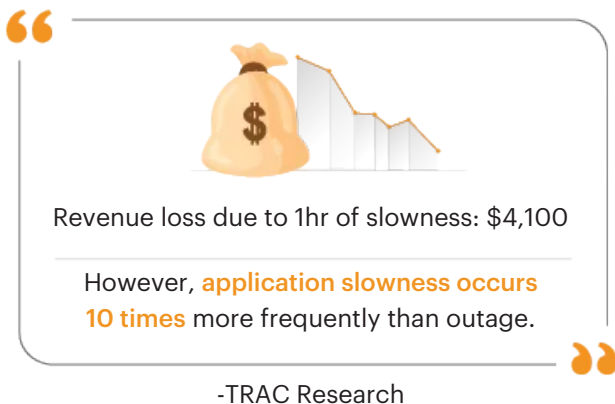




How to Get Full-Stack Visibility for Your Java Applications

Introduction

Java is one of the most popular technologies for application development. In a recent [eG Innovations and DevOps Institute survey of Application Performance Management in the new normal](#), almost 60% of respondents reported that they were using Java technologies for their key business applications. Java technology powers key business applications in various domains like finance, healthcare, insurance and education, and many of these applications process tens of millions of requests per day. When these applications slow down, they affect the user experience and in turn, impact business revenue. This is one of the reasons why performance monitoring for Java applications is garnering attention.



“

Revenue loss due to 1hr of slowness: \$4,100

However, **application slowness occurs 10 times** more frequently than outage.

”

-TRAC Research

What is Java APM?

Java APM stands for Java Application Performance Monitoring. Gartner defines application performance monitoring (APM) as software that enables the observation and analysis of application health, performance and user experience. Typical users of such solutions in an organization are IT operations teams, Site Reliability Engineers (SREs), reliability engineers, cloud and platform ops, application developers, and product owners.

There are two key capabilities of an APM solution:

#1 Measure user experience when accessing the application: User experience is important because it is a key measure of the success of any IT application or initiative. After all, if users are complaining of slowness, it doesn't matter what the CPU usage or memory utilization levels are.

Two common ways of measuring user experience are:

- ◆ **Synthetic Monitoring:** Tools in this category enable the creation of software robots to check different workflows that users follow when accessing an application. By repeatedly following these workflows from one or multiple locations, periodically, synthetic monitoring provides an indication of the experience users are likely to see.
- ◆ **Real user monitoring (RUM):** This refers to the ability to passively observe user activities and compute the response time that users see. Using JavaScript injection, RUM collects data directly from users' browsers or devices, tracking metrics such as page load times, network latency, JavaScript errors, and user interactions.

Learn more about: [Synthetic vs. Real User Monitoring \(RUM\)](#)

Both approaches to user experience monitoring have their pros and cons, and ideally, one should implement both approaches as the two approaches complement each other.

#2 Deep-dive/troubleshoot application performance: While knowing when there is a user experience problem is important, being able to determine the reasons for user experience issues and to resolve them is even more important.

Java APM tools instrument the Java application code, collect metrics during runtime, and report them to a centralized monitoring system. These metrics are used to analyze application performance, detect bottlenecks, and identify areas for optimization.

This whitepaper focuses on the second Java APM capability, i.e., deep-dive/troubleshooting application performance and how it is implemented in the eG Enterprise solution from eG Innovations.

Why is Java APM Challenging?

Detecting if an application is slow is the easy part. Even if you don't have proactive means of detecting a problem, it is likely that many users will call the helpdesk when a problem occurs. To determine why an application is slow can be very difficult for several reasons:

- » Modern Java applications are multi-threaded. Threads may be short lived or long running and there can be complex interactions and synchronizations between threads.
- » At the same time, Java applications are multi-tiered and different application components can execute on different systems, or even in different networks. There can be many different modes of communication between the tiers – e.g., message queues, API gateways, sockets, web service calls, etc.
- » The different application tiers may execute on physical machines, virtual machines, cloud instances or containers, and each tier may involve a different application server or middleware layer (e.g., front-end based on JBoss, core application based on Oracle WebLogic).

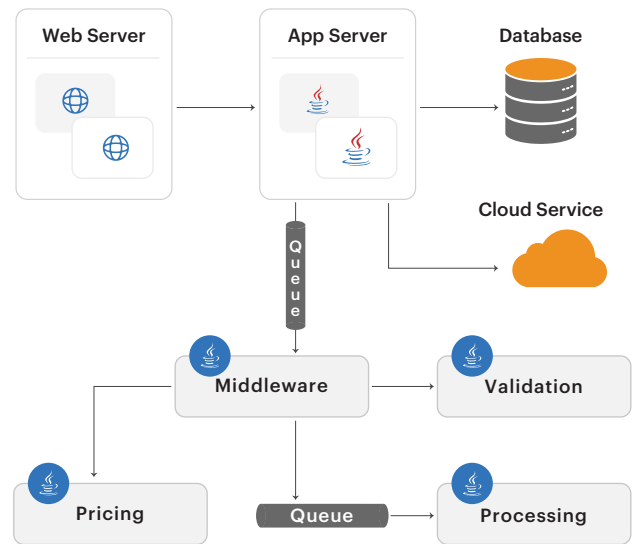


Figure 1: A complex, multi-tiered application has several application and middleware tiers with interactions among them.

- » Applications may also have external dependencies – e.g., payment gateways, 3rd party web services, etc. which may not be under the control of your organization.
- » It is also common to have applications that make use of different technologies. The front-end could be based on Node.js while the business logic may be in Java.

Because of the inter-dependencies between tiers in a multi-tier application, a slowdown in one tier (e.g., the database), can result in slowness in many of the other tiers (e.g., front-end, middleware, etc). External dependencies can also slow down transaction processing and result in increased latency. Even in the case of internal dependencies, the different tiers involved may be controlled by different IT teams (e.g., application team, database team, messaging team, system team, etc). All of this means that determining why an application is slow can be a huge challenge.

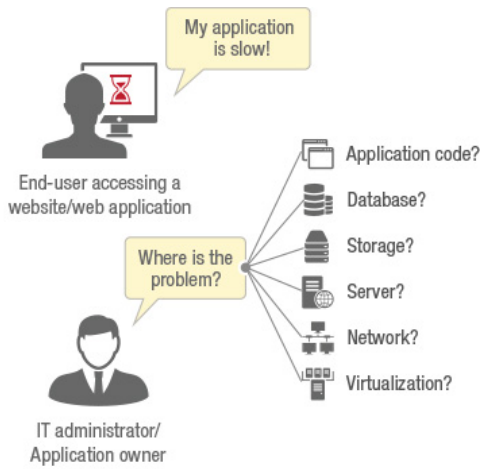


Figure 2: The toughest IT problem that admins face is users complaining that their applications are slow.

Common questions that must be answered when diagnosing an application slowness issue include:

- ◆ Is it an application logic/code-level issue and if so, which module/line of code?
- ◆ Is it due to an inefficient database query, or a slow external service call?
- ◆ Is it a configuration bottleneck in the application server?
- ◆ Is it a sizing issue in the Java Virtual Machine?
- ◆ Is it an infrastructure issue – network, storage, server, etc.?

This is where APM tools help in pinpointing the exact source of a performance issue.

Three Focus Areas for Java APM

There are three key areas to focus on when it comes to Java APM:

- **Monitoring the Java virtual machine (JVM):** The Java Virtual Machine (JVM) is the core of the Java application architecture. It interprets and translates the Java bytecode

into operations on the host platform. Because the Java middleware - including application servers such as Tomcat, JBoss EAP, WildFly, GlassFish, IBM WebSphere, Oracle WebLogic - runs on the JVM, a performance issue in the JVM has a significant impact on business services it supports. Monitoring of the JVM is an integral part of any Java application performance monitoring strategy. IT Ops and DevOps teams use JVM performance metrics to troubleshoot server-side bottlenecks. Developers and architects can also benefit from JVM monitoring by uncovering code-level issues.

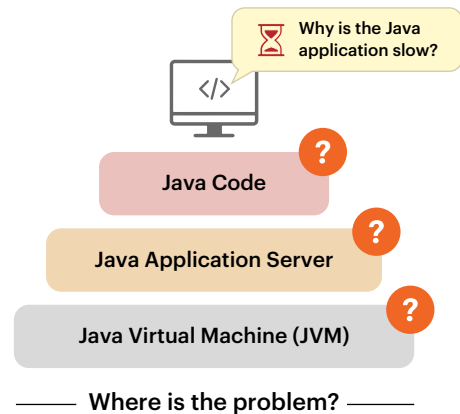


Figure 3: Java APM covers monitoring of the JVM, Java application server and Java code.

- **Monitoring the Application Server:** Application servers provide out of the box support for developing web-based Java applications. They include support for Java Servlets, Java Server Pages, WebSockets, Struts, thread pools and database connection pools, and other application constructs that greatly simplify and standardize application development. Performance bottlenecks can occur due to application server configuration limitations (e.g., insufficient number of worker threads in the server configuration, excessive queuing in the application server, database connection pool issues, etc). APM tools must provide out of the box monitoring for common Java application servers.

- » **Monitoring of the Application Code:** Inefficiencies in the application code can affect application performance. Infinite loops, inefficient code logic, database queries that don't use the right indexes, external service calls, absence of caching repeatedly used objects, etc., can all result in application slowdown. For years, a big challenge in monitoring application code has been the need to modify the application code in order to allow monitoring to be done. This can be time consuming and cumbersome and requires cooperation between the development team and the operations team. The advent of bytecode instrumentation has allowed modern APM tools to monitor application code performance without needing any changes to the applications themselves.

What is Bytecode Instrumentation?

Modern APM tools use a technique called dynamic bytecode instrumentation which allows them to modify the code of Java applications “on the fly” as they are loaded into memory via a Java agent. It may seem surprising that applications can modify themselves at runtime, but that is how the JVM works because Java is a dynamically loaded language. APM tools have capabilities to dynamically attach a Java agent to an already running JVM, such that you don't need to edit any startup scripts or even restart the application server. In a nutshell, the code is altered – not only on disk but dynamically in the runtime memory.

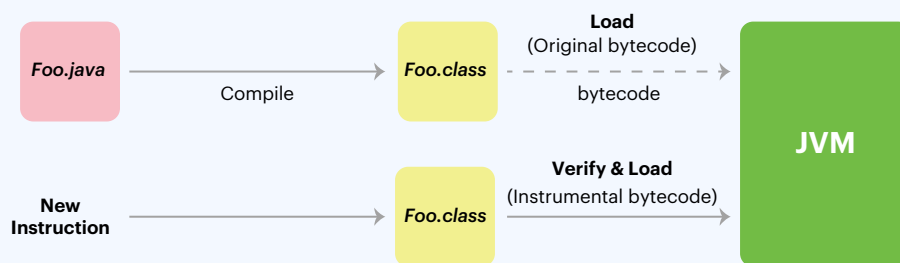


Figure 4: How bytecode instrumentation works.

Bytecode instrumentation is a well-known technique that has existed since Java 1.5. There are open source libraries such as ASM, cglib, javassist and Byte Buddy available to implement Bytecode manipulation. While the mechanism of implementing bytecode instrumentation is straightforward these days, there is a lot of complexity in actually implementing a monitoring solution using bytecode instrumentation.

To ensure that the overhead of instrumentation is minimal, APM tools must determine:

- ◆ Should each Java method be instrumented? If not, which ones should not be instrumented?
- ◆ How long should the instrumentation of a transaction run for?
- ◆ How to determine when to stop instrumenting a transaction?
- ◆ Since the application server could be handling thousands of transactions, where will the results of instrumentation be stored? Where will these transactions be aggregated and how often?
- ◆ How to perform aggregation without needing to synchronize among threads, which can be expensive?
- ◆ How can this be achieved without affecting the response time experienced by end users?

eG Enterprise: Delivering Full-Stack Java APM

The eG Enterprise observability solution from eG Innovations provides full-stack visibility into Java applications and their performance. While user experience monitoring can be achieved with synthetic and real-user monitoring, eG Enterprise also provides extensive application deep-dive and insights.

In-depth JVM Monitoring

eG Enterprise monitors most commonly used types and versions of JVMs. Key monitoring capabilities include:

» **Thread level monitoring** to report total threads, runnable threads, blocked threads, deadlocked threads and so on. By tracking total threads over time, one can identify thread leaks in an application. Blocked and deadlocked threads are often an indicator of performance issues. Stack traces are provided so IT operations teams can quickly identify problem areas in the application and inform the development teams.

- » **CPU monitoring in the JVM:** The CPU used by each thread of the JVM is tracked and high CPU consuming threads are identified. This can be used to detect run-away threads that may be responsible for slowness of the application. Stack traces captured over time help identify the code snippet/method that could be the cause of the problem.
- » **Track garbage collection activity in the JVM:** eG Enterprise allows you to easily track all garbage collection activity in the JVM.

- ◆ Identify when garbage collection happened, how often it is happening, how much memory is being collected each time and how long the garbage collection activity is running for.
- ◆ Identify times when garbage collection is taking too long and adversely affecting Java application performance.
- ◆ Compare the performance of different garbage collection settings and determine the optimal setting for each application, using historical and real-time data, plus custom reports.

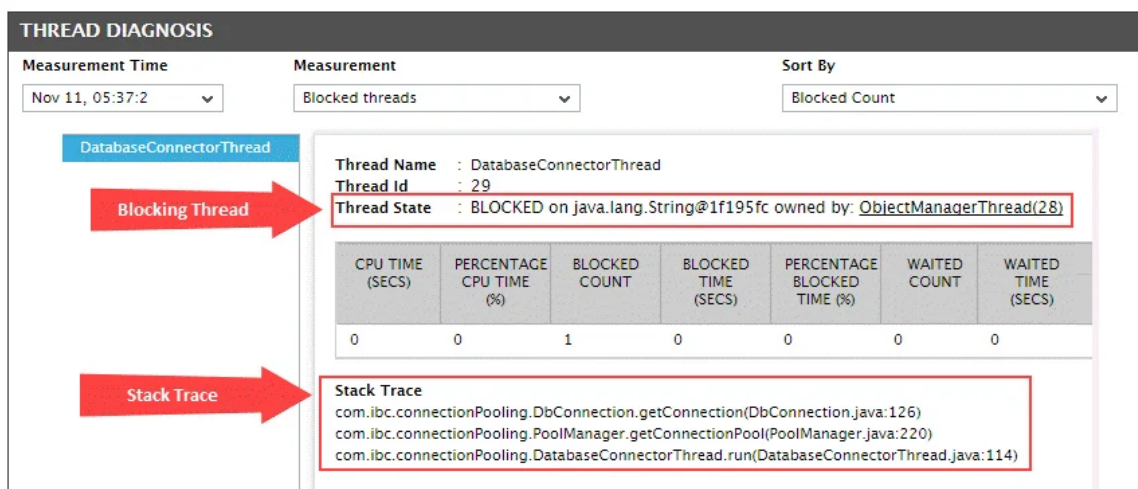


Figure 5: Stack trace of Java threads showing thread blocking.

➤ **Proactively detect memory leaks in application code:** Although the JVM performs automatic memory management, it cannot counter for memory leaks in your Java applications. Memory leaks are difficult to identify because they can take several days or weeks to manifest. Identifying the cause of Java memory leaks can be challenging, especially in production. eG Enterprise periodically takes and analyzes memory dumps of the JVM and identifies the top consumers of Java heap memory. Based on the automatic analysis of Java memory usage by eG Enterprise, administrators can be alerted to potential memory leaks in the application and can easily identify the exact Java class that may be causing a memory leak.

See [JVM monitoring perfected](#) for more details about eG Enterprise's JVM monitoring capabilities.

Details of Object Instances in the JVM Heap

CLASS NAME	INSTANCE COUNT	INSTANCE PERFORMANCE	MEMORY USED (MB)	PERCENTAGE MEMORY USED
OCT 26, 04:33:51 Heap Details				
com.acme.teleportal.AcmeTeleCustomer	9603908	98.9097	146.544	75.8182
[Ljava.lang.Object;	11944	0.123	39.5119	20.4425
l	2706	0.0279	1.9081	0.9872
[C	20063	0.2066	1.6163	0.8362
[B	5601	0.0577	1.4605	0.7556
java.lang.String	11641	0.1199	0.2664	0.1379
java.lang.Class	1655	0.017	0.1652	0.0854

Figure 6: Details of objects in the JVM heap memory.

Monitoring of Java Application Servers

There is no common mechanism for monitoring application servers. Most application servers expose performance metrics through Java Management Extensions (JMX). Some application servers support custom interfaces/protocols (e.g., T3 for Oracle WebLogic, PMI for IBM

WebSphere, etc). eG Enterprise uses JMX and custom interfaces where applicable to collect performance metrics that provide an indication of application server health. Performance metrics collected through these means include:

- ◆ *Thread pool activity:* Number of threads provisioned, number in use and utilization of each pool.
- ◆ *Log monitoring:* to detect any crashes or abnormalities reported in the server logs.
- ◆ *Database connection pool utilization:* Pool capacity, connections in use, utilization level, database response time.
- ◆ *Servlet/JSP activity:* Requests and response rates, errors, average processing time, max processing time, etc.
- ◆ *Application workers:* Completed requests, pending requests, stuck requests.

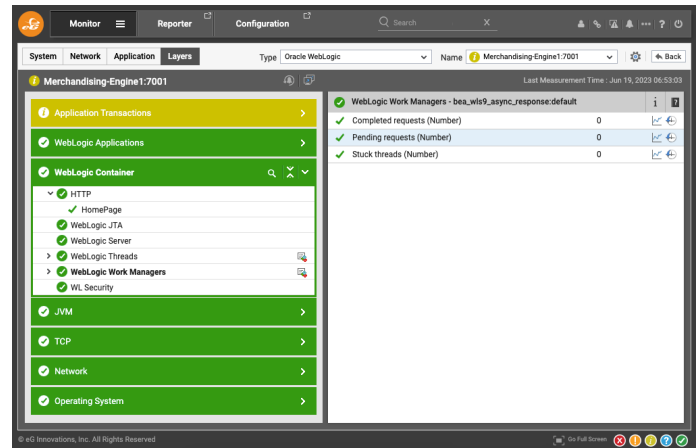


Figure 7: Monitoring model of an application server.

Monitoring of Java Application Code

To monitor Java application code, eG Enterprise uses bytecode instrumentation. A thin APM agent layer is introduced that sits above the JVM and is able to capture method calls, SQL calls, HTTP calls, external service calls, etc.

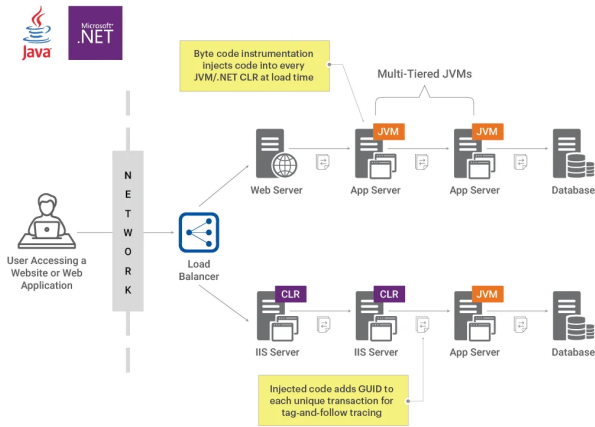


Figure 8: How the tag and follow methodology used by eG Enterprise APM works.

With bytecode instrumentation, you can trace the transaction processing at each application tier. You can identify how much time was spent in each JVM executing Java code, and importantly how much time was spent issuing and waiting for database queries, web service calls to third parties, HTTP calls to other tiers and so on.

With multi-tier applications, knowing the processing details tier-wise is not sufficient. IT operations teams and development teams expect to see end-to-end traces, so they can see how much time was spent by a specific transaction in each tier. eG Enterprise uses a “tag-and-follow” methodology for this. Transactions are tagged in the first tier they are processed at. The tag is carried forward in a HTTP request body if a request is passed from one application tier to another by the APM agent in each tier. Tagging is supported for other non-application tiers as well – for example, for message queues, the tag is part of the message header.

For each tagged transaction, eG Enterprise captures the processing time for transactions at each tier, external calls made, and time taken those calls and details of how the transaction was processed (e.g., database queries, Java stack details, etc).

It is to be noted that unlike some other APM solutions, eG Enterprise does not sample a subset of transactions. Take a typical enterprise application distributed across multiple application tiers. eG Enterprise traces all transactions across all application tiers. For each application tier, these traces are analyzed and a set of healthy, slow, stalled and error transactions are ranked from worst to best based on (a) latency and (b) first occurrences of error and stored as Top N traces for detailed diagnosis. This ranking is specific to each application tier and is per business transaction. This ranking based Top N selection helps administrators focus on what’s problematic for that specific tier since each application tier could be experiencing different performance bottlenecks.

To put together the complete transaction flow, all the traces collected at each application tier are sent to the central management server, where the transaction processing flow graph is constructed (see Figure 9 below). As you can see, from the eG Enterprise web interface, an admin can see the total transaction processing at each tier.

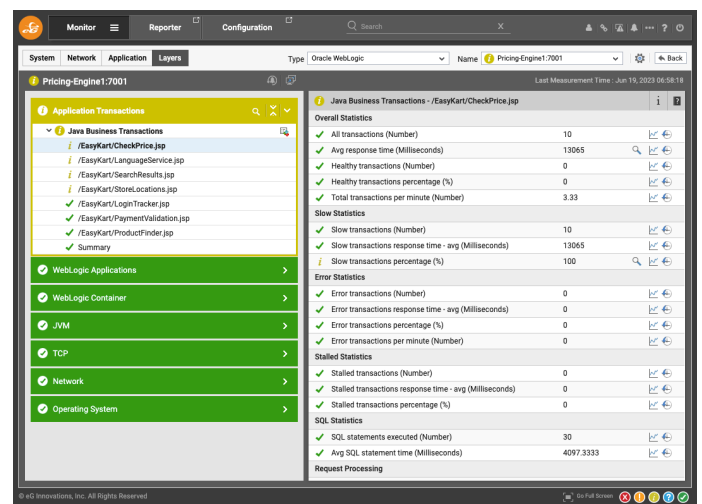


Figure 9: Details of transactions processed by an application tier. All transactions are analyzed (not just a subset).

From the Figure 9, you can see that there is an alert because slow transactions are high. By clicking on the detailed diagnosis (i.e., the magnifying glass to the right of the metric), an admin can see a list of transactions whose processing is slow.

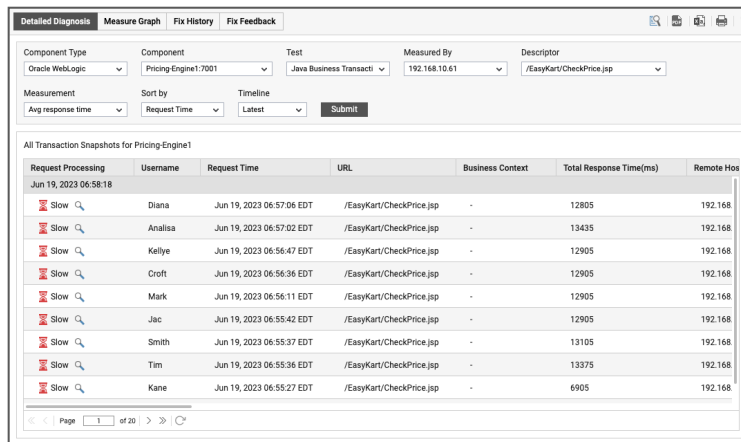


Figure 10: Details of slow transactions processed.

Drilling further into any specific transaction (from the magnifying glass again), one can see how the transaction was processed. The flow graph shows the tiers involved in processing and the time taken at each tier. From this figure, it is apparent that processing of database queries is the bottleneck. Three queries were executed, taking a total time of 12,035 msecs.

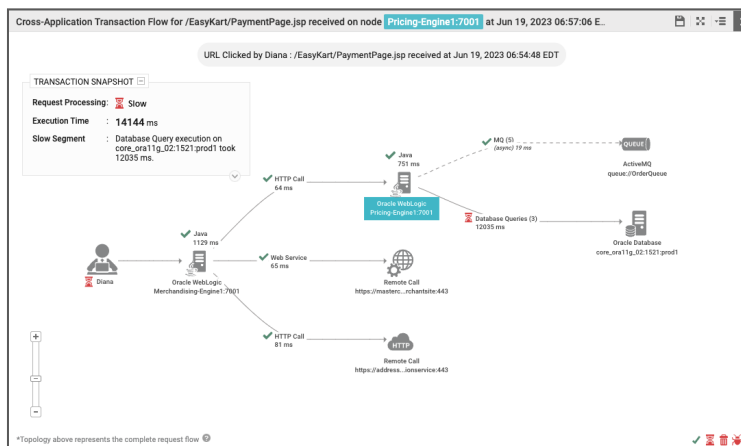


Figure 11: Transaction processing flow graph showing time spent in each tier.

Why is Transaction Tracing Effective?

Transaction tracing has become very popular for many reasons:

- ◆ It provides in-depth insights into distributed application performance, without needing any change to the application code. Even IT operations teams can implement transaction tracing without needing any assistance from development teams.
- ◆ Application insights can be obtained without needing any permissions from other administration teams – e.g., database team, server team, etc.
- ◆ Breakdown of transaction processing time by tier gives conclusive proof to answer today’s toughest IT question which is “Why is the application slow?”.

By clicking on the Database Queries in Figure 11, an admin can see the details of the slow queries that were processed.

Call Drill Down for /EasyKart/CheckPrice.jsp on Pricing-Engine1:7001

Summary

Execution Analysis

Hot Spots

Slow SQL Queries

Error Details

Remove Call Details

3 out of 3 SQL queries are slow/have errors : Total SQL time : 12035 ms

QUERY TYPE	QUERY DETAILS	COUNT	AVG EXECUTION TIME (ms)	TOTAL EXECUTION TIME (ms)	% TIME	ERROR	DATABASE	RELATED ALERTS
SELECT	select * from AP_IN...	1	6135	6135	47.91%	-	SQL	Q
SELECT	SELECT * FROM lby...	1	4357	4357	34.03%	-	SQL	Q
SELECT	SELECT * FROM lby...	1	1543	1543	12.05%	-	SQL	Q

Query

```
select * from AP_INV_SELECTION_CRITERIA_ALL where checkrun_name = '3EB64'
```

Figure 12: Details of queries issued to the database tier by the application.

As you can see above, eG Enterprise APM provides a lot of detail about transaction processing that makes it simple to answer the question “Why is my application slow?”.

Database monitoring tools also provide details of slow database queries. How is the information provided by APM tools different?

- ◆ Database monitoring tools provide insights from the database perspective and are intended for use by the database administration team. A database server may be used by many applications. The top queries across applications is listed, so a database admin can analyze them and determine how to optimize the database. Queries and accesses seen at the database tier may not even be related to specific applications – e.g., a backup job can cause a lot of IOPS on the database server.
- ◆ Insights provided by APM tools are specific to each application and how it accesses/uses the database server. These insights can be obtained without needing any additional permissions on the database tier. Application developers can use these insights to optimize how their application uses the database (e.g., detect queries that may not be optimized).

While the example above showed a multi-tier application where each of the application tiers used Java technology, the same functionality is also available if any of the application tiers uses Microsoft .NET, Microsoft .NET Core, Node.js or PHP as well. Also, it does not matter where an application tier runs – i.e., on a physical machine, virtual machine, cloud instance, or a container.

Since there is a lot of data collected for each transaction at each tier, it is practically impossible to store all the details for display. Doing so would require a very large database.

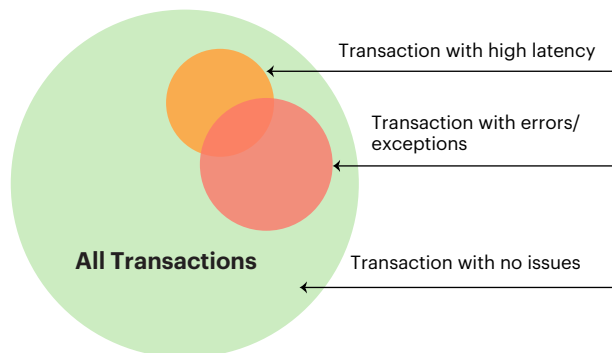


Figure 13: During typical operation of an application, only a small percentage of transactions are processed slowly or with errors.

Typically, a high percentage of transactions are processed by an application without any errors, and a small subset of transactions is processed slowly or has been processed with exceptions (see Figure 13). Therefore, eG Enterprise provides administrators with controls to decide whether to collect detailed stack traces for healthy transactions or not (e.g., if a transaction is healthy, it was processed quickly, and it may not be necessary to collect detailed traces). By turning off details for healthy transactions, one can save on database storage of a large number of application stack traces. The processing time value below which a transaction is considered healthy is also configurable for each application and for each transaction. For example, if a request takes more than 10 seconds to be processed, it will be considered a slow transaction and one that takes more than 60 seconds can be deemed to be a stalled transaction.

There are a number of other configuration settings that can be tuned when configuring eG Enterprise, as you can see from Figure 14.

- ◆ The maximum execution time cutoff, i.e., the maximum time for which eG Enterprise APM waits for a transaction to be processed, is configurable. The larger the value, the more time that eG Enterprise APM waits for a transaction to be processed and potentially, higher its processing overhead and data storage required.
- ◆ Tracing can be enabled for each URL, or insights can be captured for a group of URLs – e.g., all payment URLs, all browsing URLs, etc. Grouping URLs also ensures that aggregate metrics are collected for groups of URLs and this also results in lower database space usage.
- ◆ The eG Enterprise agent captures detailed stack traces for the slowest N URLs/URL groups. For slow transactions, traces for the slowest N transactions in a measurement period are stored. The value of N is configurable (default value is 10 in any measurement period). For exceptions too, only stack traces for N transactions with exceptions are stored.

The larger the value of N, greater than the bandwidth used between the agent and the manager and the greater the storage required in the database. If a transaction does not fall in the top N transactions for a node (i.e., an application component of one tier), while the processing time, status and details of the transaction are included in the total values, the individual traces will not be available in eG Enterprise. So in the transaction flow graph, the corresponding node itself may be missing. If you see a large number of such fragmented traces, you can consider increasing the value of N.

TEST PERIOD	3 mins
HOST	8d71423fc183
PORT	8080
BTM PORT	13931
IP MASKING CHARACTER	x
BUSINESS TRANSACTION NAMING RULES	None
USERNAME/BUSINESS CONTEXT CONFIGURATION	None
MAX URL SEGMENTS	3
EXCLUDED PATTERNS	.tif*.otf*.woff*.woff2*.eot*.cff*.afm*.lwf*.fif*.fon*.pfm*.pfb*.std*.pro*.xsf*.jpg*.jpeg*.jpe*.jif*.jfi*.j2*.j2k*.jpf*.jpx*.jpm*.jxr*.hdp*.wdp*.mj2*.webp*.gif*.png*.apng*.mng*.tiff*.tif*.xbm*.bmp*.dib*.svg*.svgz*.mpg*.mpeg*.mpeg2*.avi*.wmv*.mov*.rm*.ra
METHOD EXECUTION CUTOFF (MS)	250
SQL EXECUTION CUTOFF (MS)	50
HEALTHY URL TRACE	<input checked="" type="radio"/> Yes <input type="radio"/> No

Figure 14: Configuration settings that influence how eG Enterprise APM works in each tier.



The intention behind restricting the traces to the top N is so that administrators and development team responsible for each tier can quickly determine the requests that took the most time in that tier and from the stack traces, they can see how to optimize the application to improve performance and ultimately, enhance user experience.

[eG Enterprise's distributed transaction tracing capabilities for Java](#) are a key to providing code-level visibility for application analysis, optimization and troubleshooting.

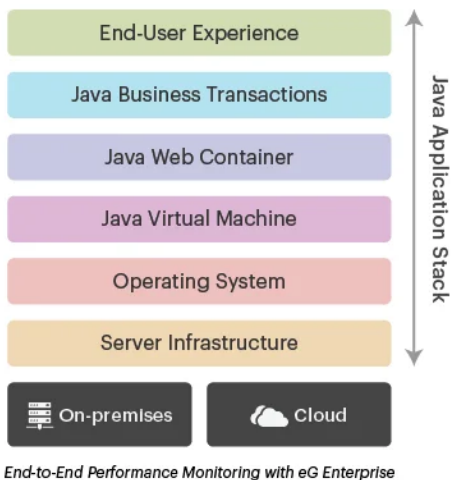


Figure 15: Full-stack visibility into Java applications.

What eG Enterprise Tracing of Application Code Reveals?

Using eG Enterprise APM, IT operations and DevOps teams can:

- ◆ *Determine the application workload:* Transaction rate (requests/min) is reported for each URL / URL group, so any change in workload patterns can be detected.
- ◆ *Report the responsiveness of the application at each tier of the application delivery chain:* Average response time, percentage of slow and stalled transactions provide insights to help IT operations teams identify problematic tiers.
- ◆ *Provide breakdown of processing times at each tier:* Besides reporting the processing time of a transaction, eG Enterprise APM also provides the details of where this time was spent – e.g., in processing database queries, third party calls, other external calls, etc. This helps identify any external bottlenecks.
- ◆ *Get details of application exceptions:* This can highlight errors in application processing that if proactively highlighted can prevent application failures down the line.

Converged application and infrastructure monitoring with eG Enterprise

While in-depth application insights are necessary and useful, they are not sufficient. When an

be in the infrastructure – e.g., the application could be running on a VM which is hosted on an overloaded hypervisor, or the application could be running on a cloud instance that has zero CPU credits. To troubleshoot application slowness accurately and efficiently, a converged approach to application and infrastructure monitoring is necessary. This is what eG Enterprise employs.

Applications and infrastructure components can be monitored from the same console. The same universal agent deployed on a server can monitor both the application stack and the infrastructure (i.e., system metrics) – there is no need for a separate machine agent! Also the integration is not just in terms of presenting a common dashboard for application and infrastructure components.

eG Enterprise automatically discovers and maps application to infrastructure dependencies in service topology maps and embeds a patented automatic root-cause diagnosis technology that correlates between the performance of different application and infrastructure tiers and highlights where the root-cause of application slowness lies.

To highlight how a converged application and infrastructure monitoring strategy works, the slow database queries in Figure 12 are actually a result of VMware hypervisor issue with free space.

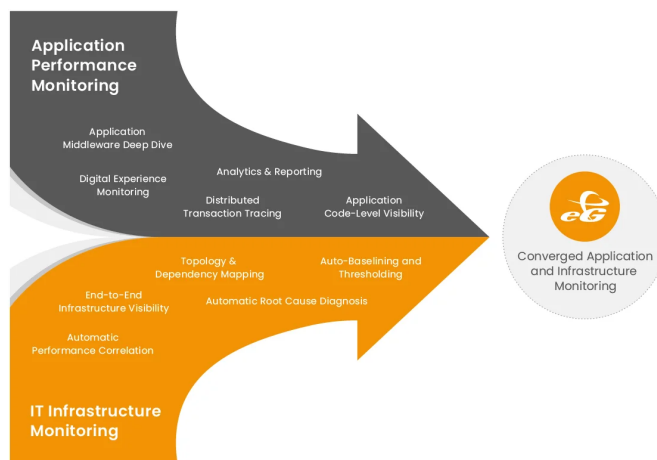


Figure 16: Converged application and infrastructure monitoring.

The database server is thin-provisioned on the hypervisor, so when the hypervisor is low on space, requests to the database server start failing. No amount of application code analysis could have identified the root-cause of this issue.

The incident management panel in eG Enterprise clearly highlights the cause-effect relationship (see Figure 17). The highest priority issue is the root-cause and indicates the space usage problem on the VMware server. The database errors and transaction slowness alarms are lower in priority, highlighting that they are secondary effects of the problem. With this level of insight, IT operations teams can troubleshoot issues quickly and efficiently and lower MTTR.

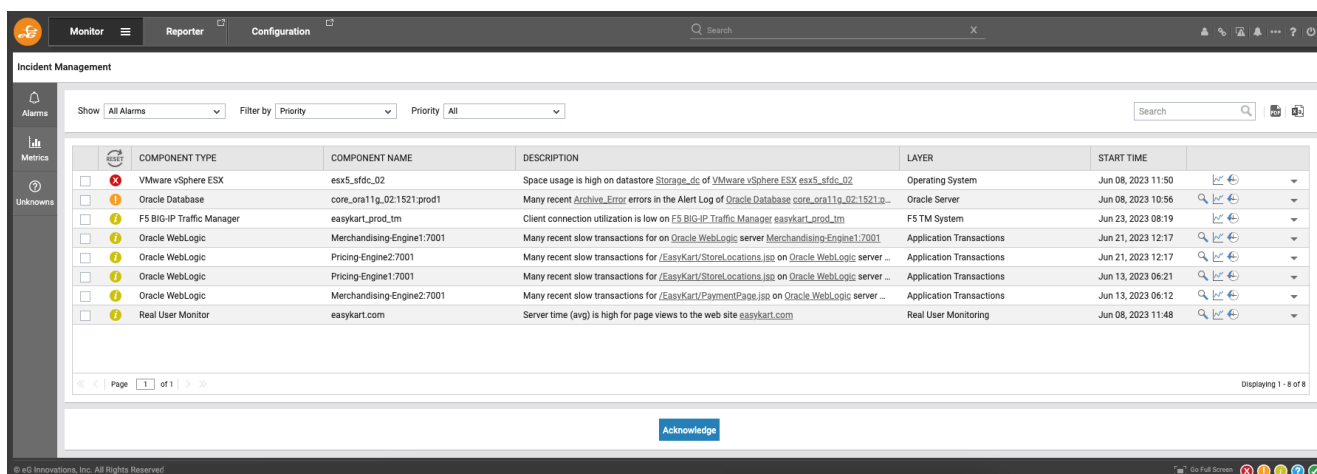


Figure 17: Root-cause diagnosis highlighted in the incident management panel of eG Enterprise.

The [Converged Application and Infrastructure Monitoring](#) technology it embeds is a key to how eG Enterprise supports tightly integrated application and infrastructure monitoring from a single pane of glass.

Conclusion

In this whitepaper, we have outlined the different challenges in monitoring Java applications and have discussed the varied capabilities included in eG Enterprise to address these challenges. The converged application and infrastructure monitoring approach adopted by eG Enterprise is the only way to conclusively address application slowness issues.

Licensed by operating systems and nodes on which it is deployed, not based on number of instances, number of CPUs/cores, or by the memory available on the monitored systems, eG Enterprise is highly cost-effective. Furthermore, it is available both as software you can deploy on-premises or as a SaaS service from the cloud. Get a 30-day free trial of eG Enterprise for application performance monitoring here: <https://www.eginnovations.com/it-monitoring/free-trial>

About eG Innovations

eG Innovations provides the world's leading enterprise-class performance management solution that enables organizations to reliably deliver mission-critical business services across complex cloud, virtual, and physical IT environments. Where traditional monitoring tools often fail to provide insight into the performance drivers of business services and user experience, eG Innovations provides total performance visibility across every layer and every tier of the IT infrastructure that supports the business service chain. From desktops to applications, from servers to network and storage, eG Innovations helps companies proactively discover, instantly diagnose, and rapidly resolve even the most challenging performance and user experience issues.

eG Innovations' award-winning solutions are trusted by the world's most demanding companies to ensure end user productivity, deliver return on transformational IT investments, and keep business services up and running. Customers include Anthem, Humana, Staples, T-Mobile, Cox Communications, eBay, Denver Health, AXA, Aviva, Southern California Edison, Samsung, and many more.

To learn more visit www.eginnovations.com

References

»» Java Application Monitoring:

[Java Application Monitoring | eG Innovations](#), [Modern Web Application Performance Monitoring \(APM\) \(eginnovations.com\)](#) and [Java Transaction Monitoring | eG Innovations](#)

»» JVM Monitoring:

[JVM Monitoring Tools – Threads, GC, Memory Leaks & more \(eginnovations.com\)](#)

»» JMX:

[JMX Monitoring | eG Innovations](#) and [How does JMX Monitoring Work | eG Innovations](#)

»» Application Server Monitoring

[JBoss Monitoring | eG Innovations](#)

[Tomcat Monitoring | eG Innovations](#)

[JBoss Performance Tuning & Monitoring | eG Innovations](#)

[10 Apache Tomcat Performance Tuning Tips and Best Practices \(eginnovations.com\)](#)

»» Java Troubleshooting:

[7 Tuning Tips to Enhance Java Application Performance \(eginnovations.com\)](#)

[How to make Java run faster - 6 Tips | eG Innovations](#)

[Top Java Performance Problems & How to Fix them | eG Innovations](#)

[Java Memory Leak: 7 Myths that SREs Need to Know \(eginnovations.com\)](#)

[Troubleshoot Java Application Issues with Java Transaction Tracing \(eginnovations.com\)](#)

[5 Java Application Monitoring Best Practices | eG Innovations](#)

»» Industry use cases:

[Seven Critical Strategies and Features for Application Performance Management in Higher and Further Education \(eginnovations.com\)](#)

Real Customer End-to-End Monitoring Case Study: [End-to-End Application Monitoring to Troubleshoot Slowness \(eginnovations.com\)](#)

A multi-cloud services example including payment gateway failures: [Monitoring and Troubleshooting Multi-cloud Infrastructures \(eginnovations.com\)](#)