human error, unless that error is "we didn't have the right safeguards in place," or "we don't understand the myriad ways our systems can potentially fail."

DAWN PARZYCH (@dparzych) is a Director at Catchpoint. She enjoys researching, writing and speaking about the intersection of technology and psychology with a focus on application performance, bias, and culture. Prior to entering the tech field, Dawn obtained a master's degree in Psychology. She has spent over 15 years working in the application performance space in a wide variety of roles at leading technology companies.

A Short History of Performance Engineering

BY ALEX PODELKO
CONSULTING MEMBER OF TECHNICAL STAFF AT **ORACLE**

QUICK VIEW

01. Performance engineering has a rather long and fascinating history.

02. Performance expertise related to a new computing paradigm usually materializes later, when the technology matures.

03. The feeling that we are close to solving performance problems has existed for at least 50+ years.

04. Don't wait for a silver bullet - understand different existing approaches to mitigating performance risks and find an optimal combination of them to address them.

Performance engineering has a rather long and fascinating history, especially if considered in the context of changing computing paradigms. While not everything in the past may be applied to every new technology, the underlying principles often remain the same — and knowledge of history keeps us from reinventing the wheel when it is unnecessary. Unfortunately, statements referring to the past are not often completely correct. The history of performance engineering is not well-known, so here is some information that I find to be quite interesting. The approach was to find the first mature appearance of still-relevant performance concepts (without diving into the in-depth history of each notion). It is not scientific research, and not much information is available overall — so a lot of important information may still be missed.

We will start by listing the following computing paradigms:

Mainframes	**Late '50s**
Distributed Systems	**Late '70s**
Web	**Mid '90s**
Mobile, Cloud	**Mid 2000s**

Performance expertise related to a new paradigm usually materializes later when the technology is more mature.

MAINFRAMES

Performance went beyond single-user profiling when mainframes started to support multiprogramming. In the early mainframe years, processing was concerned mainly with batch loads. Mainframes, however, had sophisticated scheduling and could ration consumed resources. They also had pretty powerful OS-level instrumentation allowing the engineers to track down performance issues. The cost of mainframe resources was high; therefore, capacity planners and performance analysts were needed to optimize mainframe usage.

While not everything in the past may be applied to every new technology, the underlying principles often remain the same — and knowledge of history keeps us from reinventing the wheel when it is unnecessary.

We can definitely say that performance engineering became a distinct discipline when instrumentation was introduced with **SMF (System Management Facilities)**, released as part of OS/360 in 1966 (still in use in IBM z/OS mainframes today).

In 1968, Robert Miller (IBM) in his ***Response Time in Man-Computer Conversational Transactions*** paper described several threshold levels of human attention. The paper was widely cited by many later researchers and remains mostly relevant now.

In 1974, monitoring was introduced with **RMF (Resource Measurement Facility)** as part of MVS (still in use). OMEGAMON for MVS by Candle (acquired by IBM in 2004), released in 1975, is often claimed to be the first real-time monitor.

A performance community, the **Computer Measurement Group (CMG)**, was created in 1974, holding annual conferences ever since — now across a wide spectrum of technologies.

In 1977, BEST/1 was released by BGS Systems (acquired by BMC in 1998), the first commercial package for computer performance analysis and capacity planning to be based on analytic models.

DISTRIBUTED SYSTEMS

When the paradigm changed to client-server and distributed systems, the available operating systems at the time didn't have much instrumentation or workload management capabilities. Load testing and system-level monitoring became the primary ways to handle multi-user performance. Deploying across multiple machines was more difficult, and the cost of rollback was significant, especially for Commercial Off-The-Shelf (COTS) software that might be deployed by hundreds or even thousands of customers. Thus, there was more of a need for performance design to be correct from the beginning.

"**Fix-it-later was a viable approach in the 1970s, but today, the original premises no longer hold — and fix-it-later is archaic and dangerous. The original premises were:**

- **Performance problems are rare.**
- **Hardware is fast and inexpensive.**
- **It's too expensive to build responsive software.**
- **You can tune software later, if necessary.**"

Have you heard something like this recently? That is a quote from Dr. Connie Smith's ***Performance Engineering of Software Systems***, published in 1990. The book presented the foundations of software performance engineering, and already had 15 pages of bibliography on the subject. The most known load testing tool, LoadRunner, was released in 1991 by Mercury Interactive (acquired by HP in 2006, now part of Micro Focus). For a while, load testing became the main way to ensure high performance of distributed systems, and performance testing groups became the centers of performance-related activities in many organizations.

"Fix-it-later was a viable approach in the 1970s, but today, the original premises no longer hold — and fix-it-later is archaic and dangerous."

The term Application Performance Management (APM) was coined by Programart Corp. (acquired by Compuware in 1999) in 1992 (in the mainframe context, as a combination of their STROBE and APMpower tools). However, STROBE, which they refer to as an application performance measurement tool, had been on the market since the '70s. Still, there is an opinion that the first APM tool — as we know them now — was Introscope by Wily Technology, founded by Lew Cirne in 1998 (acquired by CA in 2006).

The history of End-User Monitoring (EUM) / Real-User Monitoring (RUM) can be traced at least to ETEWatch (End-to-End Watch), an application response time monitor released in 1998 by Candle (acquired by IBM in 2004, then a part of Tivoli). However, EUM/RUM gained popularity later with development of web and mobile technologies.

WEB AND MOBILE

Most existing expertise was still applicable to the back-end. The first books to apply existing knowledge and techniques to the web were published in 1998 — for example, ***Web Performance Tuning*** and ***Capacity Planning for Web Performance***.

In 2007, Steve Souders published ***High Performance Web Sites: Essential Knowledge for Front-End Engineers***, stating

that 80-90% of user response time is spent in the browser, which started a movement of Web Performance Optimization (WPO) centered on the client-side.

The WPO community was built around the **Velocity Conference** (first held in 2008) and **Web Performance meetups**. Velocity was a very popular performance conference — at least until Steve Souders stepped off as an organizer, O'Reilly merged Web Performance into the Fluent conference, and Velocity became more of a DevOps conference. Maybe it was an indication that WPO had become more mature and integrated with other aspects of technology.

Mobile technologies supported the further development of web performance, as client-side performance was even more important on mobile devices.

CLOUD

In the last ten years, we saw another paradigm shift to the cloud. While the term "cloud computing" was popularized when **Amazon released its Elastic Compute Cloud in 2006, references to "cloud computing" have appeared as early as 1996**. Technologies mature more quickly nowadays — for example, Amazon's own monitoring solution **CloudWatch was released only three years later, in 2009**. Of course, many established performance products started to support cloud, and new products still enter the market.

While the cloud looks much different than mainframes, there are many similarities between them, especially from a **performance point of view**. They both provide:

- Availability of computer resources to be allocated.
- An easy way to evaluate the cost associated with these resources and implement chargeback.
- Isolation of systems inside a larger pool of resources.
- Easier ways to deploy a system and pull it back if needed without impacting other systems.

However, there are notable differences that make managing performance in the cloud more challenging. First of all, there is no instrumentation on the OS level, and even resource monitoring becomes less reliable due to the virtualization layer. So, instrumentation must be on the application level. Second, systems are not completely isolated from the performance point of view, and they could impact each other, and we mostly have multi-user interactive workloads, which are difficult to predict and manage. That means that such performance risk mitigation approaches like APM, performance testing, and capacity management become very important in a cloud environment.

WHAT LIES AHEAD?

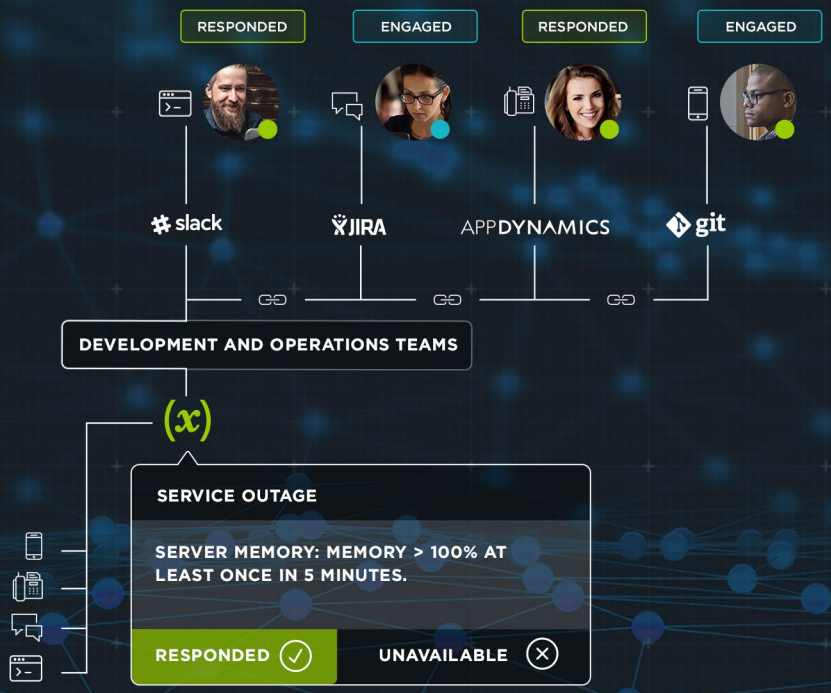
While performance is the result of all design and implementation details, the performance engineering area remains very siloed — maybe due to historic reasons, maybe due to the huge scope of expertise. People and organizations trying to condense all performance-related activities together are rather few and far apart. Attempts to span different silos (for example, DevOps) often leave many important performance engineering areas out.

Technologies mature more quickly nowadays — for example, Amazon's own monitoring solution CloudWatch was released only three years later, in 2009.

The main lesson of the history is that the feeling that we are close to solving performance problems has existed for the last 50+ years, and it will probably stay with us for a while — so instead of hoping for a silver bullet, it is better to understand different existing approaches to mitigating performance risks and find an optimal combination of them to address performance risks in your particular context.

ALEX PODELKO has specialized in performance since 1997 and is currently working as a Consulting Member of Technical Staff at Oracle, responsible for performance testing and optimization of Performance Management and Business Intelligence products. His collection of links, recent articles, and presentations can be found at alexanderpodelko.com. He serves as director for the Computer Measurement Group (CMG), an organization of performance and capacity planning professionals.

Maximize Your APM Investment with xMatters



Connect your APM alerts to your on-call schedule, ticketing system, status page, chat, etc.

ADD RICH INSIGHTS TO YOUR ALERTS

ENABLE COLLABORATION VIA CHAT, SMS, VOICE, ETC.

RELAY MONITORING ALERTS BETWEEN SYSTEMS

Identify and resolve incidents faster with xMatters. xMatters.com/maxapm

(x) matters®

Enhance Your APM Notifications

APM software notifies someone when one of your applications misbehaves. But knowing you have a problem is just the first step. With the average cost of downtime estimated as high as $8,800 a minute, you need to get the right people involved ASAP. There are 3 core elements to turning your APM alerts into quick action:

1. RELAY ALERTS BETWEEN SYSTEMS

Insights from your APM tool need to be relayed to other systems to either ensure other teams have visibility or to make sure resolution reaches the finish line. Automate communication between your tools, teams, and people to get data out of your APM solution and into your service desk, issue tracker, and other systems.

2. ENRICH NOTIFICATIONS

Solving complex issues requires data from several systems, not just the APM notification. Adding relevant data from multiple systems to every notification gives your people the business insight to solve issues faster.

3. ENGAGE THE RIGHT PEOPLE TO COLLABORATE

All this data is great, but it must get to an actual human. Automate finding the people who are on-call right now. Engage them on their preferred devices. Empower them with alerts that have actions like Open Ticket, Start Chat Room, Launch Conference Bridge, Update Status, and more.

xMatters uses powerful closed-loop integrations with APM, ITSM, chat, and many more solutions to give useful actions for individuals and teams. xMatters pushes data back into APM tools so users have easy access to information. And it preserves every action and chat transcript in your systems of record for easier collaboration and better post-event analysis. Read the full whitepaper about **3 Steps to Monitoring in a Connected Enterprise**.

WRITTEN BY ABBAS HAIDER ALI
CTO, XMATTERS

PARTNER SPOTLIGHT

xMatters

xMatters is an integration-driven collaboration platform that relays data between systems while engaging the right people to resolve incidents.

CATEGORY
IT Alerts

OPEN SOURCE?
No

RELEASE SCHEDULE
Quarterly

CASE STUDY

Kellogg Cuts Resolution Times by 83%

The Kellogg Company initiated a three-year plan to elevate company efficiency. Along with finding a new IT monitoring solution, they sought to automate IT event notifications and integrate them with the monitoring.

xMatters tied smoothly into Kellogg's broader cloud-based infrastructure, including its new monitoring solution and ITSM and CMDB applications. Events in the monitoring and help desk systems automatically kick off tailored alerts from xMatters for 2,000 potential events to the relevant people across 88 global teams via phone, SMS, email and push—whichever format the recipient prefers. If someone doesn't accept the alert, xMatters escalates the issue to the next person on the list, ensuring a timely response and preventing alert fatigue.

With xMatters, Kellogg was able to:

- Reduce mean time to resolution by 83%
- Reach alerting accuracy exceeding 99.99%
- Cut resource costs by 92% in conjunction with monitoring and ticketing—a savings that is expected to add up to $2.5 million over 5 years

STRENGTHS

- Connect insights from APM tools to the people, teams, and systems needed to drive business processes forward.
- Leverage group on-call schedules and rotations, escalation rules, and user device preferences to automatically engage the right resources.
- Customized response options allow teams to quickly get up to speed and take action during incidents.
- Robust self-service integration platform with 100s of prebuilt configurations that can be installed in minutes.
- 15 years of experience integrating people into toolchains spanning DevOps, Ops, and Service Management solutions.

NOTABLE CUSTOMERS

- 3M
- Dealertrack / Cox Automotive
- Fiserv
- Fujitsu
- The Kellogg Company
- Manpower
- ViaSat
- Walgreens

WEBSITE xmatters.com/maxAPM **TWITTER** @xmatters_inc **BLOG** xmatters.com/blog

Executive Insights on the State of Performance

QUICK VIEW

01. Performance and scalability are the key benefits to focus on with regards to application tuning and testing.

02. The most significant changes in performance testing and tuning have been the cloud and the complexity of tools and solutions.

03. Real world problems being solved by performance testing and tuning are stability, speed, scalability and security of apps.

BY TOM SMITH
RESEARCH ANALYST AT **DZONE**

To gather insights on the current and future state of Performance Testing and Tuning, we talked to 14 executives involved in systems performance. Here's who we spoke to:

- **Dawn Parzych**, Director of Product and Solution Marketing, **Catchpoint Systems Inc.**
- **Andreas Grabner**, DevOps Activist, **Dynatrace**
- **Amol Dalvi**, Senior Director of Product, **Nerdio**
- **Peter Zaitsev**, CEO, **Percona**
- **Amir Rosenberg**, Director of Product Management, **Perfecto**
- **Edan Evantal**, VP, Engineering, **Quali**
- **Mel Forman**, Performance Team Lead, **SUSE**
- **Sarah Lahav**, CEO, **SysAid**
- **Antony Edwards**, CTO and **Gareth Smith**, V.P. Products and Solutions, **TestPlant**
- **Alex Henthorn-Iwane**, V.P. Product Marketing, **ThousandEyes**
- **Tim Koopmans**, Flood IO Co-founder & Flood Product Owner, **Tricentis**
- **Tim Van Ash**, S.V.P. Products, **Virtual Instruments**
- **Deepa Guna**, Senior QA Architect, **xMatters**

KEY FINDINGS

1. **Performance and scalability are the key benefits to improving testing and tuning.** Performance is key for any online service or app. Start with the product and the desired user experience. Identify what performance metrics are going to be used to delineate between "good" and "bad" performance for a given workload. Once you have established metrics for success, have a reproducible, reliable, and representative test scenario with multiple users, as well as the ability to measure actual real-time performance.

Know how the application scales and how you plan to deliver performance expectations as the app grows. Be able to monitor every transaction and command in real time so you are able to see dynamic and automated scaling of compute resources based on real usage trends.

2. **The most significant changes in performance testing and tuning have been the cloud and the complexity of tools and solutions.** Cloud and DBaaS cover the operational basics; however, IT departments are still responsible for incidents and performance, and need ways to identify when problems are occurring in real-time. This includes monitoring containers, PaaS, AI, and serverless technologies.

The complexity and scale of applications and infrastructure has grown along with the legs of communication with geographically dispersed workers, customers, and devices. There are multi-

ple choices of open source databases; however, as the choices have grown, so has the complexity. In the last 12 months, we've seen tremendous growth in client-side apps thanks to HTML5 and the UX expectations from apps and transactions.

3. The most popular technical solutions for performance testing and tuning are **open source and proprietary solutions developed and used by companies, rather than third-party tools.** Companies are also developing their own open source tools for developers to use. Tool kits are easy to use for summaries and queries.

4. Real world problems being solved by performance testing and tuning are the **stability, speed, scalability, and security** of apps used for social media, rich media, financial services, healthcare, retail, and cold storage purposes.

One respondent's social media client wanted to launch a new product in a week. They had a big budget for advertising and anticipated having 500,000 users in two months. However, they had only tested their app with 10 users. The respondent's team created a test database to see how the app would function with 500,000 users. It took the home page three hours to load. The client delayed the launch while they re-architected the features and queries to accelerate response.

Another respondent helps to increase engagement with Citibank's mobile and web digital channels to increase transaction volume, reduce time to market, improve performance, and improve the overall user experience.

One executive's company helps to address flakiness in mobile apps due to poor bandwidth and poor device performance. They've helped a professional social media platform with latent, non-responsive apps, a major airline with their mobile and web applications, rich media companies with streaming difficulties, and banks trying to facilitate the transactions of their customers.

5. The most common issue affecting performance testing and tuning is **not implementing best practices.** There's often an incorrect use of network bandwidth across applications. Developers need to ensure that queries are valid queries before production, not just testing queries. You need to maintain a well-maintained platform with efficient code, have an integrated view of the infrastructure and the application, and optimize the configuration of the app. You need to have a deployment environment management system that replicates real use cases. Monitor performance on diverse technologies, and don't leave performance testing and tuning up to an individual instead of a team that can test for more diverse scenarios.

6. The future of performance testing and tuning is **automation, DevOps, and user-centric integrated testing.** Performance testing will continue to shift left with automated analysis and performance testing integrated as a part of DevOps methodologies. There will be an integrated single-pane view of the application and the infrastructure. AI will be used to increase and improve automation, as well as explore applications for potential performance issues. New testing tools and methods will emerge to provide more visibility for cloud applications and services. There will be self-healing performance engineering and platform architectures, as well as smarter remediation actions based on better data and automation scripts. Performance testing in isolation is a thing of the past. We'll continue to see more load tests happening during continuous integration or deployment pipelines.

7. The biggest concerns around performance testing and tuning today is the lack of it. **Only 6% of companies are doing continuous performance testing, and no one is doing continuous UI testing.** There needs to be greater integration between applications teams and infrastructure and the realization that performance testing and tuning is a shared responsibility.

8. The skills developers need to ensure the applications they are developing perform well when tested are to **shift everything left in the SDLC** that they can to save themselves time down the road and to produce more reliable apps. Think functionality first and scalability a very close second. Code efficiently so your app is able to scale efficiently. Check the CPU and memory usage for every piece of code. Be familiar with APM, coding centered languages, and DevOps culture and methodologies. Know the stress levels for the apps you are building. Think about how users will be accessing and using the application or service, and be willing to adjust to the constantly shifting technology landscape.

TOM SMITH is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.

Solutions Directory

This directory of monitoring, hosting, and optimization services provides comprehensive, factual comparisons of data gathered from third-party sources and the tool creators' organizations. Solutions in the directory are selected based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	HOSTING	WEBSITE
Akamai	ION	CDN, network & mobile monitoring & optimization, FEO	Available by request	SaaS	akamai.com/us/en/solutions/products/web-performance/web-performance-optimization.jsp
Apica	Apica Systems	APM, infrastructure monitoring	30 days	SaaS	apicasystems.com
AppDynamics	AppDynamics	APM, mobile & web RUM, database monitoring, infrastructure visibility	15 days	On-premise or SaaS	appdynamics.com
AppNeta	AppNeta	APM, synthetic monitoring, network monitoring, ITOA, real user monitoring	Available by request	SaaS	appneta.com/overview
Appnomic Systems	AppsOne	ITOA	Demo available by request	On-premise or SaaS	appnomic.com/appsone
Riverbed	SteelCentral Aternity	APM, ITOA, real user monitoring	Free tier available	On-premise	riverbed.com/products/steelcentral/steelcentral-aternity.html
BigPanda	BigPanda	ITOA, alert software	21 days	SaaS	bigpanda.io/product
BMC	BMC TrueSight Pulse	APM, network monitoring, ITOA, database monitoring	14 days	SaaS	bmc.com/truesightpulse
BrowserStack	BrowserStack	FEO	Available by request	SaaS	browserstack.com
Bugsnag	Bugsnag	Application montioring	14 days	On-premise or SaaS	bugsnag.com/product
CA	CA App Synthetic Monitor	APM, synthetic monitoring	30 days	SaaS	ca.com/us/products/ca-app-synthetic-monitor.html
CA	CA App Experience Analytics	Mobile APM	30 days	SaaS	ca.com/us/products/ca-app-experience-analytics.html
CA	CA Unified Infrastructure Management	Infrastructure Monitoring	30 days	On-premise	ca.com/us/products/ca-unified-infrastructure-management.html

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	HOSTING	WEBSITE
Catchpoint	Catchpoint Suite	Synthetic, RUM, UEM	14 days	On-premise or SaaS	catchpoint.com
Citrix	Cedexis Impact	Infrastructure monitoring, FEO, ITOA	Available by request	SaaS	cedexis.com/impact
Circonus	Circonus	Infrastructure monitoring, ITOA	Demo available by request	SaaS	circonus.com
CloudFlare	CloudFlare	CDN, network, mobile, web monitoring & optimization, FEO	Free tier available	CDN	cloudflare.com
Correlsense	SharePath	APM, network monitoring, middleware monitoring	Free tier available	On-premise or SaaS	correlsense.com/product
CoScale	CoScale	APM, infrastructure monitoring, ITOA, real user monitoring	30 days	SaaS	coscale.com
Datadog	Datadog	Performance metrics integration & analysis	14 days	SaaS	datadoghq.com/product
Dotcom-Monitor	Dotcom-Monitor	APM, infrastructure monitoring, FEO	30 days	SaaS	dotcom-monitor.com/web-monitoring-solutions
Dyn	Dyn	Infrastructure monitoring, network monitoring, ITOA	Demo available by request	On-premise	dyn.com
Dynatrace	Dynatrace Application Monitoring	APM, ITOA	15 days	On-premise	dynatrace.com/solutions/application-monitoring
Dynatrace	Dynatrace Data Center RUM	RUM (web & non-web), synthetic, ITOA	Demo available by request	On-premise	dynatrace.com/platform/offerings/data-center-rum
Dynatrace	Dynatrace Managed	APM (cloud-native optimized)	15 days	On-premise or SaaS	dynatrace.com/trial/managed
Dynatrace	Dynatrace Synthetic	Synthetic monitoring, managed load testing	15 days	SaaS	dynatrace.com/capabilities/synthetic-monitoring
Dynatrace	Dynatrace UEM	Real user monitoring (web & mobile)	30 days	On-premise	dynatrace.com/platform/offerings/user-experience-management
eG Enterprise	eG Enterprise	APM & RUM for web applications, Citrix, middleware, DB, virtualization, analytics, reporting	Free tier available	On-premise or SaaS	eginnovations.com
Evolven	Evolven	ITOA	Demo available by request	On-premise	evolven.com
ExtraHop Networks	ExtraHop Networks	ITOA	Demo available by request	SaaS	extrahop.com/platform
f5	Big-IP Platform	APM, network monitoring	Available by request	On-premise or SaaS	f5.com/products/big-ip
Fortinet	FortiSIEM	ITOA, network monitoring	Demo available by request	SaaS	fortinet.com/products/management/fortisiem.html
FusionReactor	FusionReactor	Java server monitor, production debugging, crash protection	14 days	On-premise	fusion-reactor.com

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	HOSTING	WEBSITE
Micro Focus	APM	APM, ITOA, real user monitoring	Available by request	On-premise	software.microfocus.com/en-us/solutions/application-performance-management
HPE	StormRunner Load	Load testing	30 days	SaaS	software.microfocus.com/en-us/products/stormrunner-load-agile-cloud-testing/overview
IBM	IBM API Connect	API management platform	Free tier available	On-premise or SaaS	ibm.com/cloud/api-connect
IBM	IBM Cloud Application Performance Management	APM, infrastructure monitoring, real user monitoring	30 days	On-premise or SaaS	ibm.com/us-en/marketplace/application-performance-management
Idera	SQL Diagnostic Manager	DB monitoring	14 days	SaaS	idera.com/productssolutions/sqlserver/sqldiagnosticmanager
Idera	Uptime Infrastructure Monitor	APM, infrastructure monitoring, network monitoring	30 days	SaaS	idera.com/it-infrastructure-management-and-monitoring
InfoVista	5view Applications	APM, network monitoring, real user monitoring	Demo available by request	On-premise	infovista.com/products/Application-Performance-Monitoring-and-Management
INTECO	INTECO Insight	APM, middleware monitoring	Demo available by request	On-premise	inetco.com/products/inetco-insight
jClarity	Censum	JVM garbage collection optimization	7 days	On-premise or SaaS	jclarity.com/censum
jClarity	Illuminate	JVM performance diagnosis & optimization	14 days	On-premise or SaaS	jclarity.com/illuminate
JenniferSoft	Jennifer	APM	14 days	On-premise	jennifersoft.com/en/product/product-summary
Librato	Librato	Performance metrics integration & analysis	30 days	SaaS	librato.com/product
LiveAction	LiveNX	Network monitoring & diagnostics	14 days	SaaS	liveaction.com/solutions/liveaction-network-performance-management
Logentries	Logentries	Log management & analytics	Free tier available	SaaS	logentries.com
Loggly	Loggly	Log management & analytics	30 days	SaaS	loggly.com
LogMatrix	NerveCenter	ITOA, APM, infrastructure monitoring, network monitoring, database monitoring	N/A	On-premise	logmatrix.com/nervecenter-for-network-monitoring
ManageEngine	ManageEngine Applications Manager	APM, network monitoring, infrastructure monitoring	30 days	On-premise	manageengine.com/products/applications_manager
Microsoft	System Center	APM	180 days	On-premise	microsoft.com/en-us/cloud-platform/system-center

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	HOSTING	WEBSITE
Monitis	Monitis	Network & IT systems monitoring	15 days	On-premise or SaaS	monitis.com
Moogsoft	AIOps	IT incident management, AI	Available by request	On-premise or SaaS	moogsoft.com/product
Nagios	Nagios XI	APM, infrastructure monitoring, network monitoring, FEO, ITOA	60 days	On-premise	nagios.com/products/nagios-xi
Nastel	AutoPilot	APM, infrastructure monitoring, FEO, middleware monitoring	Demo available by request	SaaS	nastel.com/application-performance-monitoring
NetScout	nGeniusONE	APM, network monitoring, ITOA	N/A	On-premise	netscout.com/product/service-provider/ngeniusone
Metricly	Metricly	APM, infrastructure monitoring, ITOA	21 days	SaaS	metricly.com
Neustar	Neustar Website Monitoring	FEO	15 days	SaaS	security.neustar/web-performance-management/website-monitoring
New Relic	New Relic APM	APM, database monitoring, availability & error monitoring, reports, team collaboration, security	14 days	SaaS	newrelic.com/application-monitoring
op5	op5 Monitor	APM, infrastructure monitoring, network monitoring, FEO, ITOA	Open source	SaaS	op5.com/op5-monitor
OpsGenie	OpsGenie	Alert software	Available by request	On-premise	opsgenie.com
Opsview	Opsview	APM, network monitoring, ITOA	60 days	On-premise	opsview.com/products
Outlyer	Outlyer	Infrastructure monitoring	14 days	SaaS	outlyer.com/features
PagerDuty	PagerDuty	ITOA, alert software	Available by request	SaaS	pagerduty.com
Pingdom	Pingdom	APM, FEO	14 days	SaaS	pingdom.com/product
Power Admin	PA Server Monitor	Infrastructure monitoring, network monitoring	30 days	On-premise	poweradmin.com/products/server-monitoring
Quest	Foglight	APM, database monitoring, RUM, ITOA	Available by request	On-premise	quest.com/foglight
Rackspace	Rackspace Monitoring	Cloud monitoring	Free tier available	SaaS	rackspace.com/cloud/monitoring
Rigor	Rigor	Performance monitoring & optimization, RUM	Available by request	On-premise or SaaS	rigor.com
Riverbed	SteelCentral	APM, infrastructure monitoring, network monitoring, ITOA	30-90 days	On-premise	riverbed.com/products/steelcentral
Runscope	Runscope	API testing & monitoring	14 days	Private	runscope.com

COMPANY	PRODUCT	CATEGORY	FREE TRIAL	HOSTING	WEBSITE
Sauce Labs	Sauce Labs	FEO, automated web & mobile testing	14 days	SaaS	saucelabs.com
ScienceLogic	ScienceLogic Platform	APM, infrastructure monitoring,network monitoring	N/A	SaaS	sciencelogic.com/product
Sentry	Sentry	Error monitoring	Free tier available	On-premise	sentry.io/features
SevOne	SevOne	Infrastructure monitoring, network monitoring	Demo available by request	SaaS	sevone.com
SmartBear Software	AlertSite	APM, synthetic monitoring, infrastructure monitoring, middleware monitoring	Available by request	On-premise or SaaS	smartbear.com/product/alertsite/overview
SOASTA	SOASTA Platform	Real user monitoring, load testing	Available by request	SaaS	soasta.com/videos/soasta-platform-overview
SolarWinds Worldwide	Network Performance Monitor	Network monitoring, ITOA, database monitoring, log management	30 days	On-premise	solarwinds.com/network-performance-monitor
SpeedCurve	SpeedCurve	FEO, ITOA	30 days	SaaS	speedcurve.com
Spiceworks	Spiceworks	Network monitoring, ITOA	Free tier available	On-premise	spiceworks.com
Stackify	Retrace	APM, network monitoring, database monitoring, ITOA	14 days	SaaS	stackify.com
Sysdig	Sysdig Monitor	Application montioring	14 days	On-premise or SaaS	sysdig.com
TeamQuest	TeamQuest	ITOA	Available by request	On-premise	teamquest.com
ThousandEyes	ThousandEyes	Network monitoring, ITOA	Available by request	SaaS	thousandeyes.com
Unravel Data	Unravel	Application montioring	Available by request	On-premise or SaaS	unraveldata.com
VictorOps	VictorOps	Alert software	14 days	On-premise	victorops.com
Virtual Instruments	WorkloadWisdom	Metrics monitoring & analytics	Demo available by request	SaaS	virtualinstruments.com
Wavefront	Wavefront	Metrics monitoring & analytics	30 days	SaaS	wavefront.com
xMatters	xMatters	IT communication, performance data analytics	14 days	On-premise	xmatters.com/products
Zabbix	Zabbix	Network monitoring	Open source	On-premise	zabbix.com
Zenoss	Zenoss	Infrastructure monitoring, network monitoring	Demo available by request	On-premise or SaaS	zenoss.com/product
Zoho Corporation	Site24x7	APM, FEO, infrastructure monitoring, network monitoring	30 days	SaaS	site24x7.com

GLOSSARY

APPLICATION PROGRAMMING INTERFACE (API): A set of definitions, protocols, and tools that specify how software components should interact; the building blocks for modern web and mobile applications.

APPLICATION PERFORMANCE MONITORING (APM): Combines metrics on all factors that might affect application performance (within an application and/or web server, between database and application servers, on a single machine, between client and server, etc.); usually (but not always) higher-level.

BENCHMARK: A set of tests that run against a particular software or piece of software that is used for gathering data and assessing the performance of what's being tested.

CONTENT DELIVERY NETWORK (CDN): Geographically and topologically distributed servers that cache content (often high-bandwidth static content, like videos, documents, or other large binaries) to minimize unnecessary transport and backbone overload.

CURL: An open source command line tool and library for transferring data with URLs.

DEEP API MONITORING: The ability to monitor a series of API endpoints invoked in sequence with contextual data passed from one call to the next; involves measuring availability, performance, and functional correctness.

DISTRIBUTED DENIAL OF SERVICE (DDOS): A malicious attack on an application consisting of superfluous traffic designed to render the service unusable.

ELASTIC SATCK: Formerly known as the ELK stack, it is a combination of Elasitcsearch, Logstash, Kibana, and Beats to ingest and process data from several different sources.

FUNCTIONAL TESTING: The process of testing if a system meets functional requirements and specifications.

HTTP HEADERS: Fields that can be added to HTTP requests and responses, allowing the client and the server to pass additional information.

HTTP METHODS: Indicates the desired action to be performed on an endpoint. GET and POST are the most commonly used. Also referred to as HTTP verbs.

KEY PERFORMANCE INDICATOR (KPI): A set of indicators to measure data or performance against a particular set of standards or requirements.

LOAD TESTING: The process of measuring a software system's response when handling a specified load.

LOG: Contains textual information about an event; a record of activities performed by an electronic device that is automatically created and maintained by another device; meant to convey detailed information about the application, user, or system activity and troubleshoot a specific issue after the fact.

METRIC: Numeric measurements in time. The format includes the measured metric name, metric data value, timestamp, metric source, and optional tag and are meant to convey small information bits.

MICROSERVICES ARCHITECTURE: A development method of designing your applications as modular services that seamlessly adapt to a highly scalable and dynamic environment.

OAUTH 2: An open standard for authorization, commonly used by APIs that provide access to user information.

PEAK PERFORMANCE: Maximum theoretical performance that can be achieved by software.

REAL-WORLD PERFORMANCE: The actual performance of software when it's tested in real-life circumstances as opposed to theoretical performance that measures against the ideal.

SEQUENCED API: API transactions that are invoked by web and mobile applications in sequence with contextual data passed from one call to the next.

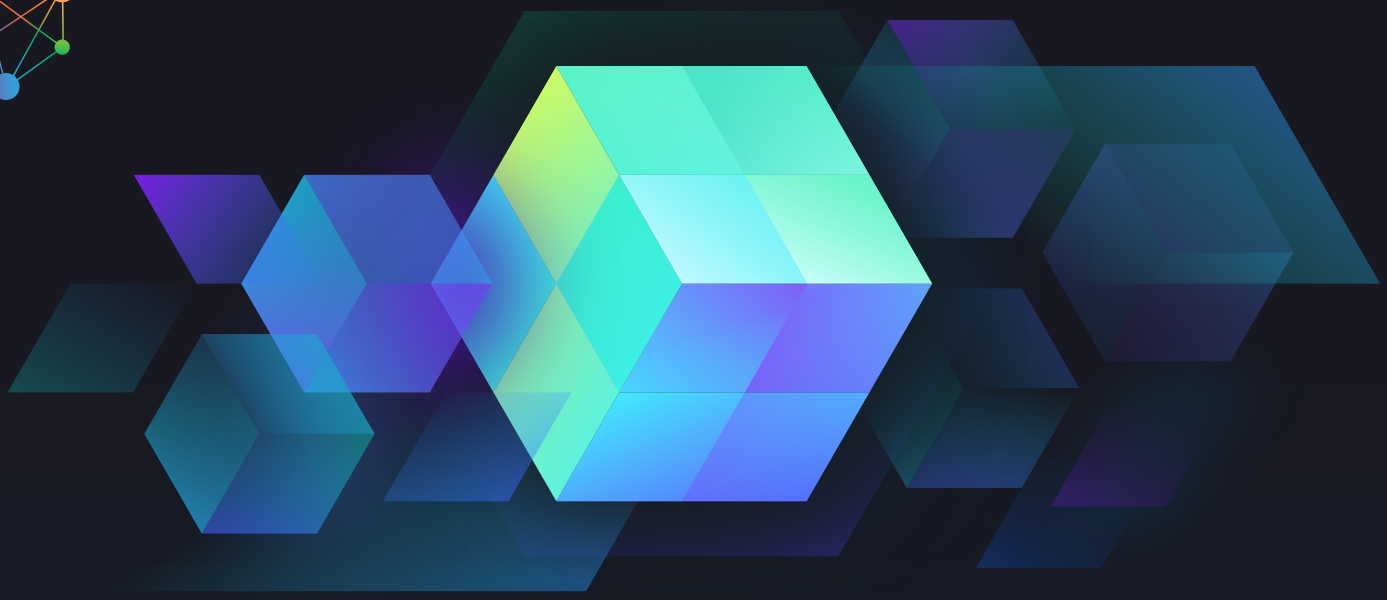
SERVICE LEVEL AGREEMENT (SLA): A contractual set of expectations for levels of service—like monthly uptime—between the cloud buyer (customer) and cloud provider (seller).

"SHIFT LEFT" TESTING: An approach in which software testing is performed earlier in the software development lifecycle—conducive with but not exclusive to Agile and DevOps methodologies.

SPRING FRAMEWORK: An open source collection of tools for building web applications in Java.

TRANSACTION JOURNAL: Refers to the simultaneous, real-time logging of all data updates in a database. The resulting log functions as an audit trail that can be used to rebuild the database if the original data is corrupted or deleted.

TRANSACTION MERGING: The queuing of operations to travel as a group to the disk instead of calling the journal directly for each operation.



INTRODUCING THE

Microservices Zone

Start Implementing Your Microservice Architecture and Scale Your Applications

Whether you are breaking down existing monolithic legacy applications or starting from scratch, see how to overcome common challenges with microservices.

Keep a pulse on the industry with topics like: Java Microservices Tutorials and Code Examples, Best Practices for a REST-Based Microservices Architecture, and Patterns and Anti-Patterns to Get Started.

Visit the Zone

REST-BASED MICROSERVICE ARCHITECTURE

PATTERNS AND ANTI-PATTERNS

INTER-SERVICE COMMUNICATION

APPLICATIONS FOR CONTAINERS