

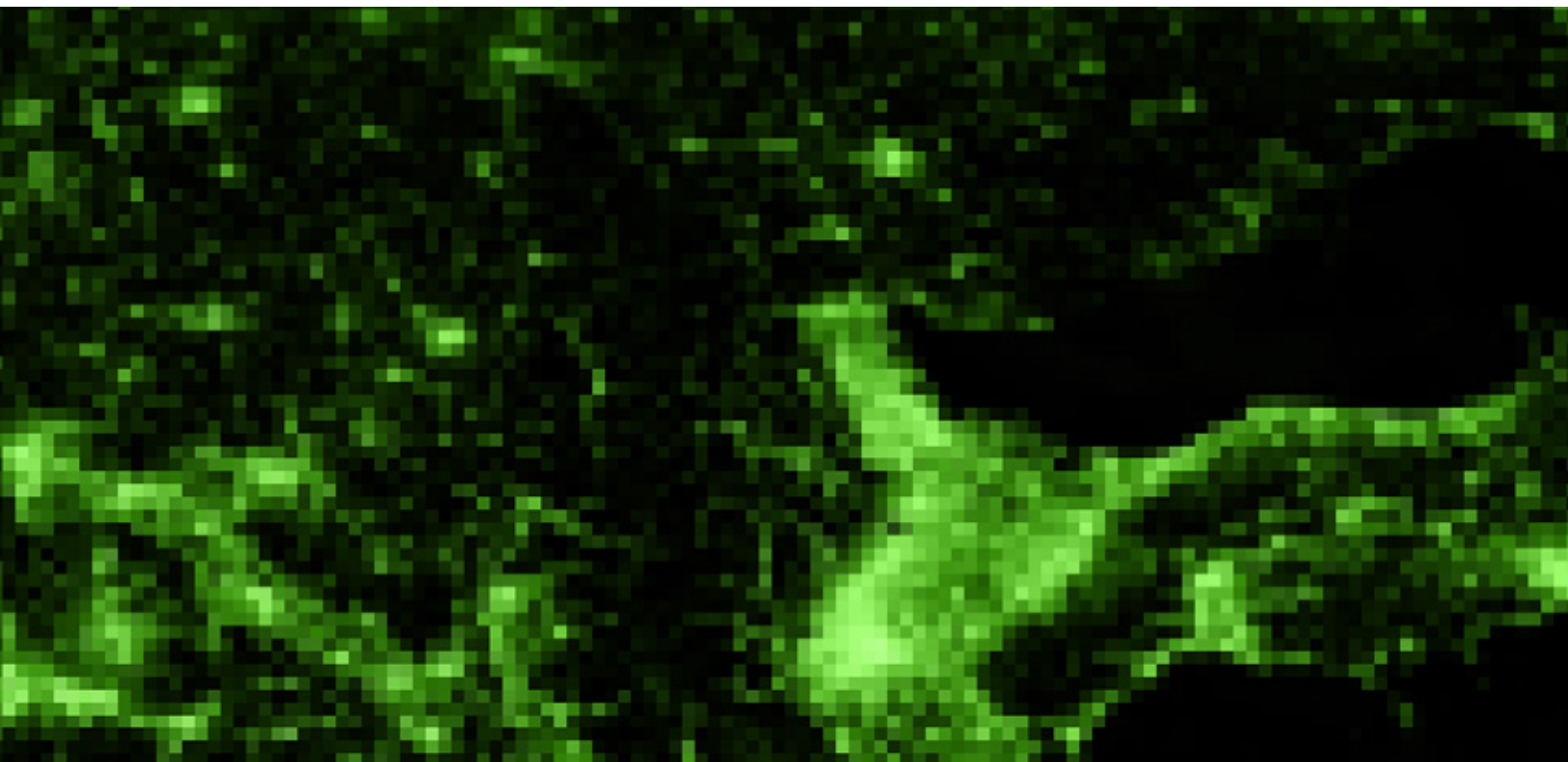
STACKIFY RETROSPECTIVE

.NET Monitoring Decrypted:

Code Level Performance Metrics, Azure Monitoring, Structured Logging and More

Table of Contents

<u>Key Sources of .NET Monitoring</u>	4
<u>Measuring Application Performance Metrics</u>	8
<u>Monitor the Status of Your IIS Application</u>	12
<u>Defining Application Dependency Mapping and Performance</u>	15
<u>Monitor Your Azure Apps and Servers</u>	19
<u>Understanding the Four Different IIS Logs</u>	23
<u>Methods to Find Slow SQL Queries</u>	27
<u>Catching Exceptions in C# and Finding Application Errors</u>	35
<u>A Retrospective on Structured Logging</u>	42





Introduction

At Stackify, we build with developers in mind. After all, who gets called when a critical application stops working over the weekend? Developers do.

It's no secret that being on call is no fun. But at least when you have amazing application monitoring in place, it can be less stressful. Today's applications rely on multiple services like SQL databases, external web services, queues, caching, cloud providers, and much more.

Fortunately, most all of these things are built for high availability. Unfortunately, minor application problems are still a reality, not to mention the occasional outage. When application problems do arise, developers need to find the problem quickly so they can go back to their weekend or finish their work items for the sprint.

In our Retrospective series, we pool together all our knowledge and experience on the most relevant and integral topics in the community and deliver them to developers. In our .NET Monitoring Decrypted guide, we share the lessons we've learned, from the best ways to find slow SQL queries to our full commitment to structured logging. We've done the research so you can do the coding.

Key Sources of .NET Monitoring

Application Monitoring is Different for Developers

Developers need several key types of data:

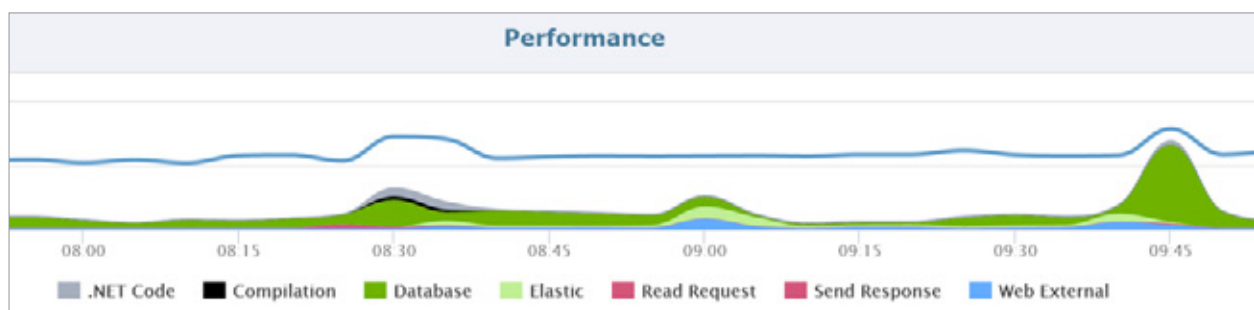
1. **Application Performance Management (APM):** Code level application performance visibility
2. **Transaction Tracing:** Code level traces of what your code is doing
3. **Metrics:** Server, application, and custom metrics monitoring
4. **Logs:** Aggregation, searching, and management
5. **Errors:** Aggregation, reporting & alerting

1. APM: Code level application performance visibility

APM solutions collect very detailed data via code profiling and transaction tracing capabilities. All of that data is aggregated and crunched to provide powerful reporting and alerting capabilities. With APM, you can quickly identify why your application is slow or not working properly.

Key APM features:

- Monitor app performance by response times, user satisfaction, etc.
- Understand performance based on application dependencies
- Identify slowest and most used web requests
- Monitor key web requests or transactions
- Identify slowest and most used SQL queries
- Monitor the performance of specific SQL queries



Track Performance of Application Dependencies

2. Transaction Tracing: Code level traces of what your code is doing

At the heart of developer application monitoring is the ability to collect very detailed performance data and transaction traces for your applications.

Traces contain these types of data:

- Web request info like URL, etc.
- Key methods in your code
- What dependencies did your code call (SQL, caching, HTTP calls, etc.)
- Application errors
- Logging statements

The screenshot displays a transaction trace for a GET request to /stackify/devices. The trace is annotated with numbered circles 1 through 7, corresponding to the legend on the right. The events and their durations (in ms) are as follows:

Event	From (ms)
Request Started GET /stackify/devices	0
Primary Handler (ASP.NET MVC)	3
DEBUG Setting #redis {"key":"LastGetTopPeopleCalled"}	3
RedisClient.Add stackifydevcentral.redis.cache.windows.net LastGetTopPeopleCalled	3
Caught System.InvalidOperationException	5
System.InvalidOperationException: This error gets thrown all the time and thrown away causing CPU problems at HelloStackify.Web.Controllers.StackifyController.Devices	5
DEBUG Query people for != Mr. and get the category list	5
SqlConnection.Open prefixdemo.database.windows.net adventureworks2012	5
SqlCommand.ExecuteReader SELECT Customer, Records: 42	5
SqlConnection.Open prefixdemo.database.windows.net adventureworks2012	8
SqlCommand.ExecuteReader SELECT ProductCategory, Records: 41	8
SELECT TOP (100) [c].[ProductCategoryID] AS [ProductCategoryID],[c].[ParentProductCategoryID] AS [ParentProductCategoryID],[c].[Name] AS [Name],[c].[rowguid] AS [rowguid],[c].[ModifiedDate] AS [ModifiedDate] FROM [SalesLT].[ProductCategory] AS [c]	9
DEBUG Querying Stackify for device list and return it	9
Invoke ActionResult	19
JsonResult.ExecuteResult	19
Request Completed Status: 200	19

Legend:

1. Request Details
2. Logging
3. Redis Cache Call
4. Error in Code
5. SQL Queries
6. Web Request Status
7. Repsonse Time (milliseconds)

Transaction Trace in Stackify Prefix

3. Metrics: Server, Application, and Custom

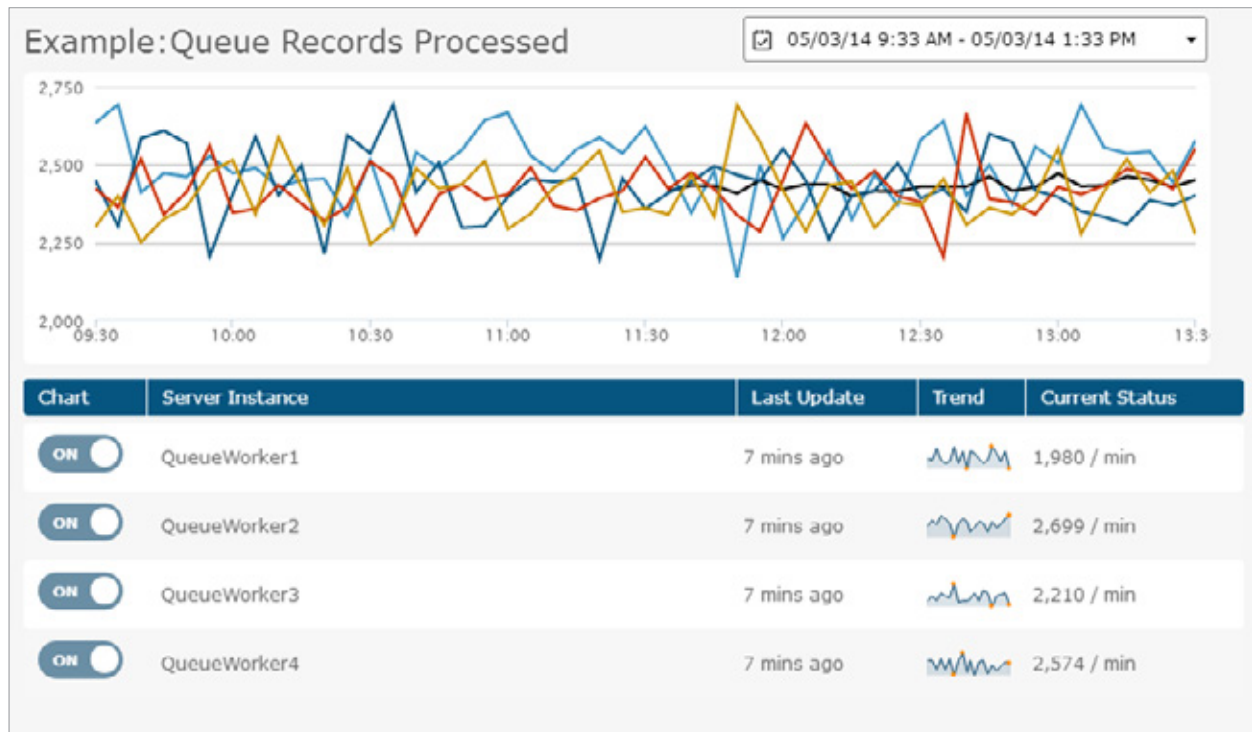
It is important to monitor basic server metrics like CPU, memory, network, and disk performance. Developers can also monitor their applications and track things like garbage collection, request queuing, transaction volumes, page load times, and much more.

Some of the types of application metrics you can monitor:

- Windows Performance Counters
- Custom application metrics
- Error rates
- Custom metrics via Retrace API

Use charting and dashboards to easily analyze and trend your monitoring metrics:

- Built-in dashboards
- Interactive charting
- Easily compare metrics across multiple servers
- See trends at a glance with sparklines



Monitor Server and Application Metrics



"Until Stackify, there was no one tool to do monitoring, performance dashboards, log aggregation, APM and more.

This same technology would come with half a dozen or more products and even more expense. Stackify provides all of this functionality out of the box, in one package, and provides you the actionable insights necessary to target problems when and where they happen."

—Jake Dubin, VeriShip



APM



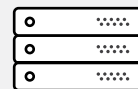
LOGS



ERRORS



MONITORING



QA/PRE-PROD



PRODUCTION

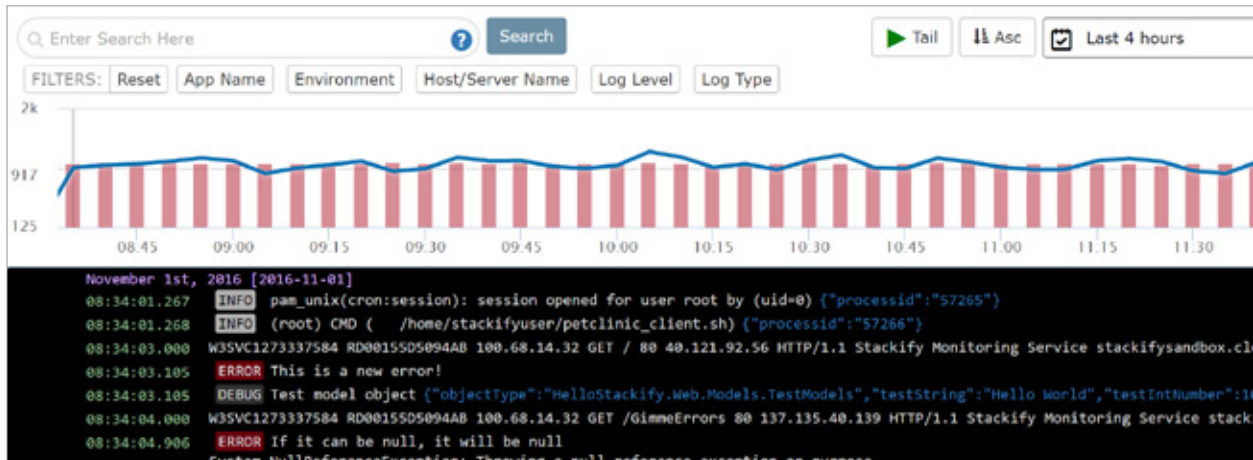
STARTING AT ONLY

\$99/mo

4. Logs: Aggregation, Searching, and Monitoring

Log management enables you to centralize all of your application and server logs in one place. Full text searching and support for structured logging makes it easy to find anything you want to see in your logs. You can even setup log searches to run every few minutes and alert your team if anything is found.

Application logs are supported by common logging frameworks like log4net, NLog, log4j, logback and others. Server logs from syslog, Windows Events, and web server access logs are also supported.








Search Application Logs

5. Errors: Aggregation, Reporting, and Alerting

Stop logging your errors to log files that you don't even look at. Let Retrace collect all of your exceptions and send you an email when new errors are found. Exceptions are uniquely identified and alerts can be setup based on error rates per application.

With Retrace, you can even see all the logging statements that occurred in the same web request and potentially see the complete transaction trace that was recorded. Being able to go from an error to understanding exactly what was going on is a little magical.

 17	ServiceStack.Redis.RedisException: could not connect to redis Instance at stackifydevcentral.redis.cache.windows.net:6379 ServiceStack.Redis.RedisNativeClient.Connect GET-stackify.devices
 144	System.InvalidOperationException: This error gets thrown all the time and thrown away causing CPU problems HelloStackify.Web.Controllers.StackifyController.Devices GET-stackify.devices
 506	java.lang.RuntimeException: Expected: controller used to showcase what happens when an exception is thrown CrashController.triggerException() GET-CrashController.triggerException
 369	System.NullReferenceException: Object reference not set to an instance of an object. Glimpse.Core.Framework.GlimpseConfiguration.<GetConfiguredTimerStrategy>b__15 GET-sqltest.eftest
 241	System.NullReferenceException: Throwing a null reference exception on purpose. (If it can be null, it will be null {\"isNull\": \"of course it is\"}) GimmeErrorsController.Index() GET-GimmeErrors.Index

Monitor Unique Application Errors

Measuring Application Performance Metrics

We spend a lot of time at Stackify thinking about application performance, especially about how to monitor and improve it. In this article, we are covering some of our most suggested key application performance metrics.

Key Application Performance Metrics

1. User Satisfaction / Apdex Scores

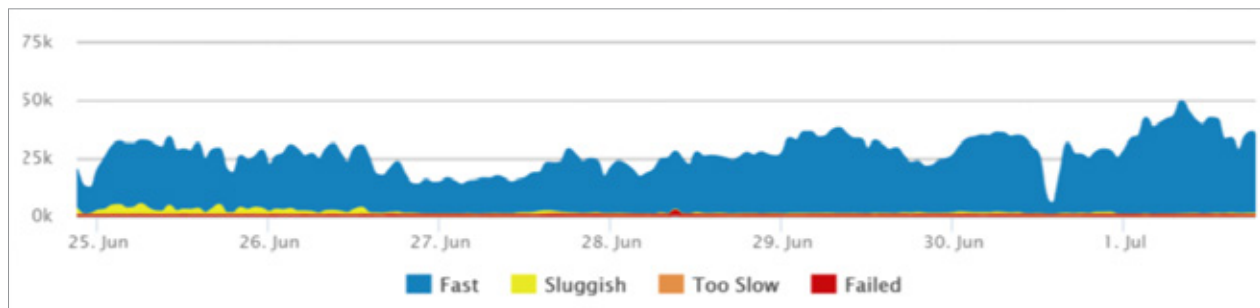
The application performance index, or Apdex score, has become an industry standard for tracking the relative performance of an application.

It works by specifying a goal for how long a specific web request or transaction should take.

Those transactions are then bucketed into satisfied (fast), tolerating (sluggish), too slow, and failed requests. A simple math formula is then applied to provide a score from 0 to 1.

$$Apdex_t = \frac{SatisfiedCount + \frac{Tolerating\ Count}{2}}{TotalSamples}$$

Retrace automatically tracks satisfaction scores for every one of your applications and web requests. We convert the number to a 0-100 instead of 0-1 representation to make it easier to understand.

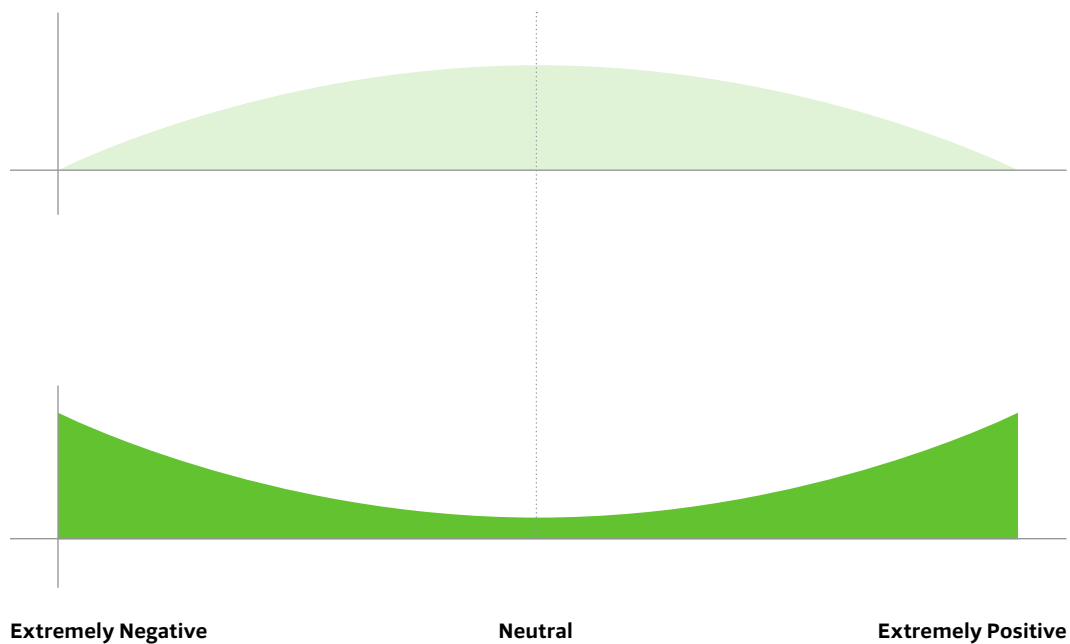


Retrace Satisfaction Chart

2. Average Response Time

Let's start by saying that averages are terrible. We highly recommend using the aforementioned user satisfaction Apdex scores as a preferred way to track overall performance. That said, averages are still a useful application performance metric.

The illustration below shows two graphs with the same average:



3. Error Rates

The last thing you want your users to see is an error. Monitoring error rates is a critical application performance metric.

There are potentially 3 different ways to track application errors:

- **HTTP Error %:** Number of web requests that ended in an error
- **Logged Exceptions:** Number of unhandled and logged errors from your application
- **Thrown Exceptions:** Number of all exceptions that have been thrown

It is common to see thousands of exceptions being thrown and ignored within an application. Hidden application exceptions can cause a lot of performance problems.

4. Count of Application Instances

If your application scales up and down in the cloud, it is important to know how many server/application instances you have running. Auto-scaling can help ensure your application scales to meet demand and saves you money during off-peak times. This also creates some unique monitoring challenges.

For example, if your application automatically scales up based on CPU usage, you may never see your CPU get high. You would instead see the number of server instances get high. (Not to mention your hosting bill going way up!)

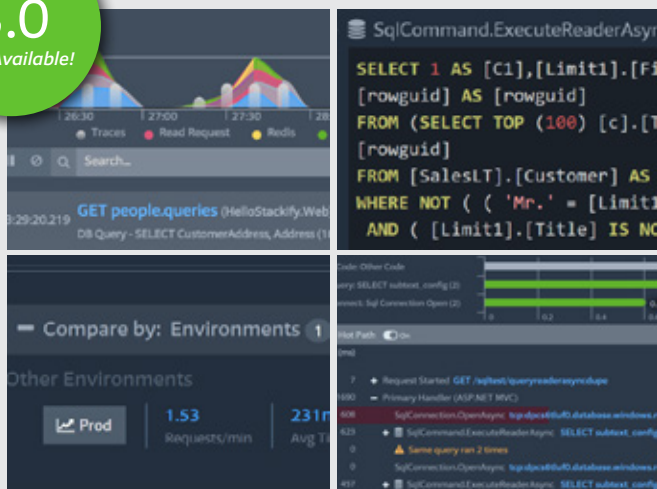
5. Request Rate

Understanding how much traffic your application receives is a critical metric. Potentially, all other application performance metrics are affected by increases or decreases in traffic.

Request rates can be useful to correlate to other application performance metrics to understand the dynamics of how your application scales.

3.0

Now Available!



"Prefix has helped me so much in writing better code. It's helped reduce calls to our database and to identify exactly why a few pages were taking so long to load. It is a great product and I am looking forward to seeing what Stackify comes up with next."

—Patrick McCarthy, Andrews McMeel Universal

Prefix Logs + Errors + Queries + More

Monitoring the request rate can also be good to watch for spikes or even inactivity. If you have a busy API that suddenly gets no traffic at all, that could be a really bad thing to watch out for.

A similar but slightly different metric to track is the number of concurrent users. This is another interesting metric to track to see how it correlates.

6. Application & Server CPU

If the CPU usage on your server is extremely high, you can guarantee you will have application performance problems. Monitoring the CPU usage of your server and applications is a basic and critical metric.

Virtually all server and application monitoring tools can track your CPU usage and provide monitoring alerts. It is important to track them per server but also as an aggregate across all the individually deployed instances of your application.

7. Application Availability

Monitoring and measuring if your application is online and available is a key metric you should be tracking. Most companies use this as a way to measure uptime for service level agreements (SLA).

If you have a web application, the easiest way to monitor application availability is via a simple scheduled HTTP check.

Retrace can run these types of HTTP “ping” checks every minute for you. It can monitor response times, status codes, and even look for specific content on the page. Retrace can use this to report application availability for you.

8. Garbage Collection

If your application is written in .NET, C#, or other programming languages that use garbage collection, you are probably aware of the performance problems that can arise from it.

When garbage collection occurs, it can cause your process to suspend and can use a lot of CPU.

Garbage collection metrics may not be one of the first things you think about key application performance metrics. It can be a hidden performance problem that is always a good idea to keep an eye on.

For .NET, you can monitor this via the Performance Counter of “% GC Time.” Java has similar capabilities via JMX metrics. Retrace can monitor these via its application metrics capabilities.

Summary

This list provides a sound basis for metrics that you should be monitoring. Depending on your type of application, there could be many other monitoring needs. Application performance measurement is necessary for all types of applications.

Monitor the Status of Your IIS Application






Let's cover the basics including HTTP ping checks, IIS Application Pools, and important Windows Performance Counters. We'll also take a look at how to use an application performance management system to simplify all of this and get more advanced IIS performance monitoring for ASP.NET applications..

Website Monitor via HTTP Testing

One of the best and easiest things you can do is set up a simple HTTP check that runs every minute. This will give you a baseline to know if your site is up or down. It can also help you track how long it takes to respond. You could also monitor for a 200 OK status or if the request returns specific text that you know should be included in the response.

Monitoring IIS via a simple HTTP check is also a good way to establish a basic SLA monitor. No matter how many servers you have, you can use this to know if your web application was online and available.

Here is an example of one of our HTTP checks we use against Elasticsearch to help with monitoring it. We do this via Retrace; you could also use tools like Pingdom. In this example, we receive alerts if the `number_of_nodes` is not what we are expecting or if it doesn't find an HTTP status of 200 OK.

Web Site (Internal): http://localhost/_cluster/health?number_of_nodes (from es-util): es-write (S1 Prod) Dashboard		Response Time	Status Code
		 0.072s	 200 - OK
Description	Last Update	Trend	Status
Response Time	5 mins ago		0.079s
Result String	5 mins ago		1 match of "number_of_nodes":19
Status Code	5 mins ago		200 - OK

Stackify Retrace

Ensure Your IIS Application Pool is Running

If you have been using IIS very long, you have probably witnessed times when your application mysteriously stops working. After some troubleshooting, you may find that your IIS Application Pool is stopped for some reason, causing your site to be offline.

Sometimes an IIS Application Pool will crash and stop due to various fatal application errors, issues with the user the app pool is running under, bad configurations, or other random problems. It is possible to get it into a state where it won't start at all due to these type of problems.

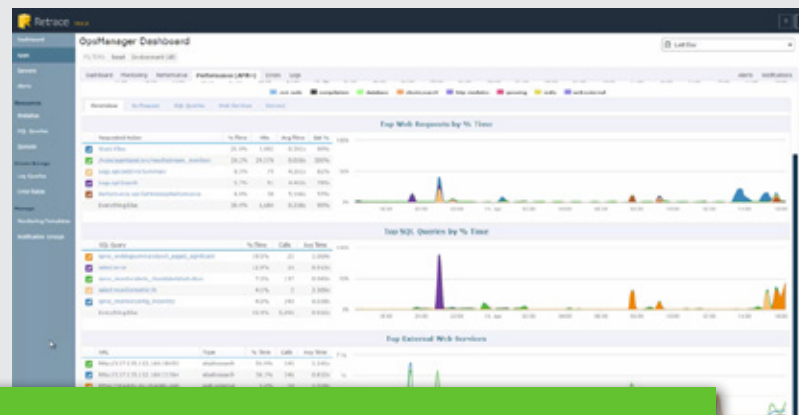
It is a good best practice always to monitor that your IIS Application Pool is started. It runs as w3wp.exe. Most monitoring tools have a way to monitor IIS Application Pools. Our product, Retrace, monitors them by default.

One weird thing about app pools is they can be set to "Started" but may not actually be running as w3wp.exe if there is no traffic to your application. In these scenarios, w3wp.exe may not be running, but there is no actual problem. This is why you need to monitor it via IIS's status and not just look for w3wp.exe to be running on your server.



Retrace is free for 14-days, then starts at only \$99/month.

APM + Errors + Logs + Monitoring in one killer tool – QA & production!



"Retrace quickly and easily finds the root cause of the issue. We saw a crash report in production and we resolved it right away. By the time a ticket came in, we were ready to push the fix. It's very good to be proactive on these errors."

–Syad, Paycor

Recommended Performance Counters for IIS Monitoring

One of the advantages of using IIS as a web server is all of the metrics available via Windows Performance Counters. There is a wide array of them available between IIS, ASP.NET and .NET. For this guide on IIS performance monitoring, let's review some of the top Performance Counters to monitor.

To simplify things a bit, we are going to split the Performance Counters up between IIS and ASP.NET particular Performance Counters. All of these are monitored by default as part of Retrace's application metrics monitoring.

System/Process Counters

- **CPU %:** The overall server and CPU usage for your IIS Worker Process should be monitored.
- **Memory:** You should consider tracking the currently used and available memory for your IIS Worker Process.

IIS Performance Counters

- **Web Service—Bytes Received/Sec:** Helpful to track to identify potential spikes in traffic.
- **Web Service—Bytes Sent/Sec:** Helpful to track to identify potential spikes in traffic.
- **Web Service—Current Connections:** Through experience with your app you can identify what is a normal value for this.

ASP.NET Performance Counters

- **ASP.NET Applications—Requests/Sec:** You should track how many requests are handled by both IIS and ASP.NET. Some requests, like static files, could only be processed by IIS and never touch ASP.NET.
- **ASP.NET Applications—Requests in Application Queue:** If this number is high, your server may not be able to handle requests fast enough.
- **.NET CLR Memory—% Time in GC:** If your app spends more than 5% of its time in garbage collection, you may want to review how object allocations are performed.

ASP.NET Error Rate Counters

- **.NET CLR Exceptions—# of Exceps Thrown:** This counter allows you track all .NET exceptions that are thrown even if they are handled and thrown away. A very high rate of exceptions can cause hidden performance problems.
- **ASP.NET Applications—Errors Unhandled During Execution/sec:** The number of unhandled exceptions that may have impacted your users.
- **ASP.NET Applications—Errors Total/Sec:** Number of errors during compilations, pre-processing and execution. This may catch some types of errors that other exception counts don't include.

You should be able to monitor these Windows Performance Counters with most server monitoring solutions.

Note: Some Windows Performance Counters are difficult to monitor because the process name or ID changes constantly. You may find it hard to monitor them in some server monitoring solutions due to this.

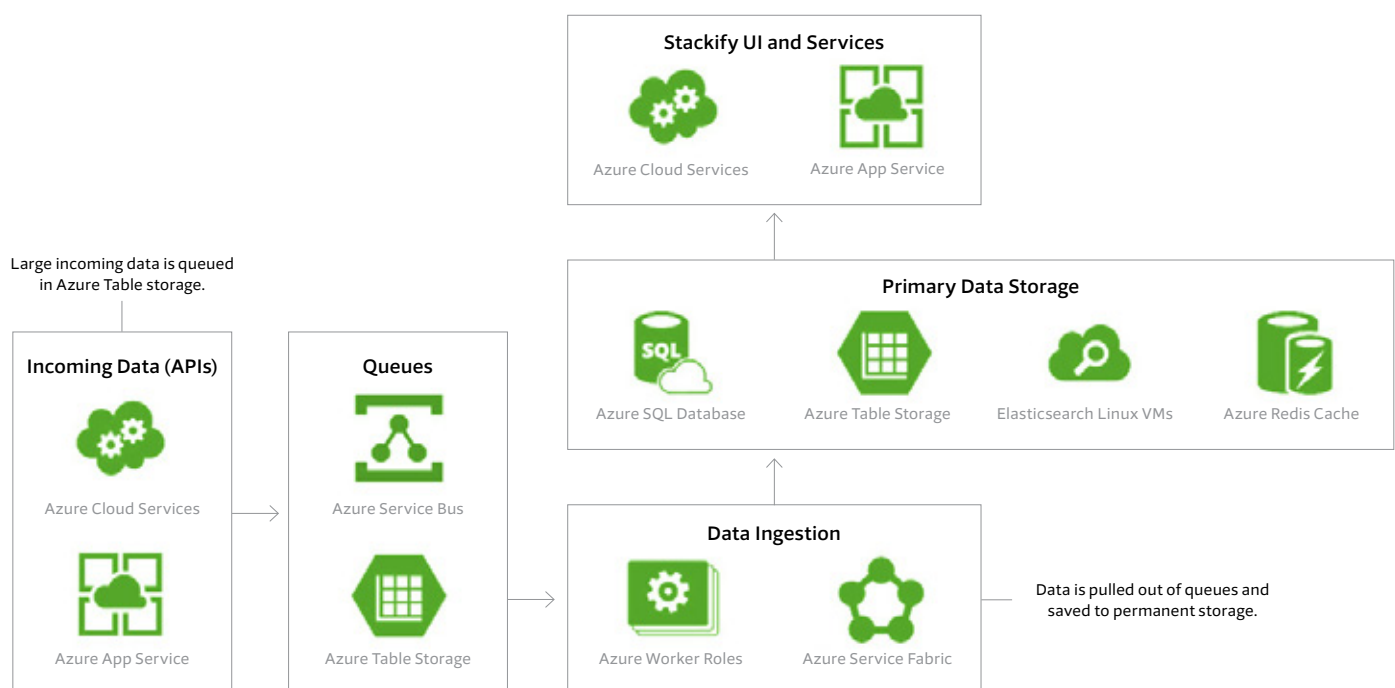
Defining Application Dependency Mapping and Performance

Modern applications tend to rely on many application dependencies. Most applications use some form of database and external HTTP based web services. If you are not intimately familiar with the code, it can be very difficult to know exactly what the code does and what dependencies that it has. It is also very important to understand the performance of those dependencies and how they could be impacting your application and your users.

What is Application Dependency Mapping?

Application dependency mapping is the process of identifying and documenting all the external dependencies of a software application. This can be done automatically by various tools or could be manually compiled by auditing the application code.

Developers commonly use visualization tools like Visio to draw out the architecture of their applications. This includes mapping out all the dependencies and how they talk to each other. Although, by doing it manually they are likely to miss some application dependencies.



Application Dependency Mapping Tools

Application dependency mapping is a common feature for most application performance management (APM) products. Most APM products use various types of code profiling and instrumentation strategies to track the overall performance of applications and their dependencies. As part of this, they can discover all the dependencies and provide a map or list of them.

It is also possible to do application dependency mapping by monitoring the network traffic on a server. Some network performance monitoring (NPM) solutions can deduce some details about what an application is connecting to and potential performance problems associated with it.

Application dependency mapping is a very useful feature of many APM & NPM tools. Retrace, our APM product, can help you identify all of your application dependencies and their performance.

Types of Application Dependencies



Azure Storage



Azure Service Bus



SQL Server



HTTP Calls



Redis



Elasticsearch



RavenDB



MongoDB



PostgreSQL



MySQL



Oracle

Common Application Dependencies

Applications use a wide variety of various application dependencies. Here are some of the common ones and how they are used. Application dependency mapping can help identify these types of common dependencies.

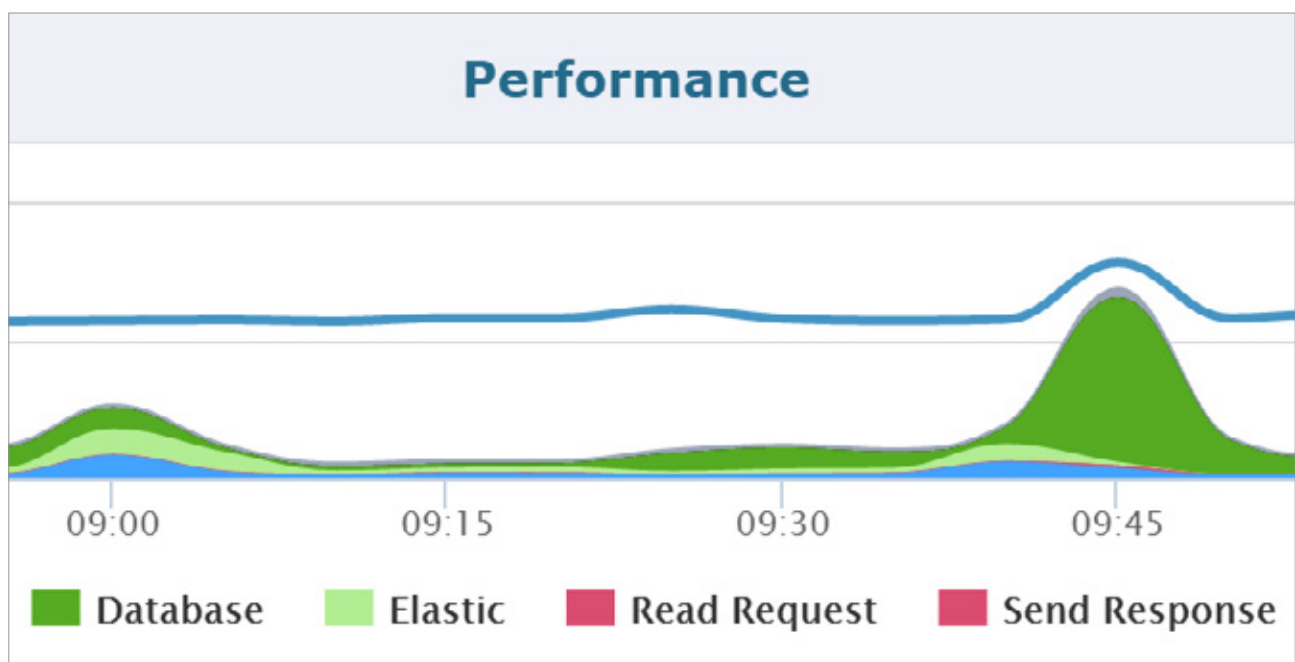
Databases (SQL, NoSQL, Documents): Most applications use some form of SQL database. Common SQL database providers are SQL Server, Oracle, PostgreSQL, MySQL, Sqlite and others. It is also increasingly common for applications to also use NoSQL or document databases like MongoDB, Cassandra and others.

Caching: Applications with a lot of traffic can greatly benefit from caching. A lot of data used by applications does not change very often and caching it can greatly reduce the load on databases or other resources. Popular caching solutions include Redis, Memcached, AppFabric, and others.

Message Queues: Queues can be used for a wide variety of purposes. They are typically used to disconnect one app from another and allow a durable messaging system to exist between them. This increases fault tolerance and prevents the receiver of the messages from getting overloaded if there is a big spike in traffic. Some common queuing solutions are MSMQ and RabbitMQ.

Web Services, HTTP APIs: Many web sites and vendors provide a wide array of APIs that can be accessed over HTTP based web services. At Stackify, we use APIs like Twilio for sending text messages and SendGrid for sending emails. Other uses could be communication between REST API based microservices or multiple applications within your own company.

How to Monitor Application Dependency Performance



Stackify Retrace

Application performance management systems can track all of your application dependencies to help understand the performance of them. If an application dependency becomes unavailable or slow, it can have a big impact on the performance of your applications.

“You don’t know what you don’t know” if you lack good application monitoring best practices. Here are some basic suggestions:

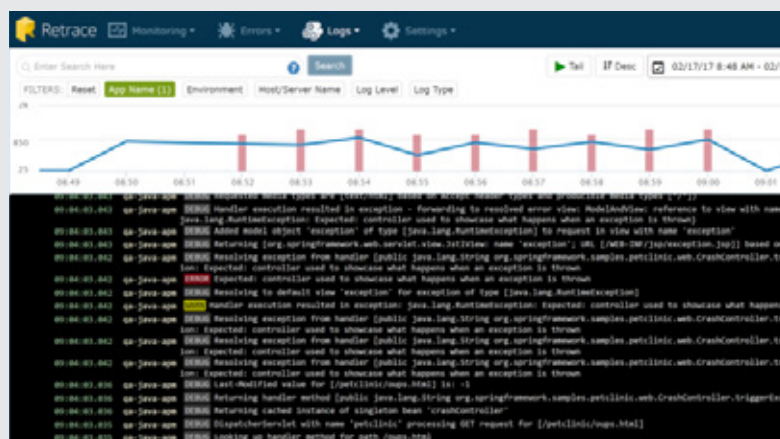
Exceptions: It is important to always collect and monitor exceptions with your application.

Exceptions can be your first line of defense for identifying connection problems or SQL query timeouts. Most APM products also collect exceptions and provide visibility around these types of problems.

SQL Queries: Being able to identify all the SQL queries within your application is very critical to identify potential performance problems. SQL queries that are slow or called too frequently are common performance problems. APM solutions can help you quickly identify which SQL query is the problem so you know how to improve the performance of your application.

Web Services: If your application utilizes a lot of web services, it is important to monitor if they are working properly. The last thing you want is for a third party service to be down and causing problems within your application that you aren’t aware of. Good exception handling and monitoring for exceptions are also useful for these types of problems.

We occasionally have issues with inbound data being processed slower than normal because some Azure queuing or storage service might be slow. It could be slow due to noisy neighbors, Azure updates, a fault within Azure’s infrastructure or a wide variety of other issues. These scenarios will always exist and monitoring them can help keep you from guessing.



“I am using Stackify to monitor a multi-tier application in the cloud, and was amazed at the easy set-up with no extra configuration.

Competitors seemed overly confusing, and costs would have exceeded our total hosting cost. Stackify was an easy choice. Honestly, we haven’t experienced a single drawback.”

*–Aaron Mehlmauer,
Management Science Associates*

Retrace

**APM + Errors + Logs + Monitoring
in one killer tool – QA & production!**

Monitor Your Azure Apps & Servers

Azure Virtual Machines

Traditional monitoring tools work fine with Azure virtual machines, including common monitoring tools like Nagios, SCOM, SolarWinds, Retrace and other common tools. Azure VMs give you complete administrative control of the server using these tools.

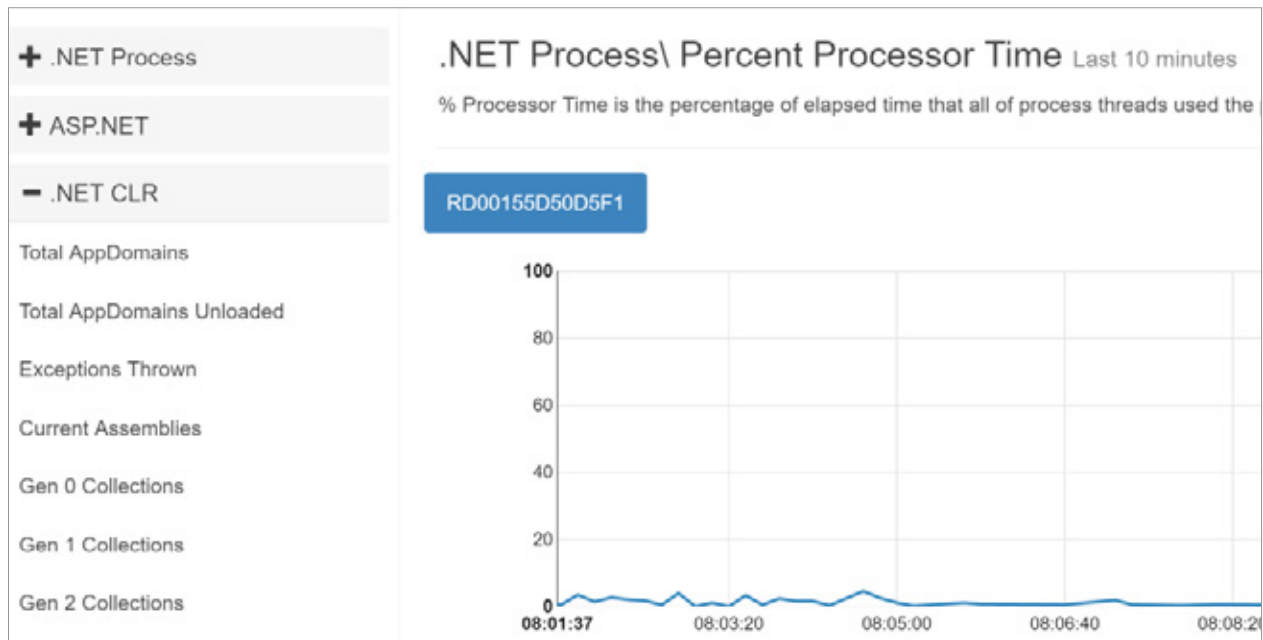
Azure App Services

App Services abstract away the underlying servers that your applications are running on. You do not have access to install traditional monitoring agents. Instead, there are some APM solutions that work with App Services, like Retrace. Azure Monitor can also provide some basic performance metrics.

Web Apps						
Name (Environment)	Instances	Requests/Min	Satisfaction	Errors/Min	Features	Alerts
AzureBakery (Azure)	1	 2	 100			  
HelloStackify.Web (Sandbox)	1	 15	 93	 3	  	  

Stackify Retrace View of Azure App Services

App Service limits the ability to monitor Windows Performance Counters. For example, this makes it impossible to monitor things like .NET Garbage Collection counters. ETW is also not fully supported. Within the Azure Portal, you can access "Metrics Per Instance" to see some Windows Performance Counters, but you are limited to these and they don't seem to show in Azure Monitor currently.



App Services Metrics Per Instance View

WebJobs can be used as a monitoring agent to perform some server and application monitoring functions. Retrace's monitoring agents are deployed this way.

Azure Cloud Services

Cloud Services do not have the same limitations as App Services. Cloud Services provide administrator level access to the servers they run on. You can configure startup tasks to install monitoring agents. You should be able to use most traditional application monitoring tools if you can script the install process. Azure Monitor does not provides near as many metrics as it does for App Services.

The screenshot shows a configuration interface for monitoring metrics. It features a list of metrics with checkboxes next to them. The 'CPU percentage' metric is selected, indicated by a checked checkbox. The other metrics, 'Disk read', 'Disk write', 'Network In', and 'Network Out', have unchecked checkboxes. The interface is dark-themed with a blue background.

Azure Diagnostics

Azure diagnostics can also be used in conjunction with your Azure virtual machines, Cloud Services, and App Services. Azure diagnostics gives you access to various types of log files, crash dumps, performance counters, and other data. This data is saved to Azure storage and can optionally be pushed into Application Insights.

How to Monitor SQL Azure

When SQL Azure first came, it had a subset of SQL features and some limited monitoring capabilities. SQL Azure is now a very robust sibling of SQL Server. SQL Azure has some unique monitoring features but also supports most standard SQL Server DMVs. Most all standard SQL monitoring tools like SQL Sentry, Ignite and others should work with SQL Azure.

Monitoring options:

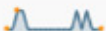



- DBA tools like SQL Sentry, Ignite
- Azure Monitor key metrics
- Standard SQL DMVs
- Retrace APM

As part of Stackify Retrace, the usage and performance of all of your SQL queries are automatically tracked. Retrace can be used to identify slow and overused SQL queries. Retrace can also track specific queries and alert you if they are slow.

SQL Query	% Time	Calls	Avg Time	Urls Using
select person	51.2%	4,858	0.153s	4
select categories	13.3%	1,266	0.152s	2
select businessentity	11.9%	1,138	0.152s	1
select person, businessentity	11.9%	1,138	0.151s	1
select productdocument	11.8%	1,138	0.151s	1

How to Monitor Azure Service Bus

If you are using Azure Service Bus in your application, it is important that you monitor your queues. If too many messages are in your queues then you know your applications are either not working or not reading the messages fast enough. Retrace can monitor individual queues so you can ensure messages are not stacking up.

Description	Last Update	Trend	Status
Active Queued Messages	a few secs ago		3 messages (Partitioned Queue)
Deadletter Messages	a few secs ago		0 messages (Partitioned Queue)
Queue Size in KB	a few secs ago		178 KB (Partitioned Queue)
Scheduled Messages	a few secs ago		594 messages (Partitioned Queue)

You can also utilize Retrace APM functionality to profile the code in your Azure Worker Roles. You can track how often you process queued messages, how long it takes, and exactly what your code is doing.

Azure Monitoring Tools

As noted above, not all monitoring tools work with Azure or all the different types of Azure resources. Microsoft provides some various tools that you can use and there are third party tools available as well. You may want to use a combination of Azure Monitor plus an APM solution like Retrace for code level performance.

Some recommended Azure monitoring tools:






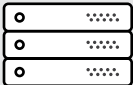
- Stackify Retrace (see below)
- Azure Monitor (via Azure Portal)
- Azure-Costs.com

Retrace is one tool every developer team needs.

(And any developer team can afford)

"I've spent most of my career as a consultant at major corporations. Some of my smaller clients have asked that I take on operational roles, but it felt like there was too much—until I found Stackify. Now I can see trouble spots and drill in. Instead of having to spend all my time finding the hotspot, I can now spend that time fixing the hotspot."

—Steven Roberts, Broad Reach Software

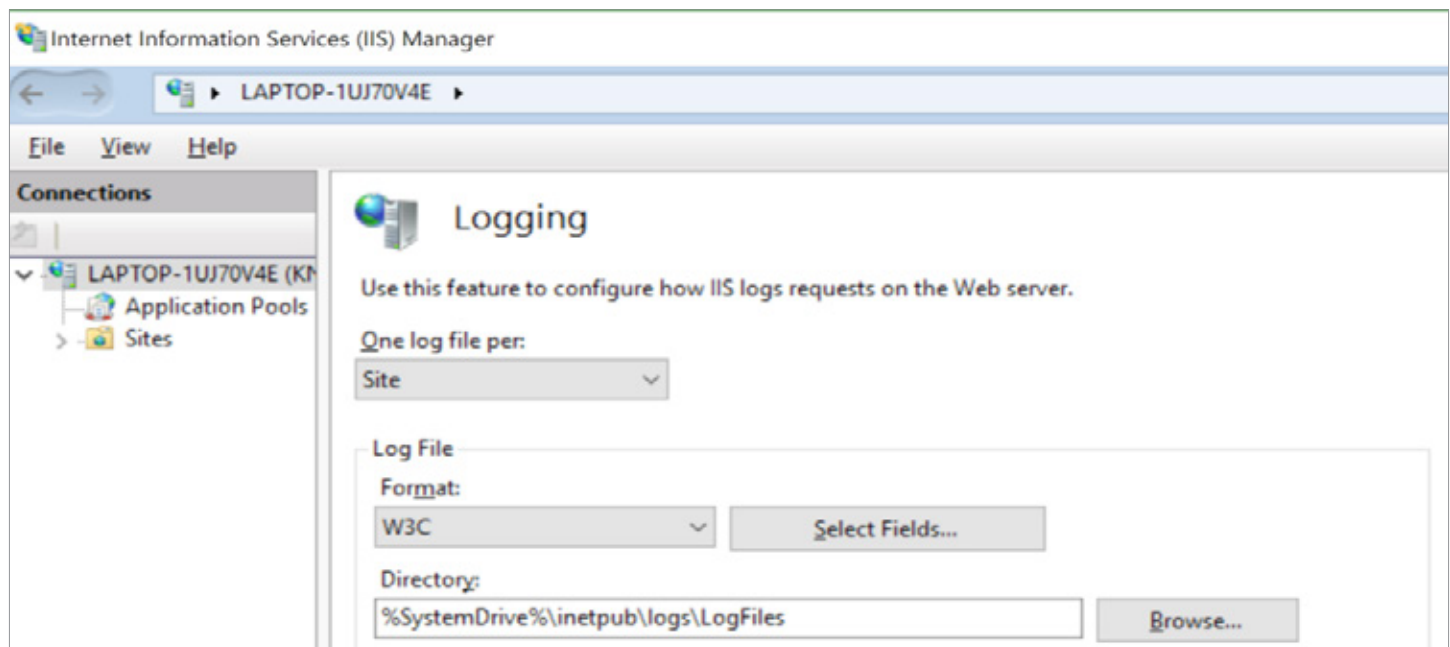
GET	WITH
 APM	 Retrace
 LOGS	
 ERRORS	
 MONITORING	
IN	STARTING AT ONLY
 QA/PRE-PROD	\$99/mo
 PRODUCTION	

Understanding the Four Different IIS Logs

You are probably familiar with normal IIS logs, but there are some other places to look if you want more detailed error messages or can't find anything in your IIS log file.

1. Standard IIS Logs

Standard IIS logs will include every single web request that flows through your IIS site. Via IIS Manager, you can verify that your IIS logs are enabled and where they are being written to.



You should find your logs in folders that are named by your W3SVC site ID numbers. Need help finding your logs? Check out: [Where are IIS Log Files Located?](#)

By default, each logged request in your IIS log will include several key fields including the URL, querystring, and error codes via the status, substatus and win32 status. These status codes can help identify the actual error in more detail.

```
#Fields: date time s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username  
c-ip cs(User-Agent) cs(Referer) sc-status sc-substatus sc-win32-status time-  
taken  
2016-09-13 21:45:10 ::1 GET /webapp2 - 80 - ::1 Mozilla/5.0 - 500 0 0 5502  
2016-09-13 21:45:10 ::1 GET /favicon.ico - 80 - ::1 Mozilla/5.0 http://  
localhost/webapp2 404 0 2 4
```

The “sc-status” and “sc-substatus” fields are the standard HTTP status code of 200 for OK, 404, 500 for errors, etc.

The “sc-win32-status” can provide more details that you won’t know unless you look up the code. They are basic [Win32 error codes](#).

By the way, if you are using Retrace, you can also use it to query across all of your IIS logs as part of its built in log management functionality.

2. Can’t Find Your Request in the IIS Log? HTTPERR is Your IIS Error Log.

Every single web request should show in your IIS log. If it doesn’t, it is possible that the request never made it to IIS or IIS wasn’t running. Also, be sure to enable your IIS logging.

Incoming requests to your server first route through HTTP.SYS before being handed to IIS. These types of errors get logged in HTTPERR. Common errors are 400 Bad Request, timeouts, 503 Service Unavailable and similar types of issues. The built-in error messages and error codes from HTTP.SYS are usually very detailed.

Where are the HTTPERR error logs?




C:\Windows\System32\LogFiles\HTTPERR

3. Look for ASP.NET Exceptions in Windows Event Viewer

By default, ASP.NET will log unhandled 500 level exceptions to the Windows Application EventLog. This is handled by the ASP.NET Health Monitoring feature. You can control settings for it via system.web/healthMonitoring in your web.config file.

Very few people realize that the number of errors written to the Application EventLog is rate limited. So you may not find your error! By default, it will only log the same type of error once a minute. You can also disable writing any errors to the Application EventLog.

Application Number of events: 27,655 (!) New events available

Level	Date and Time	Source
 Information	1/20/2017 7:25:11 AM	Security-SPP
 Warning	1/20/2017 7:25:11 AM	ASP.NET 4.0.30319.0
 Information	1/20/2017 7:25:08 AM	.NET Runtime

Event 1309, ASP.NET 4.0.30319.0

General Details

Event code: 3005
Event message: An unhandled exception has occurred.
Event time: 1/20/2017 7:25:11 AM
Event time (UTC): 1/20/2017 1:25:11 PM
Event ID: 74904640aea045a88bbd5a4b08add42d
Event sequence: 8
Event occurrence: 1
Event detail code: 0

Application information:
 Application domain: /LM/W3SVC/1/ROOT-1-131293923086948710
 Trust level: Full
 Application Virtual Path: /
 Application Path: C:\inetpub\wwwroot\
 Machine name: LAPTOP-1UJ70V4E

Process information:
 Process ID: 15912
 Process name: w3wp.exe
 Account name: IIS APPPOOL\DefaultAppPool

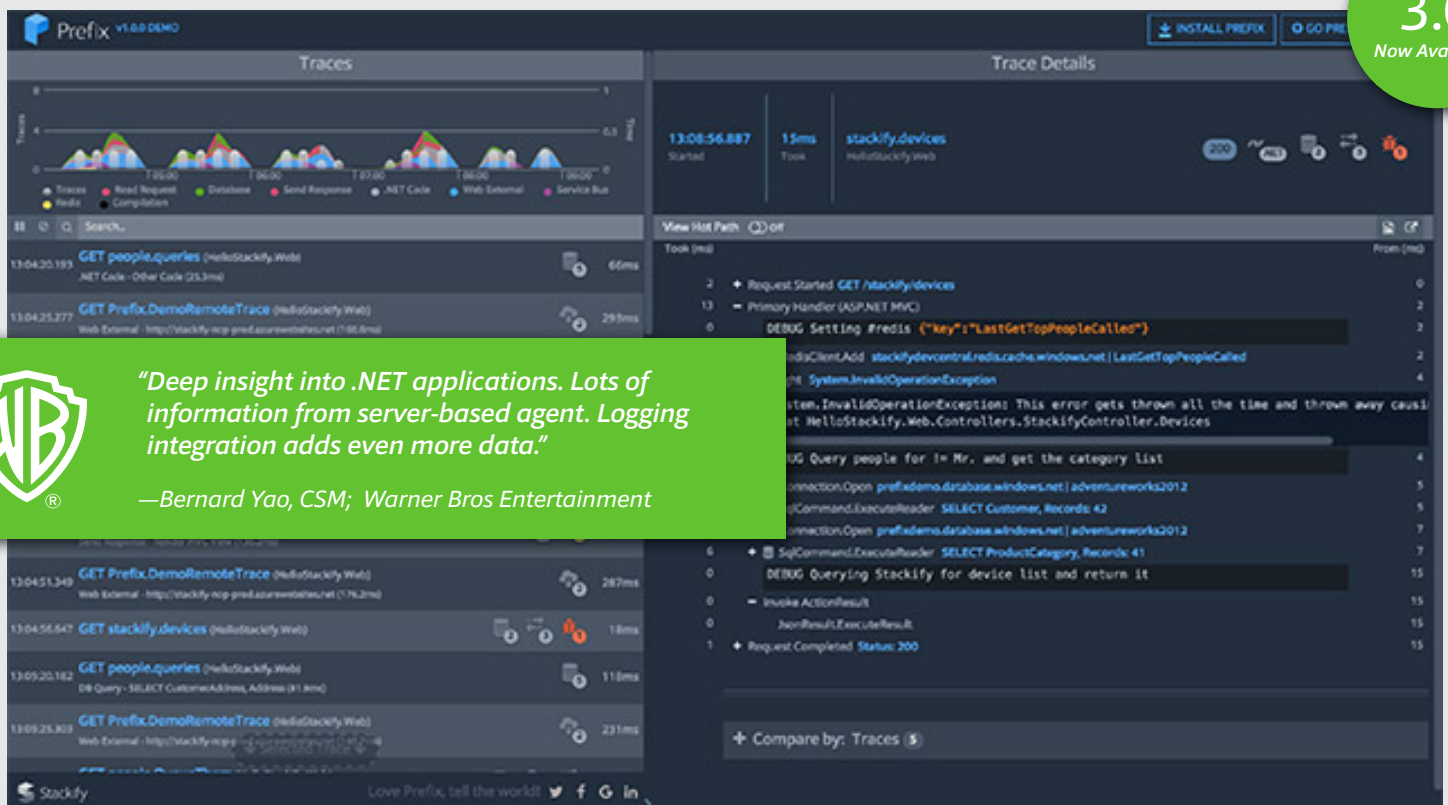
Can't find your exception?

Depending on if you are using WebForms, MVC, Web API, WCF or other frameworks, you may have issues with ASP.NET not writing any errors at all to ASP.NET due to compatibility issues with the health monitoring feature.

4. Failed Request Tracing for Advanced IIS Error Logs

Failed request tracing (FRT) is probably one of the least used features in IIS. It provides robust IIS logging and works as a great IIS error log. FRT is enabled in IIS Manager and can be configured via rules for all requests, slow requests, or just certain response status codes.

The only problem with FRT is it is insanely detailed. It tracks every detail and every step of the IIS pipeline. You can spend a lot of time trying to decipher a single request.



The screenshot displays the Prefix application interface. On the left, a 'Traces' panel shows a timeline of requests with a search bar and a list of recent requests. The 'Trace Details' panel on the right shows a specific request's timeline, including steps like 'Request Started', 'Primary Handler (ASP.NET MVC)', and 'DEBUG Setting #redis'. A green callout bubble in the top right corner says '3.0 Now Available!'. A green banner at the bottom left features the Warner Bros. Entertainment logo and a quote from Bernard Yao, CSM.

Prefix 3.0 Now Available!

"Deep insight into .NET applications. Lots of information from server-based agent. Logging integration adds even more data."

—Bernard Yao, CSM; Warner Bros Entertainment

Prefix

Prefix is a lightweight tool that shows .NET developers:

Logs + Errors + Queries + More, in real time—and it's free!

Methods to Find Slow SQL Queries

SQL performance tuning is a never-ending battle. We have team members that have worked with SQL Server databases with terrabytes of RAM all the way down to Stackify's massive fleet of little SQL Azure databases. We've seen a little bit of everything over the years. Here, we'll provide some tips for how developers can find slow SQL queries and do performance tuning in SQL Server.

1. Find Slow Queries With SQL DMVs

One of the great features of SQL Server is all of the dynamic management views (DMVs) that are built into it. There are dozens of them and they can provide a wealth of information about a wide range of topics.

There are several DMVs that provide data about query stats, execution plans, recent queries and much more. These can be used together to provide some amazing insights.

For example, this query below can be used to find the queries that use the most reads, writes, worker time (CPU), etc.

```
SELECT TOP 10 SUBSTRING(qt.TEXT, (qs.statement_start_offset/2)+1,
((CASE qs.statement_end_offset
WHEN -1 THEN DATALENGTH(qt.TEXT)
ELSE qs.statement_end_offset
END - qs.statement_start_offset)/2)+1),
qs.execution_count,
qs.total_logical_reads, qs.last_logical_reads,
qs.total_logical_writes, qs.last_logical_writes,
qs.total_worker_time,
qs.last_worker_time,
qs.total_elapsed_time/1000000 total_elapsed_time_in_S,
qs.last_elapsed_time/1000000 last_elapsed_time_in_S,
qs.last_execution_time,
qp.query_plan
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) qt
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) qp
ORDER BY qs.total_logical_reads DESC -- logical reads
-- ORDER BY qs.total_logical_writes DESC -- logical writes
-- ORDER BY qs.total_worker_time DESC -- CPU time
```

The result of the query will look something like this below. The image below is from a marketing app. You can see that one particular query (the top one) takes up all the resources.

By looking at this, we can copy that SQL query and see if there is some way to improve it, add an index, etc.

(No column name)	execution_count	total_logical_reads	total_logical_writes	total_worker_time	total_elapsed_time_in_S	last_execution_time
select Page, Query, Clicks, Impressions, clicks/convertfloa...	7	311978	2	33134409	179	2017-06-16 15:01:59.670
SELECT v.name AS [Name], SCHEMA_NAME(v.schema_j...	1	44390	2	171245	0	2017-06-16 17:53:39.593
select Timestamp, SUM(TotalPageViews) PageViews, Tag ...	3	33630	0	710290	0	2017-06-16 14:47:28.603
select Timestamp, SUM(TotalPageViews) PageViews, Tag ...	1	11210	0	232029	0	2017-06-16 17:22:31.287
select SUM(TotalPageViews) PageViews, Tag from view_...	1	11157	0	205006	0	2017-06-16 17:22:42.037
select Timestamp, SUM(TotalPageViews) PageViews, Tag ...	2	8329	5	411043	0	2017-06-16 15:57:55.270
select top 100 Page, Author, Published, Title, ImageUrl, Ta...	2	222	0	6227	0	2017-06-16 15:01:59.023
select is_federation_member from sys.databases where na...	1	16	0	487	0	2017-06-16 17:52:16.707
SELECT TOP 10 SUBSTRING(qt.TEXT, lqs.statement_sta...	1	0	0	43947	0	2017-06-16 17:52:19.603

Pros: Always available basic rollup statistics.

Cons: Doesn't tell you what is calling the queries. Can't visualize when the queries are being called over time.

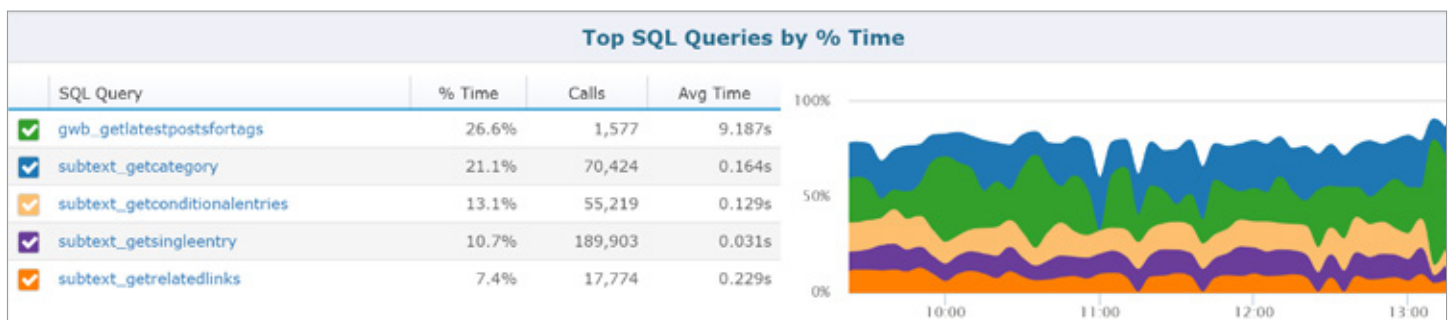
2. Query Reporting via APM Solutions

One of the great features of many application performance management (APM) tools is their ability to track SQL queries. For example, Retrace tracks SQL queries across multiple database providers, including SQL Server.

Retrace can tell you how many times a query has been executed, how long it takes on average, and what transactions are calling it. This is really valuable information for SQL performance tuning.

APM solutions collect this data by doing lightweight performance profiling against your application code at runtime.

Below is a screenshot from Retrace's application dashboard that shows for a particular application, which SQL queries take the longest.

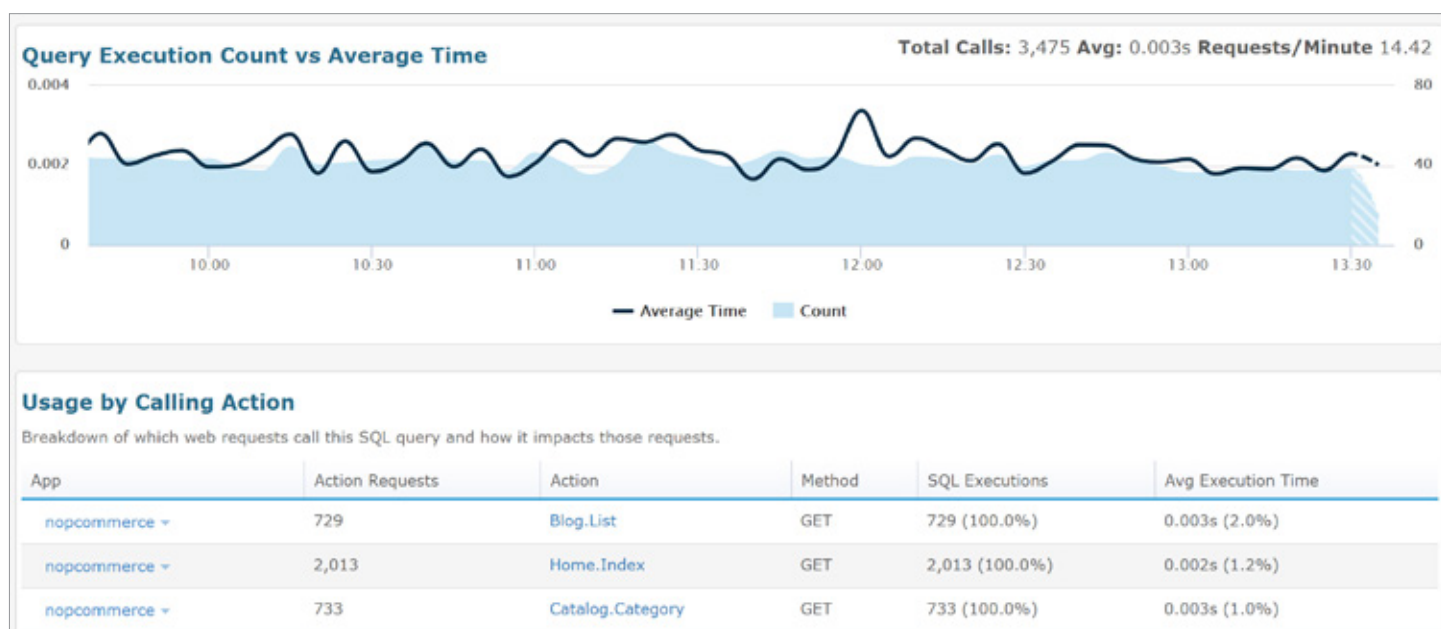


Retrace collects performance statistics about every single SQL query being executed. You can search for specific queries to hunt down potential problems.

Search

SQL Query	% Time	Calls	Avg Time	Urls Using
productloadallpaged	31.9%	732	0.061s	1
select product	7.2%	2,008	0.005s	1
select currency	6.4%	3,460	0.003s	3
select language	6.2%	3,460	0.003s	3
select discount	6.2%	2,736	0.003s	2
select store	5.8%	3,508	0.002s	4
insert customer, select customer	5.6%	3,460	0.002s	3
select genericattribute	5.5%	4,188	0.002s	3
select customerrole	5.4%	3,460	0.002s	3
update customer	5.3%	3,460	0.002s	3

By selecting an individual query, you can see how often that query is called over time and how long it takes. You can also see what web pages use the SQL query and how their performance is impacted by it.



Since Retrace is a lightweight code profiler and captures ASP.NET request traces, it can even show you the full context of what your code is doing.

Below is a captured trace that shows all of the SQL queries and other details about what the code was doing. Retrace can even show log messages within this same view. Also, notice that it shows the server address and database name the query is being executed on. You can also see how many records were returned.

Took (ms)		From (ms)
37	➤ Request Started GET /desktops	0
264	▼ Primary Handler (ASP.NET MVC)	37
0	SqlConnection.Open stackify-nop.database.windows.net nopCommerce-prod	49
7	SqlInternalConnection.BeginSqlTransaction	49
2	➤ SqlCommand.ExecuteNonQuery UPDATE Customer, Records: 1	57
9	SqlInternalTransaction.Commit	59
0	SqlConnection.Open stackify-nop.database.windows.net nopCommerce-prod	68
1	➤ SqlCommand.ExecuteReader SELECT Category, Records: 1	68
0	SqlConnection.Open stackify-nop.database.windows.net nopCommerce-prod	76
1	SqlInternalConnection.BeginSqlTransaction	76
1	▼ SqlCommand.ExecuteReader INSERT GenericAttribute, SELECT GenericAttribute, Records: 1	78
	INSERT INTO [dbo].[GenericAttribute] ([EntityId],[KeyGroup],[Key],[Value],[StoreId]) VALUES (@_p0,@_p1,@_p2,@_p3,@_p4)	
	SELECT [Id]	
	FROM [dbo].[GenericAttribute]	
	WHERE @@ROWCOUNT > @_p5 AND [Id] = scope_identity()	
8	SqlInternalTransaction.Commit	79
0	SqlConnection.Open stackify-nop.database.windows.net nopCommerce-prod	101
6	➤ SqlCommand.ExecuteReader SELECT Currency, Records: 2	101
0	SqlConnection.Open stackify-nop.database.windows.net nopCommerce-prod	107
3	➤ SqlCommand.ExecuteReader SELECT GenericAttribute, Records: 1	108
0	SqlConnection.Open stackify-nop.database.windows.net nopCommerce-prod	111
36	➤ SqlCommand.ExecuteReader ProductLoadAllPaged	117
0	SqlConnection.Open stackify-nop.database.windows.net nopCommerce-prod	156
1	➤ SqlCommand.ExecuteReader SELECT Discount, Records: 2	159
132	▼ ViewResultBase.ExecuteResult	168

As you can see, Retrace provides comprehensive SQL reporting capabilities as part of its APM capabilities. It also provides multiple monitoring and alerting features around SQL queries.

Pros: Detailed reporting across apps, per app, and per query can show transaction traces and details on how queries are used. Starts at just \$99 a month. Is always running once installed.

Cons: Does not provide the number of reads or writes per query.

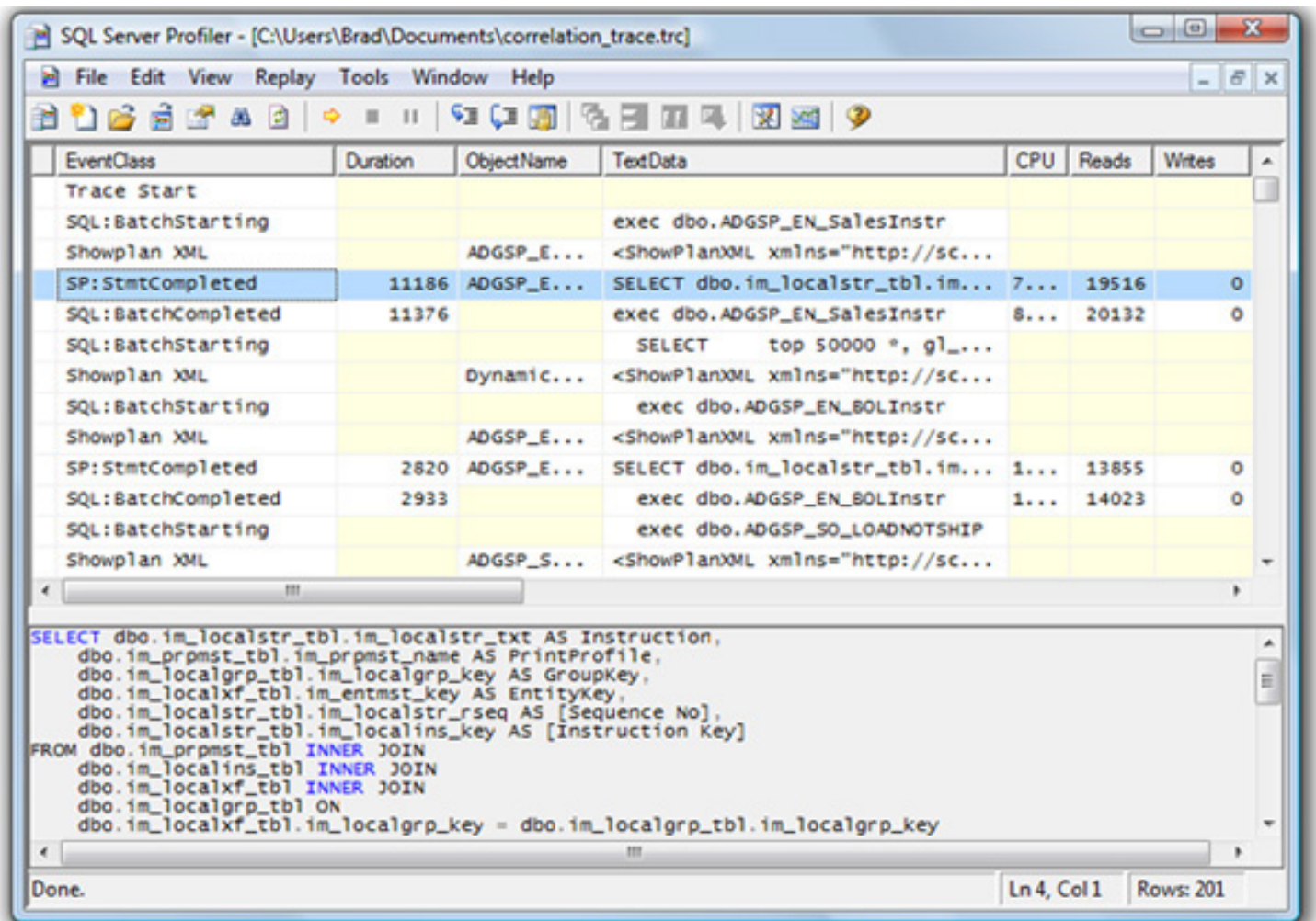
3. SQL Server Profiler (DEPRECATED!)

The SQL Server Profiler has been around for a very long time. It is very useful if you are trying to see in real time what SQL queries are being executed against your database.

NOTE: Microsoft has announced that SQL Server Profiler is being deprecated!

SQL Profiler captures very detailed events about your interaction with SQL Server.

- Login connections, disconnections, and failures
- SELECT, INSERT, UPDATE, and DELETE statements
- RPC batch status calls
- Start and end of stored procedures
- Start and end of statements within a stored procedure
- Start and end of a SQL batch
- Errors written to the SQL Server error log
- A lock acquired or released on a database object
- An opened cursor
- Security permission checks



EventClass	Duration	ObjectName	TextData	CPU	Reads	Writes
Trace Start						
SQL:BatchStarting			exec dbo.ADGSP_EN_SalesInstr			
Showplan XML		ADGSP_E...	<ShowPlanXML xmlns="http://sc...			
SP:StmtCompleted	11186	ADGSP_E...	SELECT dbo.im_localstr_tbl.im...	7...	19516	0
SQL:BatchCompleted	11376		exec dbo.ADGSP_EN_SalesInstr	8...	20132	0
SQL:BatchStarting			SELECT top 50000 *, gl_...			
Showplan XML		Dynamic...	<ShowPlanXML xmlns="http://sc...			
SQL:BatchStarting			exec dbo.ADGSP_EN_BOLInstr			
Showplan XML		ADGSP_E...	<ShowPlanXML xmlns="http://sc...			
SP:StmtCompleted	2820	ADGSP_E...	SELECT dbo.im_localstr_tbl.im...	1...	13855	0
SQL:BatchCompleted	2933		exec dbo.ADGSP_EN_BOLInstr	1...	14023	0
SQL:BatchStarting			exec dbo.ADGSP_SO_LOADNOTSHIP			
Showplan XML		ADGSP_S...	<ShowPlanXML xmlns="http://sc...			

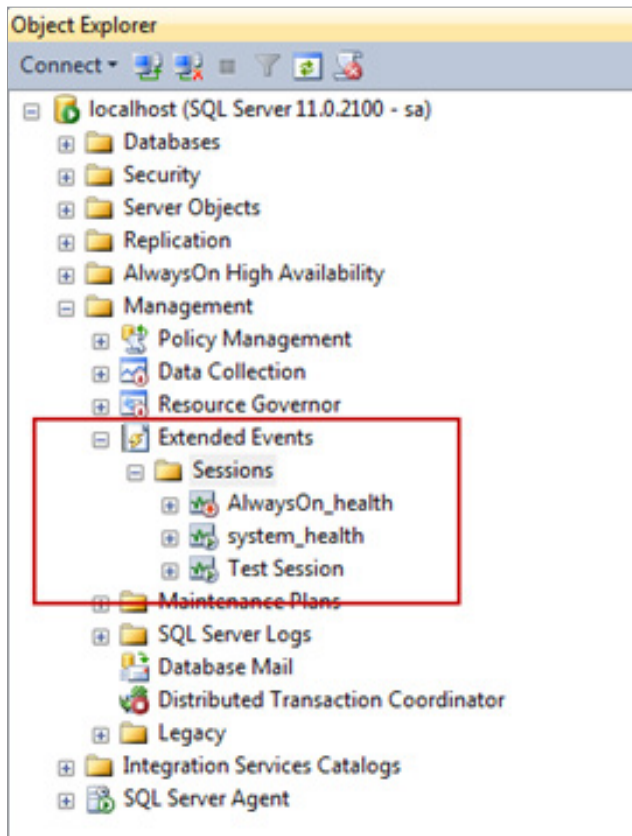

```
SELECT dbo.im_localstr_tbl.im_localstr_txt AS Instruction,
dbo.im_prmst_tbl.im_prmst_name AS PrintProfile,
dbo.im_localgrp_tbl.im_localgrp_key AS GroupKey,
dbo.im_localxf_tbl.im_entmst_key AS EntityKey,
dbo.im_localstr_tbl.im_localstr_rseq AS [Sequence No],
dbo.im_localstr_tbl.im_localins_key AS [Instruction Key]
FROM dbo.im_prmst_tbl INNER JOIN
dbo.im_localins_tbl INNER JOIN
dbo.im_localxf_tbl INNER JOIN
dbo.im_localgrp_tbl ON
dbo.im_localxf_tbl.im_localgrp_key = dbo.im_localgrp_tbl.im_localgrp_key
```

Pros: Very detailed data available.

Cons: You have to manually turn it on. This forces you to recreate a scenario you are trying to capture. It is eventually going away in favor of Extended Events.

4. SQL Server Extended Events

The SQL Profiler has been replaced by SQL Server Extended Events. This is sure to anger a lot of people but we can understand why Microsoft is doing it.



Extended Events works via Event Tracing (ETW). This has been the common way for all Microsoft related technologies to expose diagnostic data.

ETW provides much more flexibility. As a developer, you could easily tap into ETW events from SQL Server to collect data for custom uses. That is really cool and really powerful.

Pros: Easier to enable and leave running. Easier to develop custom solutions with.

Cons: Since it is fairly new, most people may not be aware of it.

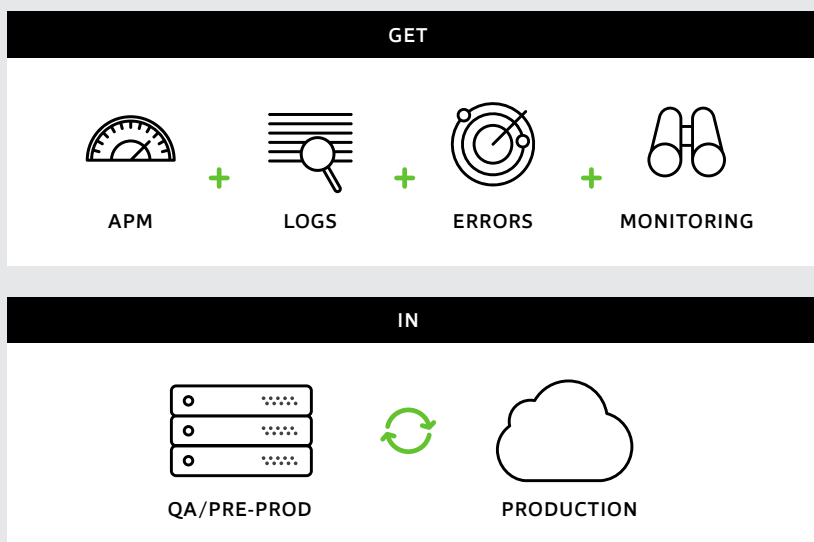


Is your dev team ready for better insights? **Retrace** is free for 14-days, then starts at only \$99/month.

Schedule a demo with our product experts now:

"We love the tool very much. It's saved us and our apps many times. It really helps get the error and logging information and helps make exception resolution turnaround quicker."

–Gajjar, Paycor

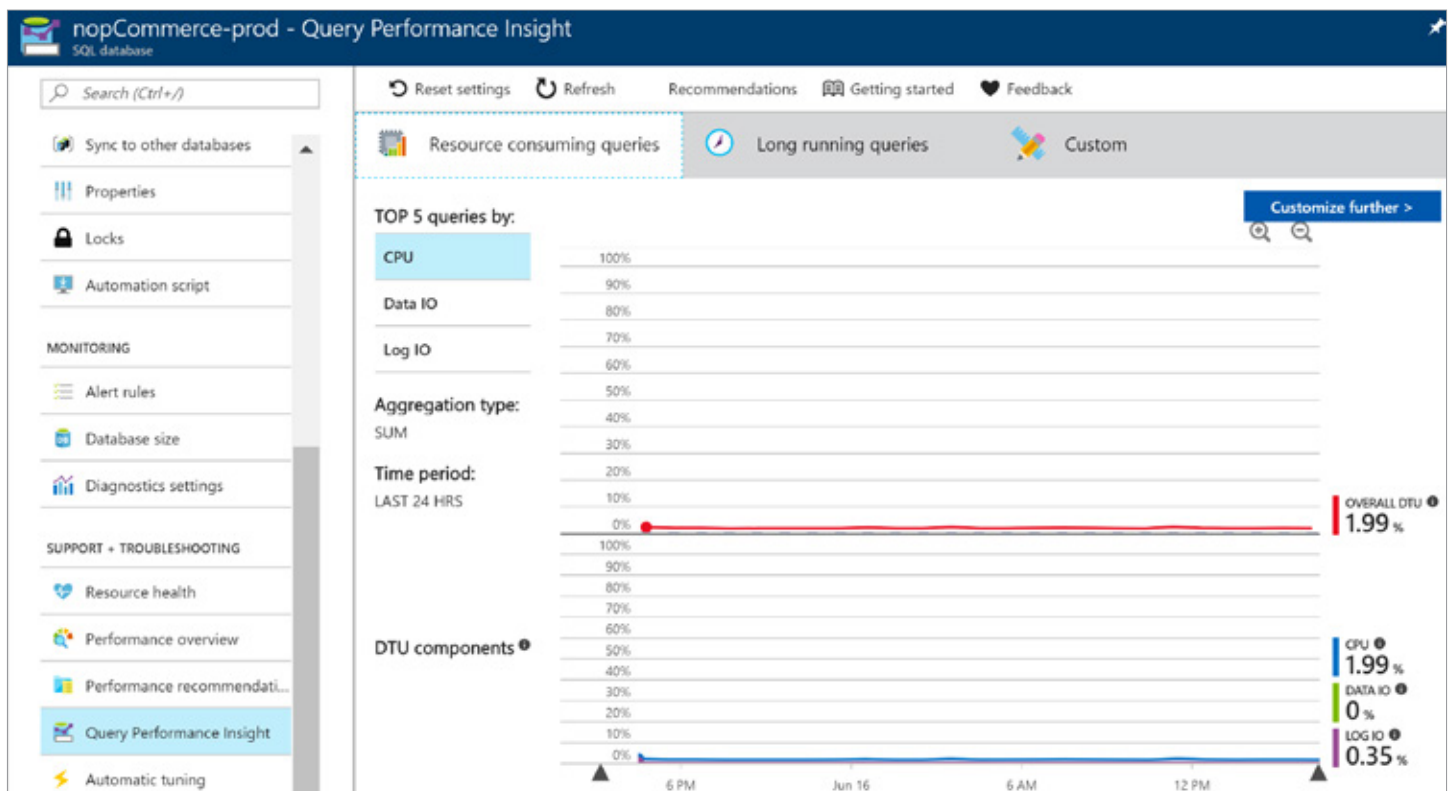


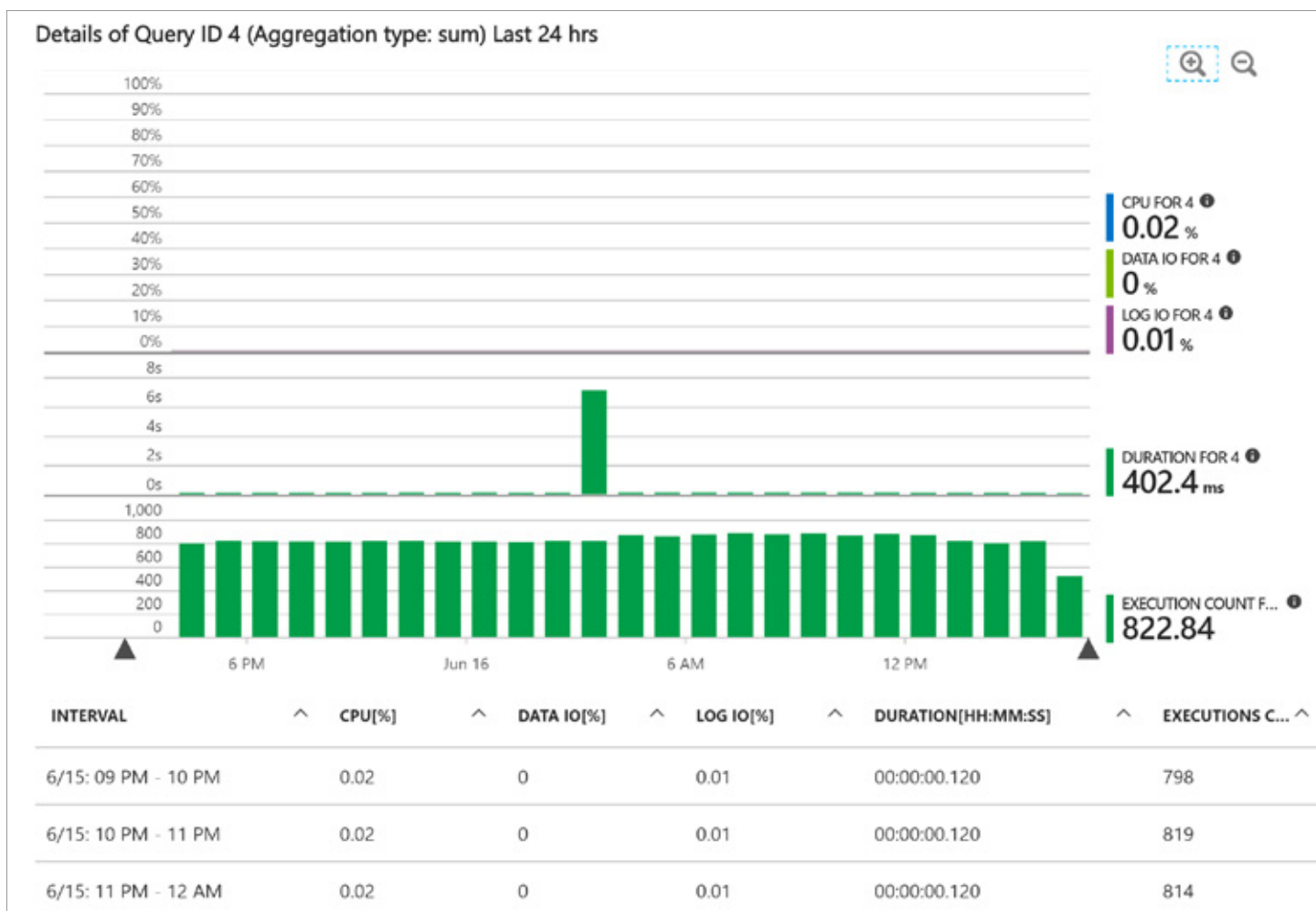
5. SQL Azure Query Performance Insights

Let's assume that SQL Azure's performance reporting is built on top of Extended Events. Within the Azure Portal you can get access to a wide array of performance reporting and optimization tips that are very helpful.

Note: These reporting capabilities are only available for databases hosted on SQL Azure.

In the screenshot below, you can see how SQL Azure makes it easy to use your queries that use the most CPU, Data IO, and Log IO. It has some great basic reporting built into it.





You can also select an individual query and get more details to help with SQL performance tuning.

Pros: Great basic reporting.

Cons: Only works on Azure. No reporting across multiple databases.

Summary

Next time you need to do some performance tuning with SQL Server, you will have a few options at your disposal to consider. Odds are, you will use more than one of these tools depending on what you are trying to accomplish.

Catching Exceptions in C# and Finding Application Errors

Exception handling is a critical component of every software application. The last thing you want is your users to see weird errors, or even worse, your application to keep crashing. In this article, we are going to discuss how to find and catch all exceptions in C# applications. .NET provides several different ways to catch exceptions and view unhandled exceptions.

Topics in this article:

- Catching “First Chance Exceptions”
- ASP.NET Exception Handling, including MVC, Web API, WCF, & ASP.NET Core
- .NET Framework Exception Events
- Find all exceptions with Retrace and no code changes
- Windows Event Viewer

Catching “First Chance Exceptions”

If you want to find and catch every single exception in your application, you need to be aware of “first chance exceptions.” The .NET Framework provides an easy mechanism to subscribe to every single exception that is ever thrown in your code. This includes exceptions that are caught deep inside your code that never gets surfaced anywhere. Although, you can’t technically “catch” them, you can subscribe to .NET Framework events so you can log them. Finding these exceptions is a great way to improve performance and eliminate weird application behaviors.

It is also important to note that first chance exceptions can include a lot of noise. Some exceptions happen because they are expected to, or may only happen when an application starts up as various warnings. Don’t be alarmed if you see some of this noise all of a sudden if you start logging these errors.

By subscribing to the event, you can potentially log every exception to help identify weird and hidden problems in your application. We don’t recommend doing this long term on a production application due to the sheer volume of noise and data it can cause. It is best used during development or to help find pesky production problems.

```
AppDomain.CurrentDomain.FirstChanceException += (sender, eventArgs) =>
{
    Debug.WriteLine(eventArgs.Exception.ToString());
};
```

ASP.NET Exception Handling

For ASP.NET web applications, you want to prevent users from receiving 500 level HTTP internal server error responses. How you do this varies depending on if you are using various ASP.NET frameworks or ASP.NET Core. For starters, be sure to enable custom errors within your web.config file so your users never see exception pages.

Subscribe to Global.asax Application_Error Event: If your application has a Global.asax, which is essentially an `HttpApplication`, you should set up events around unhandled exceptions. It is important to know that depending on if you are using MVC, Web API, Nancy, SerivceStack, WCF, etc, that not all exceptions may bubble up to your Global.asax error handler. Those specific web frameworks may also have their own error handling mechanisms. We will cover those below as well!

```
//Global.asax
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Error(object sender, EventArgs e)
    {
        Exception exception = Server.GetLastError();
        if (exception != null)
        {
            //log the error
        }
    }

    protected void Application_Start()
    {
        //may have some MVC registration stuff here or other code
    }
}
```

MVC Error Handling

With MVC, you have the ability to utilize the `HandleErrorAttribute` to do custom error handling per MVC controller action. It also has an `OnException` event within the Controller. In general, you should be safe with global error handling via your `Global.asax` file to catch all exceptions.

Web API Error Handling

Web API has more advanced exception handling capabilities that you need to be aware of.

- **Exception filter:** Ability to customize error handling for specific controllers and actions.
- **Exception logger:** Enables logging all unhandled exceptions.
- **Exception handler:** Global handler to customize the response back to the calling party of your API.

An example of using the unhandled exception logger:

```
public class UnhandledExceptionLogger : ExceptionLogger
{
    public override void Log(ExceptionLoggerContext context)
    {
        var log = context.Exception.ToString();
        //Write the exception to your logs
    }
}
```

You then need to register your exception logger as part of the startup configuration of Web API.

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        //Register it here
        config.Services.Replace(typeof(IExceptionLogger), new
        UnhandledExceptionLogger());
        // Web API routes
        config.MapHttpAttributeRoutes();
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id?}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

WCF Error Handling

Like Web API, WCF has a lot of customization options around exception handling. If you are using WCF, it is really critical that you set up an *IServiceBehavior* and *IErrorHandler* to properly catch all exceptions.

Check out this example on CodeProject for the proper way to do it: [WCF Global Exception Handling](#).

ASP.NET Core Error Handling

A lot has changed with ASP.NET Core. How you collect unhandled exceptions is done via an *ExceptionHandlerAttribute*. You can register it as a global filter and it will function as a global exception handler. Another option is to use a custom middleware designed to do nothing but catch unhandled exceptions.

```
public class ErrorHandlingFilter : ExceptionFilterAttribute
{
    public override void OnException(ExceptionContext context)
    {
        var exception = context.Exception;
        //log your exception here
        context.ExceptionHandled = true; //optional
    }
}
```

You must also register your filter as part of the Startup code.

```
//in Startup.cs
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc(options =>
    {
        options.Filters.Add(new ErrorHandlingFilter());
    });
}
```

.NET Framework Exception Events

The .NET Framework provides a couple events that can be used to catch unhandled exceptions. You only need to register for these events once in your code when your application starts up. For

ASP.NET, you would do this in the Startup class or Global.asax. For Windows applications, it could be the first couple lines of code in the Main() method.

```
static void Main(string[] args)
{
    Application.ThreadException += new ThreadExceptionHandler(Application_
ThreadException);
    AppDomain.CurrentDomain.UnhandledException += new UnhandledExceptionHandler
(CurrentDomain_UnhandledException);
}
static void Application_ThreadException(object sender, ThreadExceptionHandlerEventArgs e)
{
    // Log the exception, display it, etc
    Debug.WriteLine(e.Exception.Message);
}
static void CurrentDomain_UnhandledException(object sender,
UnhandledExceptionHandlerEventArgs e)
{
    // Log the exception, display it, etc
    Debug.WriteLine((e.ExceptionObject as Exception).Message);
}
```

Find All C# Exceptions With Retrace

One of the great features of Retrace is its error monitoring capabilities. Retrace can automatically collect all .NET exceptions that are occurring within your application with no code changes. This includes unhandled exceptions but can also include all thrown exceptions or first chance exceptions.

The best thing about this is it works with all types of ASP.NET applications. It works perfectly with MVC, WCF, Web API, .NET Core, etc.

Retrace provides excellent reporting around all of your exceptions. You can even set up alerts for high application exception rates or when a new exception is found.

Retrace offers three modes:

1. No exceptions
2. Unhandled exceptions only
3. All exceptions thrown – Use this to catch all exceptions

View Exceptions in Windows Event Viewer

If your application has unhandled exceptions, they may be logged in the Windows Event Viewer under the category of "Application." This can be helpful if you can't figure out why your application suddenly crashes.

Application Number of events: 23,619 (!) New events available		
Level	Date and Time	Source
Information	4/14/2017 10:28:05 AM	Windows Error Reporting
Error	4/14/2017 10:28:01 AM	Application Error
Error	4/14/2017 10:28:01 AM	.NET Runtime

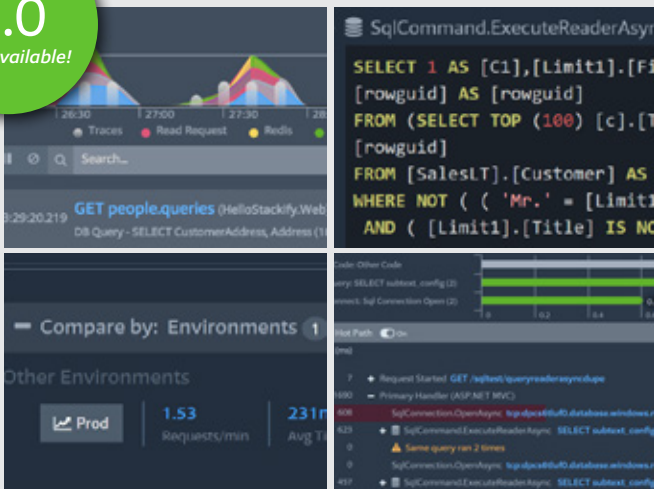
Windows Event Viewer may log 2 different entries for the same exception. One with a .NET Runtime error and another more generic Windows Application Error.

From the .NET Runtime:

```
Application: Log4netTutorial.exe
Framework Version: v4.0.30319
Description: The process was terminated due to an unhandled exception.
Exception Info: System.IndexOutOfRangeException
               at Log4netTutorial.Program.Main(System.String[])
```

3.0

Now Available!



"Prefix has helped me so much in writing better code. It's helped reduce calls to our database and to identify exactly why a few pages were taking so long to load. It is a great product and I am looking forward to seeing what Stackify comes up with next."

—Patrick McCarthy, Andrews McMeel Universal

Prefix Logs + Errors + Queries + More

Logged under Application Error:

```
Faulting application name: Log4netTutorial.exe, version: 1.0.0.0, time stamp:
0x58f0ea6b
Faulting module name: KERNELBASE.dll, version: 10.0.14393.953, time stamp:
0x58ba586d
Exception code: 0xe0434352
Fault offset: 0x000da882
Faulting process id: 0x4c94
Faulting application start time: 0x01d2b533b3d60c50
Faulting application path: C:\Users\matt\Documents\Visual Studio
2015\Projects\Log4netTutorial\bin\Debug\Log4netTutorial.exe
Faulting module path: C:\WINDOWS\System32\KERNELBASE.dll
Report Id: 86c5f5b9-9d0f-4fc4-a860-9457b90c2068
Faulting package full name:
Faulting package-relative application ID:
```

Summary

Having good best practices around logging and exceptions is critical for every software application. Logging is typically the eyes and the ears of the developer.

Retrace gives developers unparalleled visibility into the performance of their applications. Find all of your application exceptions and even get complete transaction tracing of what was happening when the exception was thrown.

A Retrospective on Structured Logging

Log files are one of the most valuable assets that a developer has. Usually when something goes wrong in production, the first thing you hear is “send me the logs.” The goal of structured logging is to bring a more defined format and details to your logging. We have been practicing structured logging at Stackify for quite a while and want to share some of our thoughts and best practices.

The problem with log files is they are unstructured text data. This makes it hard to query them for any sort of useful information. As a developer, it would be nice to be able to filter all logs by a certain customer # or transaction #. The goal of structured logging is to solve these sorts of problems and allow additional analytics.

For log files to be machine readable for more advanced functionality, they need to be written in a structured format that can be easily parsed. This could be XML, JSON, or other formats. But since virtually everything these days is JSON, you are most likely to see JSON as the standard format for structured logging.

Structured logging can be used for a couple different use cases:

1. **Process log files for analytics or business intelligence:** A good example of this would be processing web server access logs and doing some basic summarization and aggregates across the data.
2. **Searching log files:** Being able to search and correlate log messages is very valuable to development teams during the development process and for troubleshooting production problems.

Structured Logging Example

A simple example will probably help to make it clear as to what structured logging really is.

Normally you might write to a log file like this:

```
log.Debug("Incoming metrics data");
```

This would produce a line like this in your log:

```
DEBUG 2017-01-27 16:17:58 - Incoming metrics data
```

Depending on your logging framework, logging some additional fields would be done like this. This gives you the ability to potentially easily search on these custom fields.

```
log.Debug("Incoming metrics data", new {clientId=54732});
```

This would produce a line like this in your log, now including the extra field:

```
DEBUG 2017-01-27 16:17:58 - Incoming metrics data {"clientId":54732}
```

As a separate example with the logging library NLog for .NET, you can use it and specify a JsonLayout along with what fields you want to log and it will very easily produce an all JSON log file with each line looking something like this:

```
{ "time": "2010-01-01 12:34:56.0000", "level": "ERROR", "message": "hello,  
world" }
```

If you were using structured logging and sending it to a log management system, it would serialize the entire message and additional metadata as JSON. This is part of the power of using structured logs and a log management system that supports them. Here is an example of what something like Stackify's logging libraries upload to Retrace to our log management system.

```
[{
  "Env" : "Dev",
  "ServerName" : "LAPTOP1",
  "AppName" : "ConsoleApplication1.vshost.exe",
  "AppLoc" : "C:\\BitBucket\\stackify-api-dotnet\\Src\\
ConsoleApplication1\\bin\\Debug\\ConsoleApplication1.exe",
  "Logger" : "StackifyLib.net",
  "Platform" : ".net",
  "Msgs" : [{
    "Msg" : "Incoming metrics data",
    "data" : "{\\\"clientid\\\":54732}",
    "Thread" : "10",
    "EpochMs" : 1485555302470,
    "Level" : "DEBUG",
    "id" : "0c28701b-e4de-11e6-8936-8975598968a4"
  ]
}]
}
```

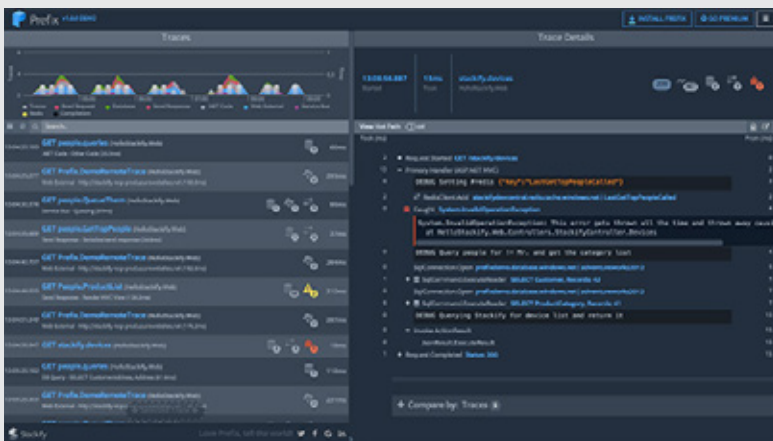
It is important to know that there is no real standard to structured logging and it can be done a lot of different ways. To get the most value out of it, you need to be using a logging framework (like log4net, log4j, etc.) that supports logging additional properties and then send that data to a log management system that can accept your custom fields and index them.

How to View Structured Logs

If you are programming with .NET or Java, you can use Prefix to view what your code is doing via transaction tracing along with your logging. Prefix can even show you any custom properties that are being logged as JSON. [Prefix is free and is the best log viewer developers can get.](#)

```

3  + Request Started GET /stackify/devices 0
17 - Primary Handler (ASP.NET MVC) 3
0  DEBUG Setting #redis {"key":"LastGetTopPeopleCalled"} 3
1  RedisClient.Add stackifydevcentral.redis.cache.windows.net | LastGetTopPeopleCalled 3
0  Caught System.InvalidOperationException 5
   System.InvalidOperationException: This error gets thrown all the time and thrown away causing CPU problems
   at HelloStackify.Web.Controllers.StackifyController.Devices
0  DEBUG Query people for != Mr. and get the category list 5
0  SqlConnection.Open prefixdemo.database.windows.net | adventureworks2012 5
2  + SqlCommand.ExecuteReader SELECT Customer, Records: 42 5
0  SqlConnection.Open prefixdemo.database.windows.net | adventureworks2012 8
10 - SqlCommand.ExecuteReader SELECT ProductCategory, Records: 41 8
    SELECT TOP (100) [c].[ProductCategoryID] AS [ProductCategoryID],[c].[ParentProductCategoryID] AS
    [ParentProductCategoryID],[c].[Name] AS [Name],[c].[rowguid] AS [rowguid],[c].[ModifiedDate] AS
    [ModifiedDate]
    FROM [SalesLT].[ProductCategory] AS [c]
  
```

Prefix

Prefix is a lightweight tool that shows Java developers:

Logs + Errors + Queries + More in real time—and it's free!

How We Use Structured Logging at Stackify

At Stackify, we use structured logging primarily to make it easier to search our logs. We care more about the benefits it provides for our developers from our logging system.

When we look at our logs, they look like this below. You can see all the custom fields we log because they show up as JSON.

```
DEBUG HandleBatch success #applog {"partition":"JAN271700p1","clientIdNumber":54732.0,"batchCountNumber":9.0,"tookMsNumber":125.0}
DEBUG ReadFromTable completed #applog {"partition":"JAN271700p1","clientIdNumber":54732.0,"avgLatencySecsNumber":3.05,"totalReadTimeNumber...
DEBUG #messagepump #async finished #queue message #logpump {"queueName":"LogProcessor","messageId":"81f18401-e4b5-11e6-80c1-00155da93582","messageObject":{
  "com...
DEBUG Removing items from the queue {"tableName":"AppLog53502","partition":"JAN271700p2","clientIdNumber":53502.0,"countNumber":3.0}
Worker_IN_9 DEBUG #monitorresults Calling CalculateSparklineFromDB {"clientIdNumber":56600.0,"monitoridNumber":1858.0}
DEBUG Appending elasticsearch with error details for client 56313
DEBUG updating logs to index complete took 0.000 secs
DEBUG Device is updated #performancemonitor {"clientIdNumber":56005.0,"appEnvId":"96e753f9-f5a5-e611-80c3-000d3a3289a8","testEpochNumber":247589...
DEBUG Device is updated #performancemonitor {"clientIdNumber":56005.0,"appEnvId":"96e753f9-f5a5-e611-80c3-000d3a3289a8","testEpochNumber":247589...
DEBUG Testing if all devices are reporting for an app complete #performancemonitor {"clientIdNumber":56005.0,"appEnvId":"96e753f9-f5a5-e611-80c3-000d3a3289a8","epochNumber":24758964...
DEBUG Signal to #performancemonitor #queue to import {"commandName":"AppEnvComplete","clientIdNumber":56005.0,"appEnvId":"96e753f9-f5a5-e611-80c3-000d3a3289a8"}
```

This enables us to very easily search by any of those fields via our log management system.

A simple search like this: "clientIdNumber:54732" shows us only those logs, helping us quickly narrow down problems for a specific client. You can search across every app and server we have from one place.

FILTERS:

```
12:31:00.710 stackifyProdAPIWest StackifyAPI_IN_2 DEBUG Signal to LogProcessor to import #applog {"partitionKey":"JAN271800p2","clientIdNumber":54732.0}
12:31:00.864 stackifyProdServicesWest ServiceHost_IN_1 DEBUG Incoming profiler detail traces {"countNumber":1.0,"clientIdNumber":54732.0}
12:31:00.989 stackifyProdServicesWest ServiceHost_IN_1 DEBUG Signal to ProfileTraces to import #profiletraces {"partitionKey":"JAN271800p2","clientIdNumber":54732.0}
12:31:01.027 stackifyProdServicesWest ServiceHost_IN_5 DEBUG Signal to @_monitormetric to import #monitormetric {"partitionKey":"JAN271800p3","clientIdNumber":54732.0}
12:31:01.608 stackifyProdServicesWest ServiceHost_IN_4 DEBUG Incoming profiler summary records {"minutesNumber":9.0,"clientIdNumber":54732.0,"clientDeviceidNumber":6.0,"maxEpochNumber":24758964}
12:31:01.623 stackifyProdServicesWest ServiceHost_IN_4 DEBUG Saving profiler summary records to SQL staging tables {"countNumber":9.0,"clientIdNumber":54732.0,"clientDeviceidNumber":6.0,"maxEpochNumber":24758964}
12:31:01.675 StackifyProdLogWorkerWestUS LogWorkers_IN_1 DEBUG ReadFromTable queue #applog {"partition":"JAN271800p3","tableName":"AppLog54732","clientIdNumber":54732.0}
12:31:01.691 StackifyProdLogWorkerWestUS LogWorkers_IN_1 DEBUG HandleBatch started #applog {"partition":"JAN271800p3","clientIdNumber":54732.0,"batchCountNumber":2.0}
12:31:01.722 StackifyProdLogWorkerWestUS LogWorkers_IN_1 DEBUG Index #logs response from ES #logsimport {"index":"c54732-17.1.27w","clientIdNumber":54732.0,"successBool":true}
12:31:01.722 StackifyProdLogWorkerWestUS LogWorkers_IN_1 WARN No log apps to update UpdateLastReceived {"clientIdNumber":54732.0}
12:31:01.722 StackifyProdLogWorkerWestUS LogWorkers_IN_1 DEBUG Removing items from the queue {"tableName":"AppLog54732","partition":"JAN271800p3","clientIdNumber":54732.0,"countNumber":2.0}
```

TIP: Log Extra Fields on Exceptions!

One of the best uses of structured logging is on exceptions. Trying to figure out why an exception happened is infinitely easier if you know more details about who the user was, input parameters, etc.

```
try
{
    //do something
}
catch (Exception ex)
{
    log.Error("Error trying to do something", new { clientid = 54732, user =
    "matt" }, ex);
}
```

Summary

It doesn't really take any longer to log custom properties as you write your logging. These extra properties can provide more details that make it easier to troubleshoot application problems. If you are using a log management system that supports searching by these custom fields, then you can also search your logs by these new properties.

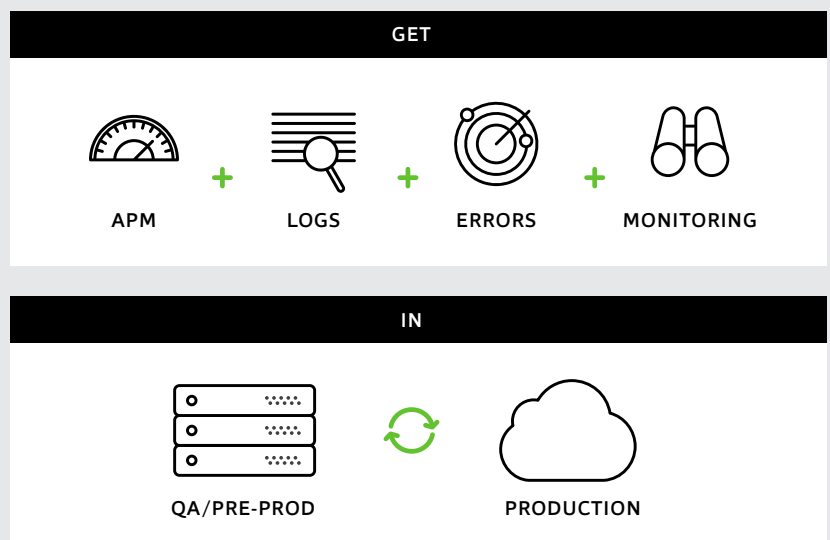
If you need help with structured logging, be sure to try out [Retrace](#), which includes logging as a standard feature along with APM, errors, monitoring, and metrics.



*Do you see what
your code is
doing and why?*

*Is your dev team ready for better insights?
Retrace is free for 14-days, then starts
at only \$99/month.*

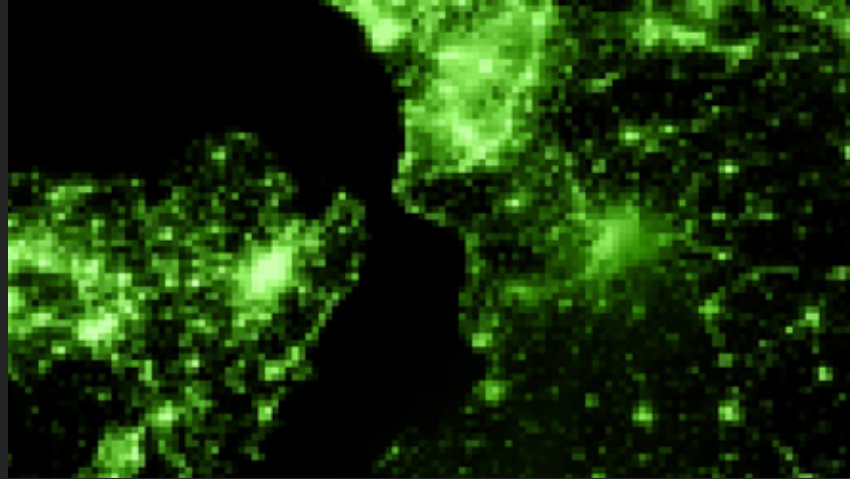
*Schedule a demo with our
product experts now:*



Build/better

©2017 Stackify

8900 State Line Rd. STE 100
Leawood, KS 66206



Stackify exists to increase developers' ability to create amazing applications as quickly as possible. We believe that the ideal development team, today and in the future, is consistently optimizing its output across the entire lifecycle of an application; from development to testing to production. Stackify's mission is to give developer teams easy access to powerful tools, which enable them to take the lead in delivering the best applications as quickly as possible.

If you're a developer, team lead, or architect, Stackify's tools were built for you. In fact, we have two game-changing, code performance products no developer or dev team should ever be without.



Prefix is a popular developer tool for finding and fixing bugs while you write your code. You have profilers and debuggers, but nothing is like Prefix, and it's free!



Retrace is an APM tool built specifically for developers and dev teams. Our agent can install on pre-prod or production servers to make black boxes transparent. Try Retrace free for 14-days.

"We're using Stackify to find potential misbehaviour in our software and to improve the performance of our support team. Most of our incoming tickets can be solved or least linked to our trace/error logs in Retrace.

As Stackify supports diagnostic contexts, it is easy for us to filter the logs by LEVEL or customer and to retrieve valuable metadata about the log context.

– Julian Neuhaus

There is good documentation how to integrate Stackify in your business applications. The 'Error View' is one of our most used features and helps a lot to find, analyze and keep track of bugs.

If you're using services/tools like Docker, AWS, Azure or Bluemix, it is a must have."