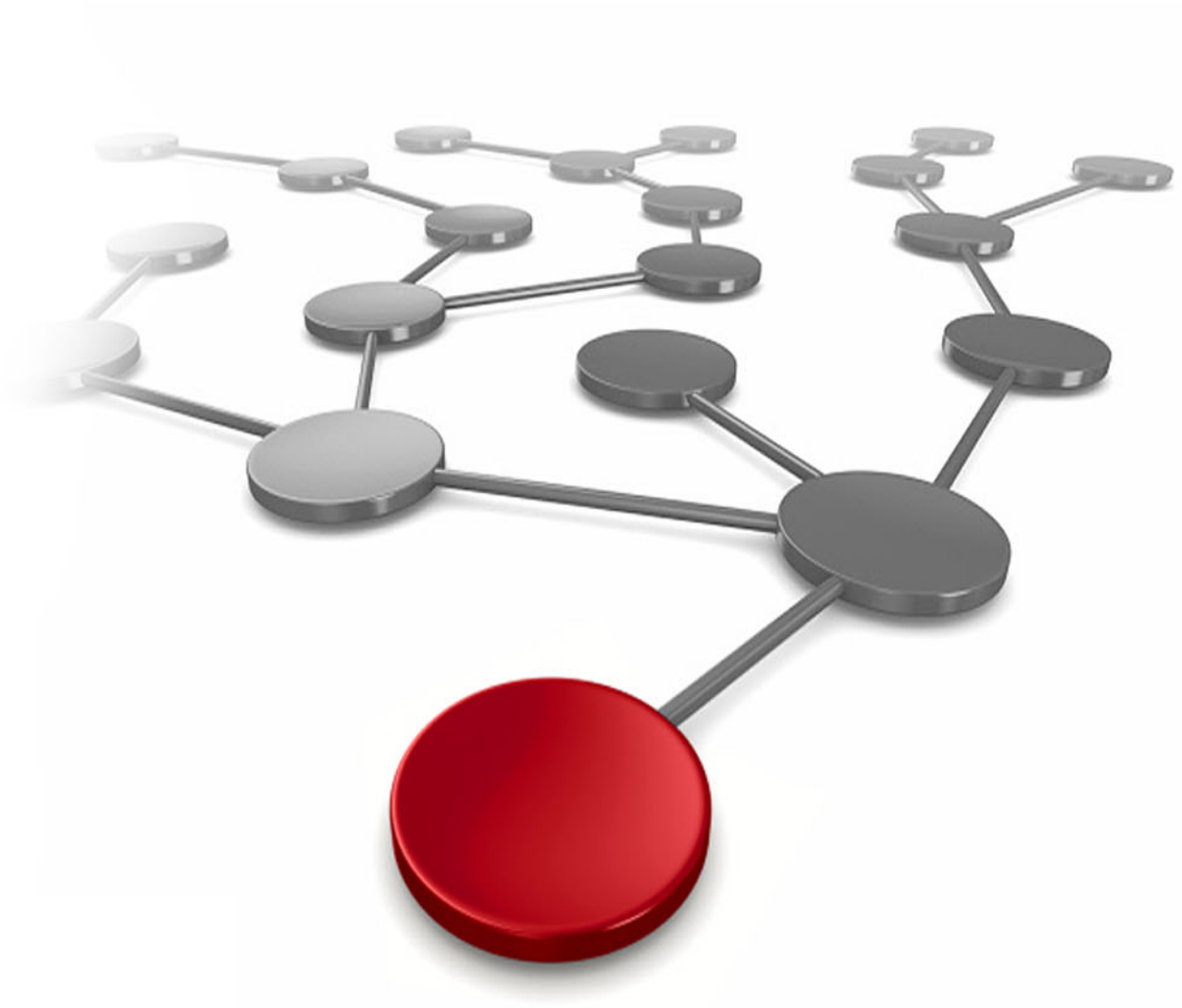
Redbooks
ibm.com/redbooks

Enterprise Java Monitoring on z/OS with OMEGAMON

A Practical Guide to Managing JVM Performance on z/OS

Christopher Walker

Nigel Williams



IBM

Redpaper

International Technical Support Organization

Enterprise Java Monitoring on z/OS with OMEGAMON: A Practical Guide to Managing JVM Performance on z/OS

February 2017

REDP-5429-00

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (February 2017)

This document was created or updated on February 23, 2017.

Contents

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

CICS®	MVS™	UrbanCode™
CICS Explorer®	OMEGAMON®	VTAM®
Cognos®	RACF®	WebSphere®
DB2®	Redbooks®	z Systems®
IBM®	Redpaper™	z/OS®
IMS™	Redbooks (logo)  ®	z13™
Language Environment®	Tivoli®	

The following terms are trademarks of other companies:

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redpaper™ publication will help you install, tailor, and configure IBM OMEGAMON® for JVM on IBM z/OS®. You can use OMEGAMON to recognize and resolve problems in monitoring Java resources on z/OS, including within IBM CICS®, IBM IMS™, and z/OS Connect EE regions. A discussion on the growth of Java on z/OS is provided and explanation on the reasons why monitoring Java resources is critical to any modern z/OS environment.

Authors

This paper was produced by a team of specialists from around the world.

Christopher Walker is an Offering Manager with IBM in the US. He has worked for IBM for over 15 years, mostly as a software developer. He initially worked in the CICS group at Hursley, United Kingdom, and later as part of IBM z Systems® monitoring in Research Triangle Park, North Carolina. He is now responsible for the coordinating the strategic direction of the OMEGAMON family of products on z/OS. He has a MEng in Software Engineering from Sheffield University, UK.

Nigel Williams is a Senior Software Engineer with IBM in the US. He has nearly 30 years experience in system performance management software. He has worked at IBM for 12 years since IBM acquired Candle Corporation where he had worked for 17 years before that. Most of his career has been spent in software development on z/OS, particularly with Java, IBM WebSphere®, CICS, IMS, and IBM VTAM® products. However, he has also developed products for multiple UNIX variants and Windows, including end-to-end response time monitors, terminal emulators, and network management tools. He is currently the lead developer and architect of OMEGAMON for JVM on z/OS. He holds a BSc degree in General Science from Durham University, England.

Thanks to the following people for their contributions to this project:

Nigel Williams
Matthieu Dalbin
IBM France

Francesco Marinucci
IBM Italy

Brennan Grusky
IBM US

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks® publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks
- Send your comments in an email to:

  redbooks@us.ibm.com
- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks
- Follow us on Twitter:

  http://twitter.com/ibmredbooks
- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm
- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Overview

This chapter introduces OMEGAMON for JVM on z/OS and discusses the structure of the book. It includes the following topics:

- Objectives of this publication
- Java on z/OS
- IBM OMEGAMON for JVM on z/OS
- Description of application environment

1.1 Objectives of this publication

This book was written to demonstrate the capabilities of IBM OMEGAMON for JVM on z/OS. The product was installed in a test environment on several JVMs, both stand-alone and in subsystems like CICS and IMS. Some of these JVMs were poorly configured or allowed to use inappropriate defaults in order to use OMEGAMON to expose these problematic configurations. Some applications were also written with flaws that might be found in Java programs that adversely affect performance, to the same end.

1.2 Java on z/OS

Java is now one of the world's most important programming languages. Twenty years after it was first introduced, there are an estimated 9 million Java developers worldwide with applications deployed on everything from mobile devices to mainframes. A major reason for this is the portability of programs written in Java. A program written and compiled on one operating system hardware platform can run unchanged on any other system that runs an equivalent version of the Java virtual machine (JVM).

1.2.1 Java on z/OS applications and workloads

Consequently, Java has become an increasingly important component of z/OS. For data serving and transaction processing, which are traditional strengths of the z Systems platform, Java has become foundational. Subsystems like CICS, IMS, IBM DB2®, and IBM MQ now typically host one or more JVMs and execute Java programs as enterprises seek to extend and modernize their business logic (Figure 1-1). Furthermore, many new z/OS application service environments run entirely in dedicated stand-alone JVMs.

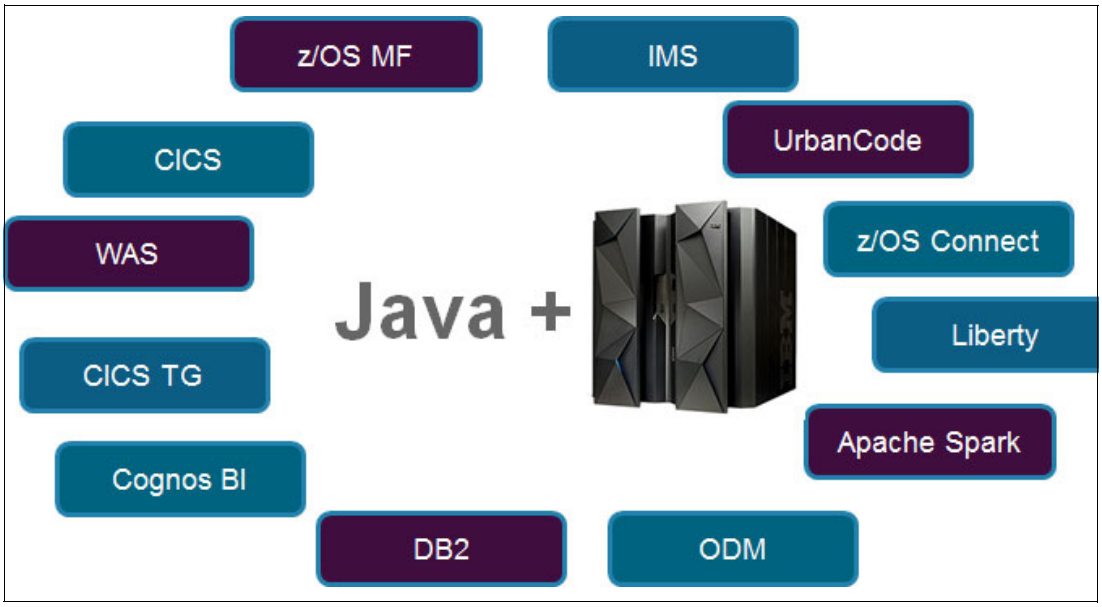


Figure 1-1 Examples of Java applications available on z/OS

WebSphere Application Server is one such example that most people are familiar with. WebSphere Liberty Profile, an Open Service Gateway Initiative (OSGi) application server environment, provides a quicker, smaller, and composable application serving environment is now common on z/OS.

Applications written in Java and deployed on z/OS have an inherent advantage over distributed applications through collocation with existing applications and databases. This configuration can result in better response times, greater throughput, and reduced system complexity when driving, for example, CICS, DB2, or IMS transactions.

IBM z/OS Connect Enterprise Edition (EE) is an example of a modern Java on z/OS application. Running on WebSphere Liberty, together with advanced Java tools, z/OS Connect EE allows programmers to easily create RESTful API interfaces to existing enterprise programs for use by cloud and mobile-based applications. This technique provides enterprises with an invaluable application modernizing capability that enables use of existing assets (and years of investment) while embracing the new advantages offered by cloud computing.

IBM Operational Decision Manager (ODM) is a family of products used by IT and business users to create and manage business decision logic. ODM can run in a stand-alone JVM on z/OS or within a WebSphere Application Server instance.

Java is also being used for batch programs, either stand-alone using UNIX System Services BPXBATCH and JZOS toolkit launcher; or with WebSphere Application Server using modern batch systems like Compute Grid. Other applications that use Java natively on z/OS you might be familiar with include z/OS Management Facility (z/OSMF), CICS Transaction Gateway, IBM Cognos® Business Intelligence, IBM UrbanCode™ Deploy, and Apache Spark.

1.2.2 Why Java on z/OS is critical to z modernization

The growth of Java of z/OS can be attributed to several factors: The number of Java developers, as already mentioned, is attractive to many enterprises running z/OS who are facing increasing challenges and costs to maintain applications developed over several years in languages such as COBOL. Introducing Java applications as part of a z modernization strategy to interact with these *systems of record* can complement and where appropriate, replace existing applications that might be difficult to maintain.

For z/OS to be a suitable platform for Java development, tooling needs to be similar to those found on distributed platform. IBM Developer for z Systems is an Eclipse-based integrated development environment (IDE) for creating and maintaining z/OS applications in many languages, including Java. Because it integrates with other Eclipsed-based tooling for z/OS (and will be familiar for application developers who have limited z/OS experience), this platform removes many barriers to entry for enterprises that want to use their existing investments in z/OS and make rapid, agile development with modern processes.

Another cost factor is the expense of running Java workloads, and here lies a major advantage to the adoption of Java natively on z/OS versus a migration off-platform: All Java code, including the JVM operation, are zIIP eligible. This gives the opportunity to invest in new workloads that can benefit from their proximity to existing applications and databases on z/OS, and can be managed by existing teams without contributing to the 4-hour rolling average utilization cost.

1.2.3 Performance considerations

A common misconception of Java technology is that it is slow or inefficient in comparison to other languages. This unwarranted reputation probably originates from the early days of Java when the development of compilation and runtime environments were in their infancy. The portability of Java code, meaning an application compiled on one operating system could be executed in a compatible runtime environment on any other operating system, meant initial performance metrics usually compared unfavorably with natively compiled applications.

Since then, there have been a number of improvements, both in the Java compiler and runtime technology and with the Java language, that invalidate this negative perception of Java on z/OS. The following are examples of these improvements:

- Improvements in just-in-time (JIT) compilation that enables frequently used code to be transformed into an optimized form that runs faster on the CPU's native instruction set.
- Updates to Garbage Collection policy and algorithms that reduce the time during which a JVM must pause to reclaim heap memory.
- Exploitation of z Systems hardware improvements such as accelerators, larger page sizes to exploit 64-bit architecture and better runtime instrumentation.

The current IBM z13™ hardware, introduced in 2015, provides a number of technical improvements used by Java such as simultaneous multithreading (SMT), single-instruction, multiple-data (SIMD), improved CP Assist for Cryptographic Function (CPACF), and new native instructions. In conjunction with Java 8, performance improvements can be seen of up to 2X in throughput-per-core for security-enabled applications and up to 50% for other applications.

The result of this investment, coupled with the security and reliability advantages z Systems are known for, means z/OS is probably the optimum platform to run business-critical Java workloads.

1.2.4 Challenges to monitor and manage Java on z/OS

This increase of Java applications and workloads on z/OS presents new challenges to system programmers, operations staff, and subject matter experts who are responsible for the monitoring and management of production systems. These roles need to ensure that the resources are correctly allocated and problems can be identified quickly and resolved with minimal impact to users. They are used to tools that allow them fast detailed access to z/OS resources or that of subsystems such as CICS or IMS. With Java, there is a whole set of new resources to manage, often within an existing subsystem, and might be uncomfortable with the idea that a JVM represents a “black box” within their visibility. Indeed, a first question many might ask is “How much Java are we running on our systems?” To address this and other questions, OMEGAMON for JVM on z/OS was developed.

1.3 IBM OMEGAMON for JVM on z/OS

Because the Java run time is an independent, autonomous component, traditional z/OS monitoring tools will not tell you much about the performance of a JVM, or the resources it consumes. OMEGAMON for JVM on z/OS is a member of the OMEGAMON family of products designed specifically to collect and analyze performance metrics for z/OS-based JVMs. Using these metrics, OMEGAMON can warn operations staff, application owners, and subject matter experts if a JVM is performing suboptimally, impacting throughput, response time, or service availability.

It uses instrumentation that is built into every IBM J9 JVM since Java 1.5, originally intended for use with the Java Health Center. No byte code injection is used and no TCP/IP ports are required for use with OMEGAMON. The Health Center instrumentation is enabled with a JVM Tool Interface (JVMTI) agent using Java command line options.

An OMEGAMON-supplied Java agent is used to connect internally to the Health Center agent. The role of the OMEGAMON Java agent is to transmit the performance metrics to an OMEGAMON Monitoring Agent by way of a common Collector. Only one OMEGAMON

Monitoring Agent/Collector pair is required per LPAR, regardless of the number of JVMs being monitored. However, additional Monitoring Agent/Collectors can be started for organizational or administrative purposes if necessary.

Note: IBM OMEGAMON for JVM on z/OS is V5.4.0 available as a stand-alone offering or as part of any one of z Systems Service Management Suites:

- IBM Service Management Suite on z/OS
- IBM OMEGAMON Performance Management Suite on z/OS
- IBM OMEGAMON on z/OS Management Suite

Support for IBM System Automation on z/OS V3.5, allowing the provision and management of the OMEGAMON for JVM agent, is provided with PTF UA82383.

The OMEGAMON Monitoring Agent/Collector pair connects into the IBM Tivoli® Monitoring infrastructure either through a remote Tivoli Enterprise Monitoring Server or direct to the hub Tivoli Enterprise Monitoring Server. As with other OMEGAMON products, OMEGAMON for JVM presents data on the graphical Tivoli Enterprise Portal client and the OMEGAMON Enhanced 3270UI (an integrated 3270 centralized management interface) with data made available through attribute groups providing users with workspaces and out-of-the-box situations to be alerted to abnormal conditions. These features enable monitoring by exception. Both user interfaces are fully customizable meaning users can adjust tables, charts, and thresholds as appropriate to their environment. See Figure 1-2.

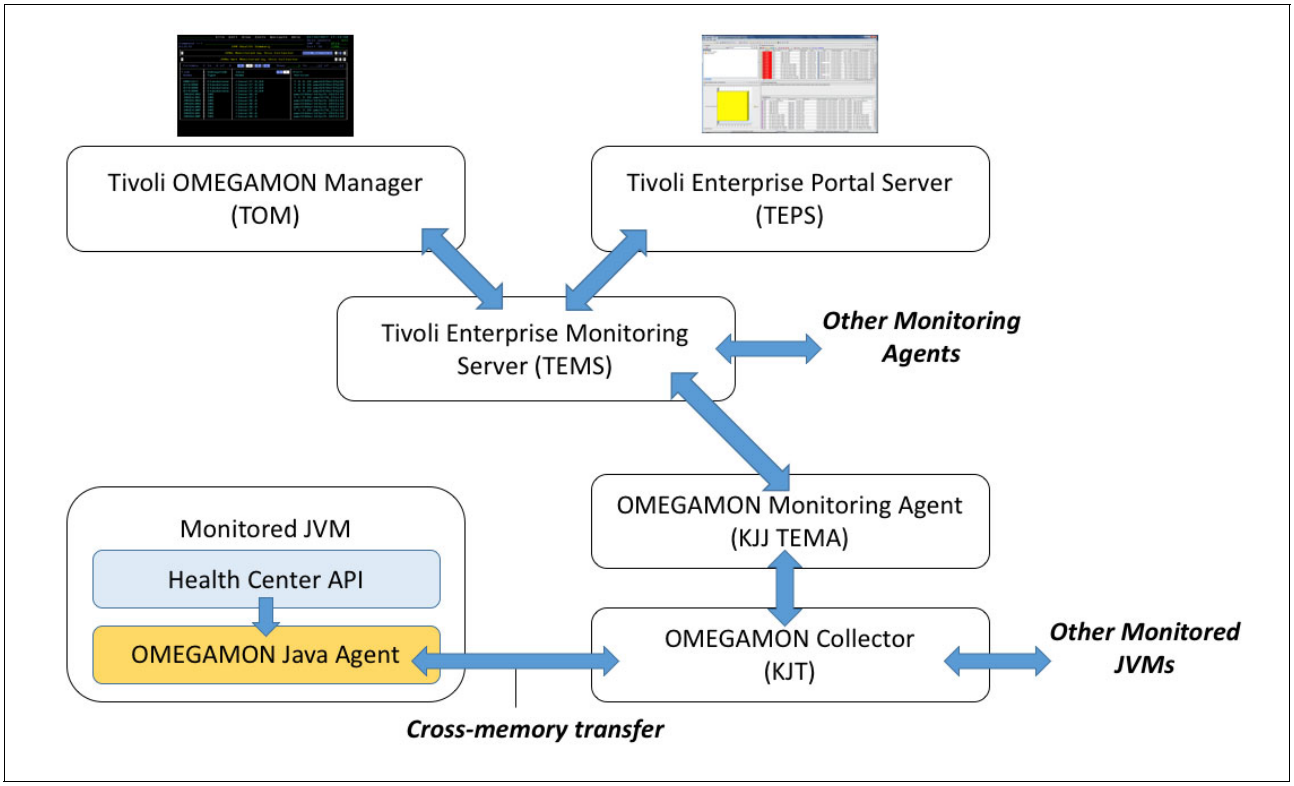


Figure 1-2 OMEGAMON for JVM on z/OS architecture

In addition to near real-time data shown on the user interface, OMEGAMON can be configured to periodically collect data from monitored JVMs and store this data for viewing later or report generation. This is known as *historical data collection* and is particularly useful for investigating problems that occurred in the recent past. Short-term history is available in both the Enhanced 3270UI and in the Tivoli Enterprise Portal. In addition, the Tivoli Enterprise Portal can be configured to produce summarized reports over longer periods by using the Tivoli Data Warehouse.

1.3.1 Data provided by OMEGAMON for JVM V5.4.0

OMEGAMON collects data in groups of *attributes*. Queries are issued from the user interface or on a scheduled basis to collect the latest data from one or more monitored JVMs. In OMEGAMON for JVM V5.4.0, attribute groups are provided for the following JVM data points:

- JVM Health Summary
- JVM Environment Data (including environment variables and CLASSPATH data)
- Garbage Collection (GC)
- Java Heap Usage
- Native Memory Usage
- Thread Details
- Lock/Synchronized Code Utilization
- CPU Usage (including zIIP offload details)

Additionally, attribute groups are provided specifically for z/OS Connect EE address spaces. These provide details about the services available through this z/OS Connect EE instance, including the average response time and data throughput size. Details of the five slowest responding requests for each service within the last sample period is also collected for further investigation.

1.3.2 Situations

Situations are a core concept in operations system management with OMEGAMON. A *situation* is a set of conditions that when met, creates an event. A condition is usually a performance or system metric that is compared to a threshold value. Conditions are examined automatically at predetermined intervals, or triggered by system events. When triggered, the event can generate an alert to the operator notifying that person of a problem, or perform an automatic action (for example, issue a log message or system command). Every OMEGAMON product delivers a number of predefined situations. These product-provided situations check for common conditions that typically cause problems in many enterprises.

Users can use these product-provided situations as-is, modify the condition thresholds to more accurately reflect their own environment, or create brand new situations to meet unique needs. Examples of product-provided situations delivered in OMEGAMON for JVM V5.4.0 capture conditions such as the following, among others:

- High GC pause times
- High heap occupancy after GC
- Blocked threads
- High lock miss percentage
- High CPU
- High deferred zIIP utilization

Note: Situations can be viewed, modified, or defined in the Tivoli Enterprise Portal and, if PTF UA83356 is applied, the OMEGAMON Enhanced 3270 UI. Before this update, the Enhanced 3270UI provided only situation status viewing.

Situations, including product-provided situations, must be started and distributed to one or more managed systems to function. This process is done with the Situation Editor in the Tivoli Enterprise Portal (Figure 1-3).

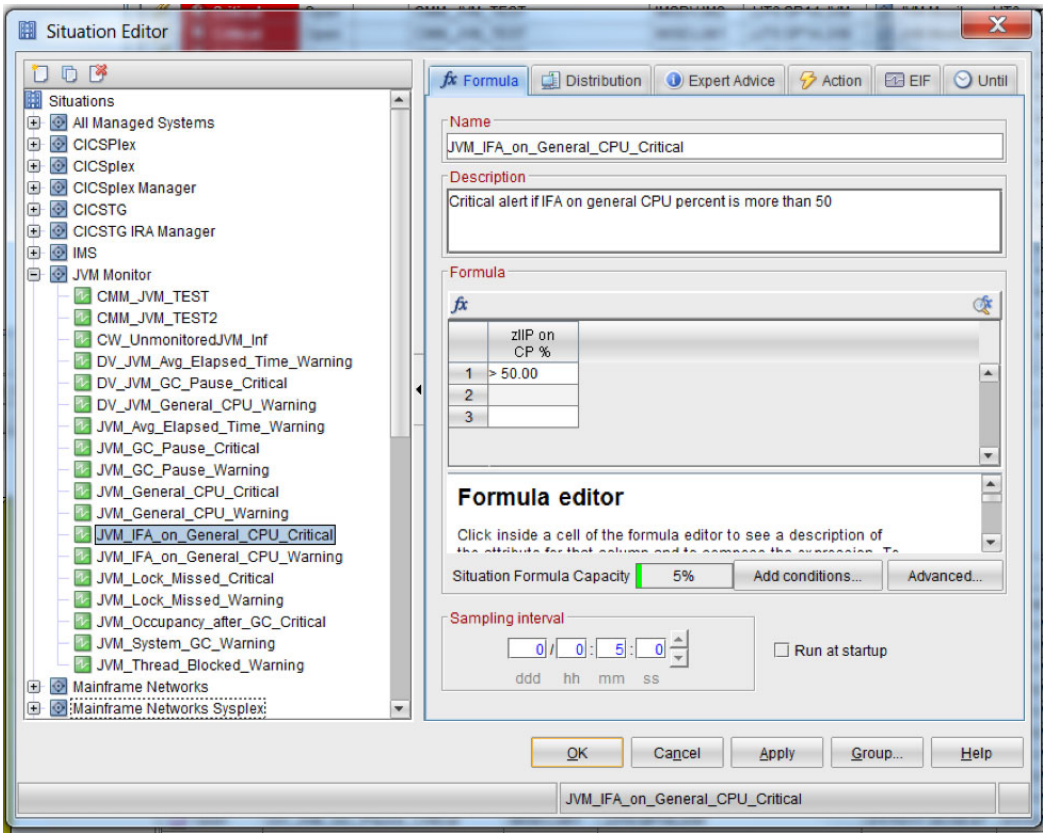


Figure 1-3 Situation Editor in the Tivoli Enterprise Portal

Click a situation name in the JVM Monitor group to select it. The example in Figure 1-3 shows that the product provided situation “JVM_Occupancy_after_GC_Critical.” This situation checks the average heap occupancy after GC every five minutes. High heap occupancy indicates that the Java heap is constrained for memory. This warning could indicate a memory leak, or that the maximum heap size is too small for the workload. If the conditions are met, then the situation is triggered and a “Critical” alert is generated.

After the thresholds and intervals are specified, the situation needs to be distributed to one or more *managed systems* to enable testing for the situation conditions. Select the Distribution tab to activate the situation on individual managed systems, or use a Managed System Group definition. In the context of OMEGAMON for JVM, a managed system represents a collector deployed on an LPAR that one or more JVM instances feed data to.

1.3.3 Historical data

Subject matter experts use OMEGAMON to examine key performance indicators (KPIs) that point to system configuration or application problems. A single real-time snapshot of these KPIs might not be enough to reveal a problem in the making. OMEGAMON collects short-term history of KPIs that can be used to spot trends that might lead to a critical performance issue or outage before they actual happen.

Historical data collection is configured at the attribute group level. A collection requires an interval to be specified to tell the monitoring infrastructure how often to run queries and collect a snapshot of data, for example “every 5 minutes” or “every hour.” As with Situation configuration, you can use either the Enhanced 3270UI or Tivoli Enterprise Portal to define what data you want to collect and store. Figure 1-4 shows an example of configuring historical collection for the “JVM Garbage Collection” attribute group using the Enhanced 3270UI. We specified a collection interval of 5 minutes and distributed the collection to a managed system list that included the LPARs in our sysplex. Generally, it is a good idea to have the same collection interval for all attribute groups you want to collect data for. You might not want to collect and store data every few minutes for some attribute groups because the data is static and will not change.

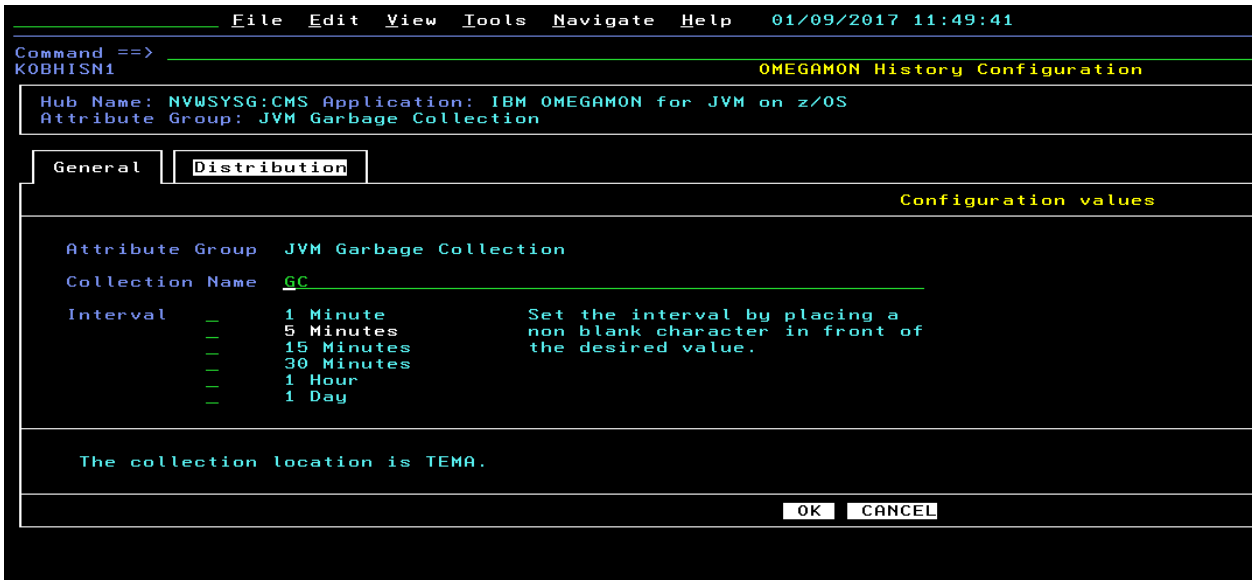


Figure 1-4 Historical Data Collection Editor in the Enhanced 3207UI

Historical data views are available in both the Tivoli Enterprise Portal and the Enhanced 3270UI. When viewing historical workspaces, you can navigate between views for the time snapshot you have chosen to view just like you would for real-time data. In the Enhanced 3270UI, you also easily skip forward and back between snapshots.

The following chapters in this Redpaper demonstrate the value that is provided by the workspaces and situations that use the attributes within these attribute groups.

1.4 Description of application environment

The environment used to test and demonstrate OMEGAMON for JVM was an existing z13 sysplex with multiple logical partitions (LPARs). The product was installed on shared DASD available to two of these LPARs. The application environment is shown in Table 1-1.

Table 1-1 Application environment

LPAR	z/OS version	Software
SYSG	2.02.00	CICS TS 5.3 z/OS Connect EE 2.0 IBM Tivoli Monitoring 6.3 FP6 OMNIMON V730 OMEGAMON for JVM V540
SP14	2.01.00	IMS/DC V14 OMEGAMON for JVM V540

1.4.1 About the applications

A simple test servlet for CICS was written by using Eclipse Luna 4.4.2 with IBM CICS Explorer® 5.3.8. The servlet was deployed as an OSGi application in an enterprise bundle archive (EBA) to CICS JVM LIBSERV1. The servlet makes some JCICS calls, including reading and writing of items in a temporary storage queue and accessing a CICS KSDS VSAM file resource.

z/OS Connect EE V2.0 was configured with APIs defined to represent access to our system by using HTTP verbs. Multiple services were created and defined within the z/OS Connect EE server's server.xml file to access a CICS catalog application by using IBM WebSphere Optimized Local Adaptor (WOLA).

IMS v14 was configured with five Java message processing (JMP) regions running a simple banking application.

2

Installation and configuration

This chapter describes the installation and configuration of OMEGAMON for JVM on z/OS V5.4.0. It does not include the installation and configuration of the IBM Tivoli Monitoring framework, which is covered in other publications.

This chapter covers the following topics:

- Installation
- Monitoring Agent configuration
- Configuring JVMs for monitoring

2.1 Installation

Installation is accomplished with SMP/E. The most convenient installation method is to order internet delivery of the product image from shopZSeries by using FROMNETWORK or FROMNTS. The product FMID is HKJJ540 and the product code is 5968-ABA. The SMP/E product installation files need about 51 MB of DASD space. The front-end support DVD `.iso` image is about 5 MB.

Note: You should also download any PTFs for the product from the Preventive Service Planning (PSP) bucket. As of this writing, two PTFs were recommended: UA83519 and UA83018.

We created the installation jobs by using the OMEGAMON JOBGEN tools. A new zFS file system was created and mounted for the UNIX System Services files required. A number of UNIX files are delivered with this product in the SMP/E file DKANJAR.

In the initial release of the product, it was necessary to download a copy of the Health Center Agent from IBM support. This download is no longer necessary, as OMEGAMON for JVM V5.4.0 packages a copy of the Java Health Center Agent for z/OS (actually there are two: One for 31-bit JVMs and one for 64-bit JVMs). It is important to use the Health Center Agent that is packaged with OMEGAMON for JVM because some older versions of the Health Center Agent that are provided with the IBM J9 JVM have some bugs, or might not be compatible with OMEGAMON for JVM.

2.2 Monitoring Agent configuration

Configuration is completed using the PARMGEN tool, a set of ISPF dialogs and scripts that guides the user through the configuration process. We will not describe every step that is required to run a PARMGEN configuration because that is not the main focus of this publication.

PARMGEN is used to configure all OMEGAMON products and other IBM service management products. For more information about using PARMGEN, see IBM Knowledge Center.

Make sure that you run the configuration jobs with a user ID that has superuser authority. The best practice is to use a user ID with read access to FACILITY class profile SUPERUSR. The user ID must also be authorized to assign extended attributes to zFS files. To allow this authorization, the user ID must have read access to the IBM RACF® FACILITY class profiles BPX.FILEATTR.APF, and BPX.FILEATTR.PROGCTL. Although it is not used by the OMEGAMON for JVM product installation, BPX.FILEATTR.SHARELIB rounds out these related profiles that allow use of the **`extatrr`** UNIX command and the **`pax`** command with **`-ppx`** option.

After the product installation jobs are completed, a Runtime Environment (RTE) should be created, or an existing one augmented to hold the OMEGAMON for JVM configuration. We created new RTEs for this paper on two LPARs in our sysplex, one on LPAR SYSG called REDSYSG, and one on LPAR SP14 called REDSP14 (Figure 2-1).

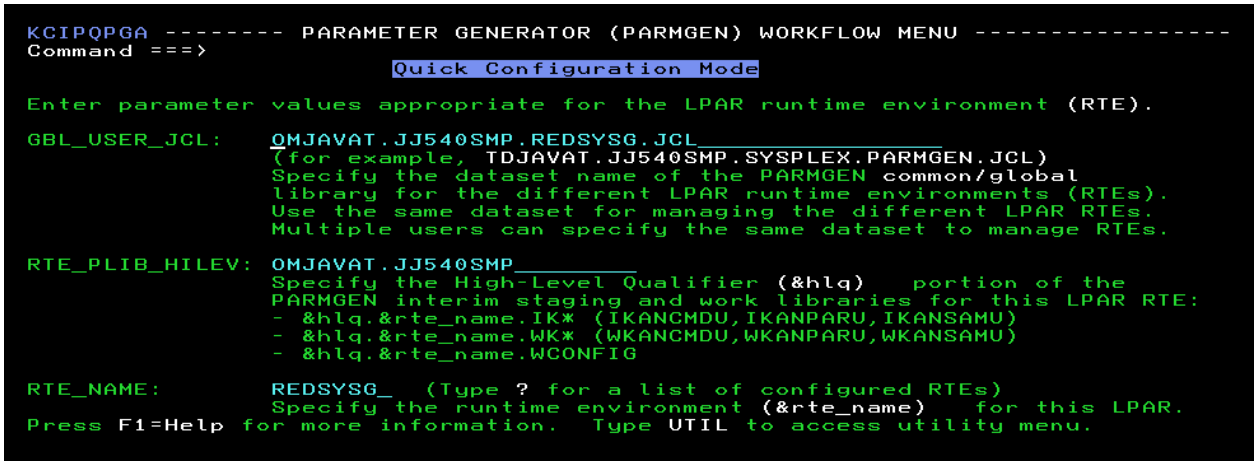


Figure 2-1 Creating an RTE using PARMGEN

This REDSYSG RTE constitutes a self-contained IBM Tivoli Monitoring monitoring environment on LPAR SYSG to support our OMEGAMON for JVM test platform. PARMGEN supplies a number of sample configuration templates that can be used to model RTEs. We chose the model $MDLHSSV “Sharing-w/-SMP RTE w/ Hub/TOM/KJJ Agent w/ variables”.

Configuration of the OMEGAMON Agent and Collector is simple. We accepted the sample defaults for the few configuration variables that are specific to OMEGAMON for JVM, except for the Collector ID and the agent and collector started task names. The LPAR we were using is shared with other IBM users and another instance of the Agent/Collector was already using the default collector ID, which is “KJJ1”. This ID is central to the OMEGAMON for JVM architecture. To allow other instances of the Agent and Collector to run on the same LPAR, a unique four-character *Collector ID* must be specified in the PARMGEN configuration parameter KJJ_COLLECTOR_ID. For this project, we chose a collector ID of “REDB.” You can use the same Collector ID for OMEGAMON for Java agents on other LPARs. We changed the names used for the started tasks IBMJJ and IBMJT to JJD0RBJJ and JJD0RBJT to match the started task naming conventions on this sysplex. JJD0RBJJ is the OMEGAMON Monitoring Agent for JVM, and JJD0RBJT is the OMEGAMON for JVM Collector. Both of these started tasks need to be running in order to collect monitoring data from configured JVMs. The order in which they are started does not matter.

The program load libraries for the product need to be APF-authorized. Because our started task naming convention uses a RACF STARTED profile that assigns a user ID authorized to issue SETPROG APF commands, we uncommented the INCLUDE MEMBER statement in the started task JCL for both procedures to allow dynamic update of the APF list.

Before we configured any JVMs for monitoring, we started the OMEGAMON for JVM Monitoring Agent and Collector from TSO/ISPF SDSF. It does not matter which order they are started (or stopped):

```
S JJD0RBJJ
S JJD0RBJT
```

We created a separate RTE on LPAR SP14 called REDSP14 containing only the OMEGAMON for JVM Monitoring Agent. We used the same Collector ID in REDSP14 as we did in REDSYSG because they are logically part of the same monitoring infrastructure.

Note: As well as verifying the installation, we could also use the product to auto-discover what JVMs are running on the LPARs. After the OMEGAMON Monitoring Agent on an LPAR is up and connected to the Tivoli Enterprise Monitoring Server, log on to the Enhanced 3270 UI, and select one of the LPAR from the JVM Overview workspace. If there are JVMs online within this LPAR, we should see them listed on the “JVM Health Summary” workspace (Figure 2-2). We discuss this scenario in more detail in Chapter 3, “Using OMEGAMON for JVM in operations management” on page 19.

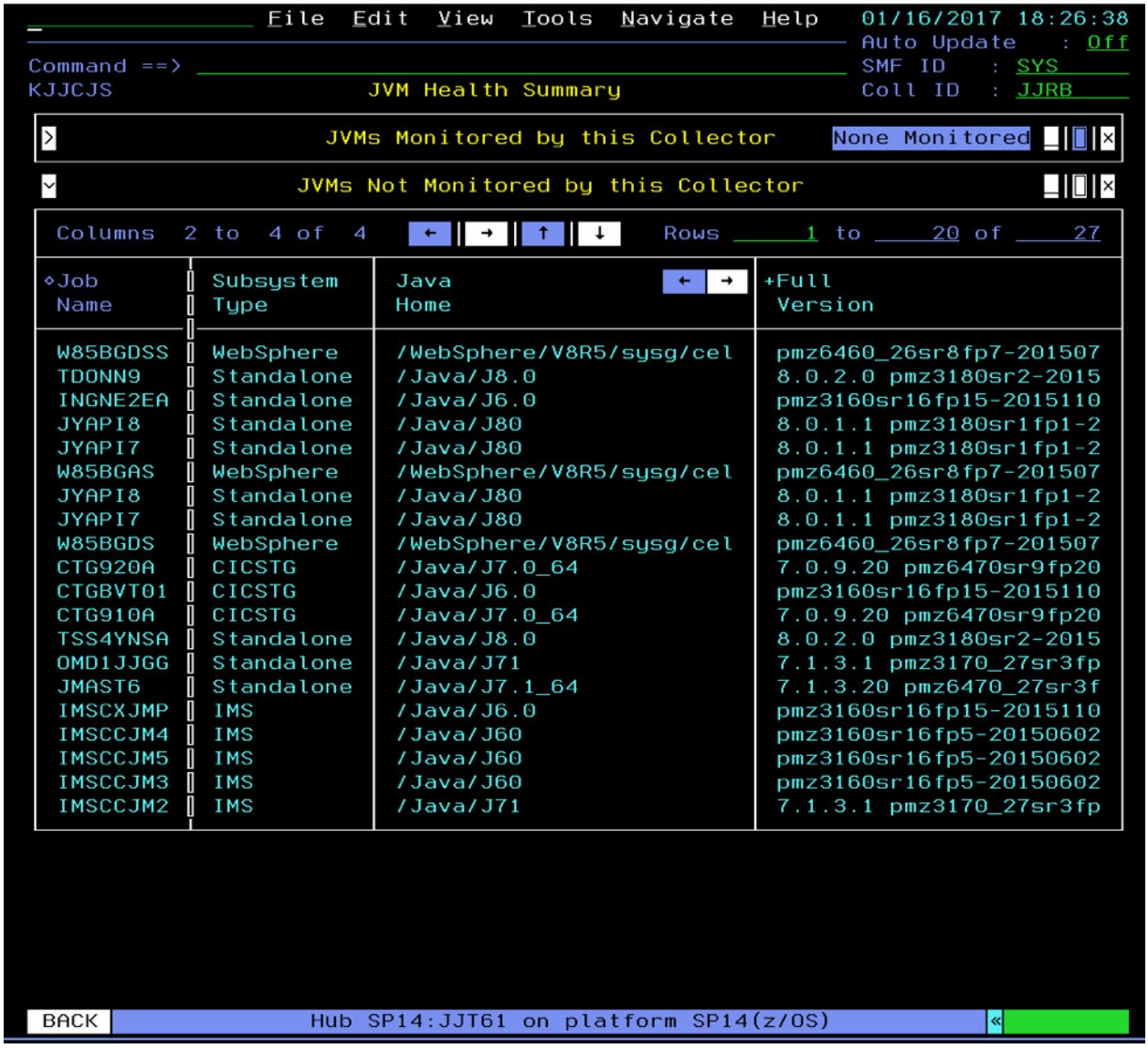


Figure 2-2 JVM Health Summary

2.3 Configuring JVMs for monitoring

Now we needed to configure JVMs for monitoring.

2.3.1 Configuring CICS

We decided to use a CICS Liberty server with a simple Java Platform, Enterprise Edition servlet application. The servlet makes some JCICS calls, including some temporary storage queue calls and reads a CICS file resource. The results are printed to the HTTP output stream and displayed in a browser.

Following the instructions in the PARMGEN post-config document (KJJDFINL), we located the profile configuration file in the zFS file system, and edited it using ISPF 3.17.

The sample profile describes the different sections. Scroll down to the section that contains the JVM configuration options (Figure 2-3).

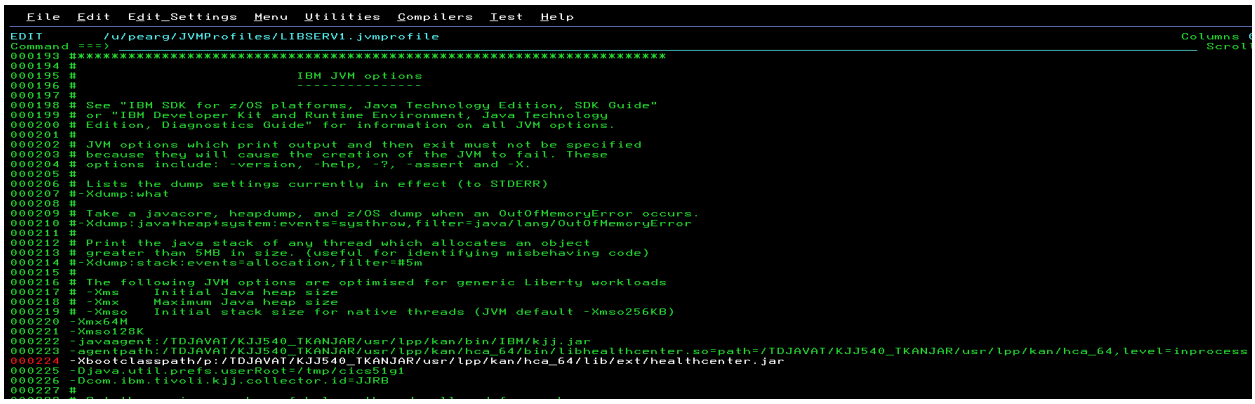


Figure 2-3 Configuration of a CICS JVMSERVER for monitoring by OMEGAMON

Because we were using an SMP/E sharing RTE, we used the Health Center Agent and the OMEGAMON Java Agent from the SMP/E target zFS directory. These are the options that we added:

```
-javaagent:/TDJAVAT/KJJ540_TKANJAR/usr/lpp/kan/bin/IBM/kjj.jar
```

This option specifies the path to the OMEGAMON for JVM Java Agent:

```
-agentpath:/TDJAVAT/KJJ540_TKANJAR/usr/lpp/kan/hca_64/bin/libheal
thcenter.so=path=/TDJAVAT/KJJ540_TKANJAR/usr/lpp/kan/hca_64,level=inprocess,disabl
eCH
```

The **-agentpath** option specifies the full path to a JVMTI agent (in this case, the HC agent provided in the OMEGAMON for JVM packaging). CICS Liberty uses a 64-bit JVM so we specified the path to the 64-bit HC Agent. Text following the first = sign is passed to the HC agent as parameters. The **path=** parameter specifies the path to the agent and must be the same as path prefix for the agent. The path prefix is the installation root for the Health Center Agent, which is the directory above the one containing `libhealthcenter.so`. The **level=inprocess** parameter tells the HC agent that only an internal connection will be used, and a TCP socket port will not be opened. **disableCH** specifies that Class Histogram data will not be collected. Because OMEGAMON does not use this data, disabling its collection provides a slight performance optimization.

```
-Xbootclasspath/p:/TDJAVAT/KJJ540_TKANJAR/usr/lpp/kan/hca_64/lib
/ext/healthcenter.jar
```

The **-Xbootclasspath** option adds the HC Agent JAR file to the JVM boot class path.

These options are normally sufficient for most installations. However, because our JVM is running on an LPAR with multiple online instances of OMEGAMON for JVM, we could not use the default collector ID (KJJ1) because it was already being used. When we set up the OMEGAMON monitoring agent in PARMGEN, we changed the value of KJJ_COLLECTOR_ID from “KJJ1” to “KJRB.” In order to connect to that Collector, our JVM must be configured with an additional system property:

```
-Dcom.ibm.tivoli.kjj.collector.id=JJRB
```

We saved our changes to the profile, and restarted the LIBSERV1 JVM in CICS by using the CEMT transaction to disable and then enable the LIBSERV1 JVM.

Returning to the e3270UI, we could now see that the CICS51G1 JVM is being monitored (Figure 2-4).

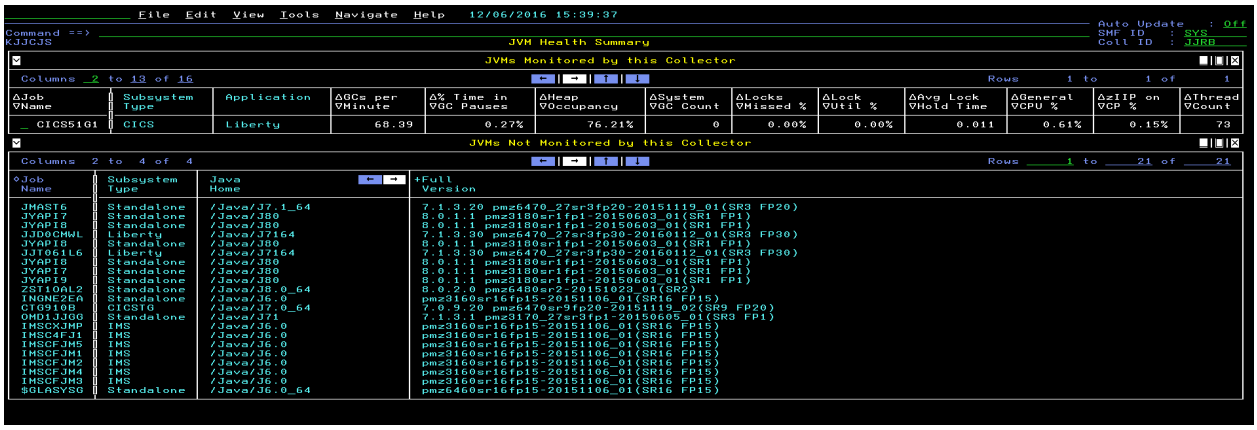


Figure 2-4 JVM Health Summary workspace in OMEGAMON Enhanced 3270 UI indicating which JVMs are being monitored, and which are not.

2.3.2 Configuring IMS

We also configured an IMS system Java message processing (JMP) region for monitoring. In IMS, the JCL for a JMP is stored in the IMS procedure library (IMS PROCLIB). The IMS JMP also uses members of the IMS PROCLIB for the JVM configuration. The JVM options member is specified with the JVMOPMAS= parameter in the JMP started task procedure JCL. We located the member used to configure our JMP and added the following options:

```
-Xbootclasspath/p:/TDJAVAT/KJJ540_TKANJAR/usr/lpp/kan/hca/lib/ext/healthcenter.jar
-agentpath:/TDJAVAT/KJJ540_TKANJAR/usr/lpp/kan/hca/bin/libhealthcenter.so=path=/TD
JAVAT/KJJ540_TKANJAR/usr/lpp/kan/hca,>
level=inprocess,disableCH
-javaagent:/rtehome/REDSP14/kan/bin/IBM/kjj.jar
-Dcom.ibm.tivoli.kjj.collector.id=JJRB
```

IMS has a slightly different syntax for specifying JVM options than other subsystems. Because the IMS PROCLIB is an IBM MVS™ data set with a logical record length of 80 bytes, parameters longer than 72 bytes must be split over multiple lines. The “>” (greater than) character is used as the continuation character, and the parameter continues in column 1 of the next line.

2.3.3 Configuring z/OS Connect Enterprise Edition

Finally, we configured an instance of z/OS Connect Enterprise Edition (EE) V2.0 to create JSON-based RESTful APIs into an existing CICS application. It runs in a Liberty Profile either stand-alone or in a CICS JVM Liberty server. To configure the z/OS Connect JVM for monitoring with OMEGAMON, follow the instructions for monitoring a Liberty profile. If using z/OS Connect in CICS, follow the instructions in 2.3.1, “Configuring CICS” on page 15.

The configuration of a stand-alone Liberty profile can be accomplished by adding the enablement options to the JVM_OPTIONS environment variable. A stand-alone Liberty server is usually run under the UNIX batch utility BPXBATSL, so environment variables are supplied in the file allocated to the STDENV DD name. In our environment, STDENV was a zFS file. We added the following options to the JVM_OPTIONS environment variable:

```
JVM_OPTIONS=-javaagent:/TDJAVAT/KJJ540_TKANJAR/usr/lpp/kan/b in/IBM/kjj.jar
-agentpath:/TDJAVAT/KJJ540_TKANJAR/usr/lpp/kan/h
ca_64/bin/libhealthcenter.so=path=/TDJAVAT/KJJ540_TKANJAR/usr/lpp/kan/hca_64,level
=inprocess,disableCH -Xbootclasspath/p:/TDJ
AVAT/KJJ540_TKANJAR/usr/lpp/kan/hca_64/lib/ext/healthcenter.jar:/TDJAVAT/
KJJ540_TKANJAR/usr/lpp/kan/bin/IBM/kjjboot.jar -Dcom.i
bm.tivoli.kjj.collector.id=JJRB
```

Note: Notice the addition of the file path to **`kjjboot.jar`** on the **`-Xbootclasspath`** option. This addition is required if you intend, as we did, to monitor service request times from your z/OS Connect EE instance with OMEGAMON for JVM.

Request monitoring requires the configuration of a z/OS Connect Interceptor supplied in the OMEGAMON for JVM zFS directory. We followed the instructions in IBM Knowledge Center to add the interceptor to the environment.

3

Using OMEGAMON for JVM in operations management

This chapter explores how OMEGAMON for JVM can be used to identify problems in JVMs when used in an operations management context. The scenarios that are covered here broadly replicate those that might be seen by an enterprise when adopting Java workloads, for example, creating a CICS application deployed within a Liberty server, or a JMS region within an IMSplex. Finally, we demonstrate the monitoring of a z/OS Connect EE instance deployed to expose the CICS applications through APIs. In each case, we use OMEGAMON for JVM to be alerted to problems before they can cause outages and identify the next steps in root cause analysis quickly.

We use the Tivoli Enterprise Portal and OMEGAMON Enhanced 3270UI (e3270UI) interchangeably throughout the scenarios to demonstrate that functionality is available in either user interface.

This chapter covers the following topics:

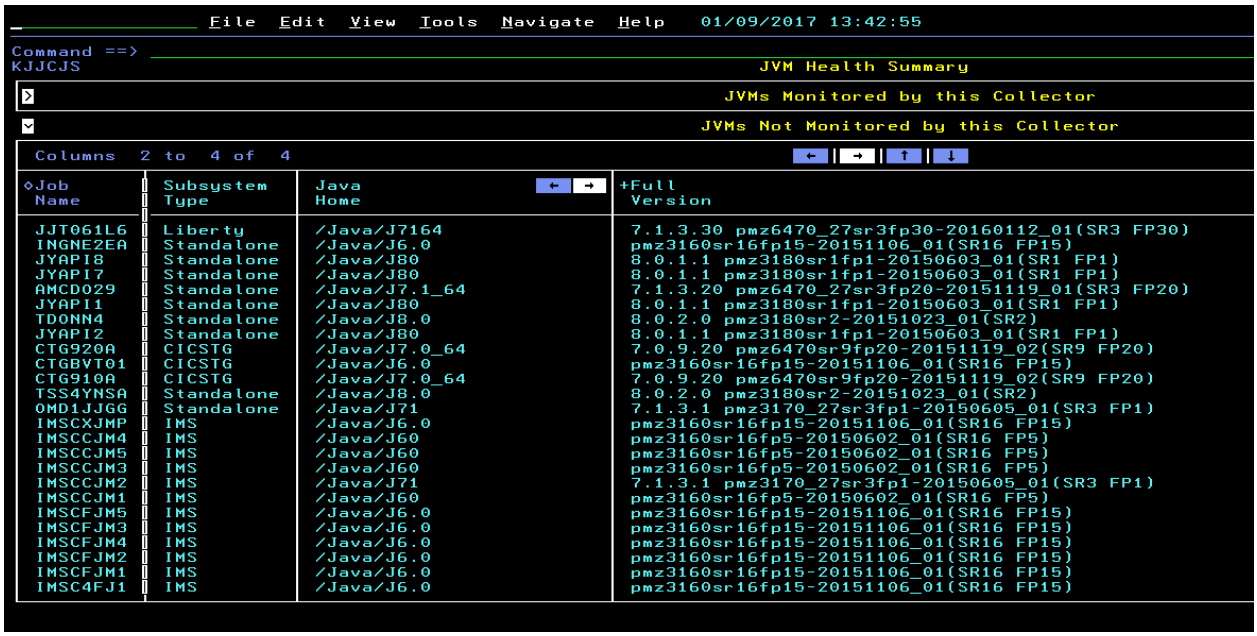
- JVM discovery and JVM environments
- High heap occupancy in CICS region
- Monitoring extended storage in IMS
- Using OMEGAMON history to identify memory leaks
- Monitoring z/OS Connect Service Request Times

3.1 JVM discovery and JVM environments

One of the key questions that an operator or systems administrator needs to understand when Java workloads are introduced to their environment is exactly how much Java is being run. We could refer to this as “dark Java,” or resources you are not aware are being consumed. With Java found in CICS, IMS, WebSphere, and countless other applications, a clear understanding of how much is Java is deployed helps resolve initial concerns on knowing what to monitor.

In the previous chapter, while configuring our scenario environments, we knew which subsystems we would be using and so enabled them for monitoring. However there might be other subsystems and JVMs online that we are not aware of. For example, a CICS programmer might enable a JVM on a region to try a new application but did not notify the operations team. It would be good to be alerted when this happens so we can be certain we have all known resources covered.

OMEGAMON for JVM has an auto-discovery function that lists all online JVMs within the LPAR the agent is deployed on. Figure 3-1 show an example where the top workspace indicates the ‘known’ JVMs we have configured for full monitoring and the lower panel indicates the JVMs discovered but are not configured for monitoring.



File Edit View Tools Navigate Help 01/09/2017 13:42:55

Command ==>
KJJCJS

JVM Health Summary

JVMs Monitored by this Collector

JVMs Not Monitored by this Collector

Columns 2 to 4 of 4

◇Job Name	Subsystem Type	Java Home	+Full Version
JJT061L6	Liberty	/Java/J7164	7.1.3.30 pmz6470_27sr3fp30-20160112_01(SR3 FP30)
INGNE2EA	Standalone	/Java/J6.0	pmz3160sr16fp15-20151106_01(SR16 FP15)
JYAPI8	Standalone	/Java/J80	8.0.1.1 pmz3180sr1fp1-20150603_01(SR1 FP1)
JYAPI7	Standalone	/Java/J80	8.0.1.1 pmz3180sr1fp1-20150603_01(SR1 FP1)
AMCDO29	Standalone	/Java/J7.1_64	7.1.3.20 pmz6470_27sr3fp20-20151119_01(SR3 FP20)
JYAPI1	Standalone	/Java/J80	8.0.1.1 pmz3180sr1fp1-20150603_01(SR1 FP1)
TDONN4	Standalone	/Java/J8.0	8.0.2.0 pmz3180sr2-20151023_01(SR2)
JYAPI2	Standalone	/Java/J80	8.0.1.1 pmz3180sr1fp1-20150603_01(SR1 FP1)
CTG920A	CICSTG	/Java/J7.0_64	7.0.9.20 pmz6470sr9fp20-20151119_02(SR9 FP20)
CTGBVT01	CICSTG	/Java/J6.0	pmz3160sr16fp15-20151106_01(SR16 FP15)
CTG910A	CICSTG	/Java/J7.0_64	7.0.9.20 pmz6470sr9fp20-20151119_02(SR9 FP20)
TSS4YNSA	Standalone	/Java/J8.0	8.0.2.0 pmz3180sr2-20151023_01(SR2)
OMD1JJGG	Standalone	/Java/J71	7.1.3.1 pmz3170_27sr3fp1-20150605_01(SR3 FP1)
IMSCXJMP	IMS	/Java/J6.0	pmz3160sr16fp15-20151106_01(SR16 FP15)
IMSCCJM4	IMS	/Java/J60	pmz3160sr16fp5-20150602_01(SR16 FP5)
IMSCCJM5	IMS	/Java/J60	pmz3160sr16fp5-20150602_01(SR16 FP5)
IMSCCJM3	IMS	/Java/J60	pmz3160sr16fp5-20150602_01(SR16 FP5)
IMSCCJM2	IMS	/Java/J71	7.1.3.1 pmz3170_27sr3fp1-20150605_01(SR3 FP1)
IMSCCJM1	IMS	/Java/J60	pmz3160sr16fp5-20150602_01(SR16 FP5)
IMSCFJM5	IMS	/Java/J6.0	pmz3160sr16fp15-20151106_01(SR16 FP15)
IMSCFJM3	IMS	/Java/J6.0	pmz3160sr16fp15-20151106_01(SR16 FP15)
IMSCFJM4	IMS	/Java/J6.0	pmz3160sr16fp15-20151106_01(SR16 FP15)
IMSCFJM2	IMS	/Java/J6.0	pmz3160sr16fp15-20151106_01(SR16 FP15)
IMSCFJM1	IMS	/Java/J6.0	pmz3160sr16fp15-20151106_01(SR16 FP15)
IMSC4FJ1	IMS	/Java/J6.0	pmz3160sr16fp15-20151106_01(SR16 FP15)

Figure 3-1 JVM Health Summary workspace showing unmonitored JVMs auto-discovered on this LPAR

Reviewing this subpanel, we can see the address space details and subsystem type. From there we can decide to configure that address space for full JVM monitoring, ignore it, or investigate further as to why this JVM is online.

To avoid viewing this window consistently to see whether anything changed, a simple low severity situation can be created off the “JVM Auto Discovery” attribute group where the Monitored attribute is set to N. After the non-configured JVM comes online, the alert will be triggered. See Figure 3-2.

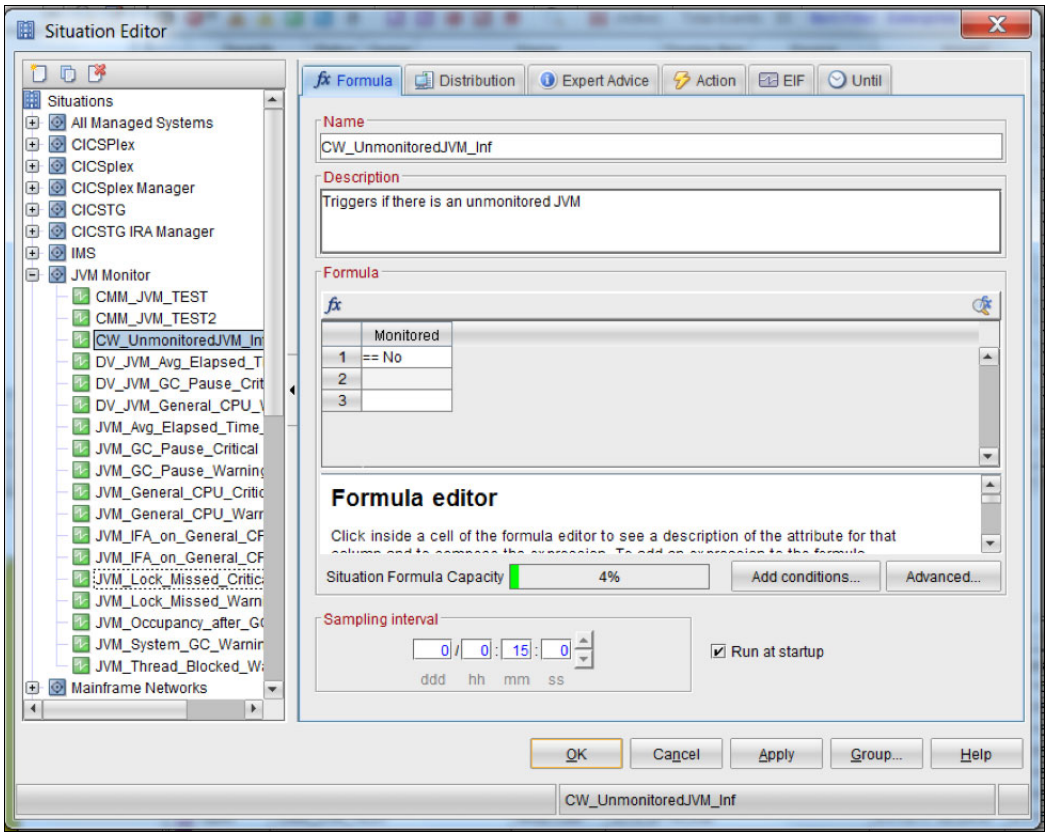


Figure 3-2 Situation Editor

Note: If you have multiple OMEGAMON for JVM agents/collector pairs deployed on the same LPAR, JVMs configured to be monitored by one agent/collector pair appear as unmonitored by all other agent/collector pairs.

Another example of using situations to quickly manage online JVM properties is to test the environment settings as provided by the JVM Environment Data and JVM Environment Data Details attribute groups. These groups contains details about the environment and system variables used by the JVM plus class path information.

We considered a scenario where we wanted to migrate all online JVMs off an older level of Java (for example, Java 6) before uninstalling it. Instead of laboriously checking each JVM in turn (there could be dozens), we created a situation that checks the version of every online JVM. Alerts are generated immediately on the address spaces affected, meaning that we can contact the application owners to start the migration planning process.

More detailed checks on individual build levels can also be performed by creating a situation off the Java full version instead.

3.2 High heap occupancy in CICS region

In this scenario, we have developed a Java application to be deployed onto a new Liberty JVM we created within a CICS region. While testing the new application, we set the JVM with a maximum heap size of 64 MB. This server has the Java Platform, Enterprise Edition feature enabled and our new application, a servlet called “Parts,” was deployed in an OSGi enterprise bundle archive.

A simulated workload of 10 concurrent users exercising the Parts servlet was started. No situations were triggered. When we increased the workload users to 15, after a short time the Tivoli Enterprise Portal situation console showed a new situation had been triggered. Refer to Figure 3-3 and Figure 3-4.

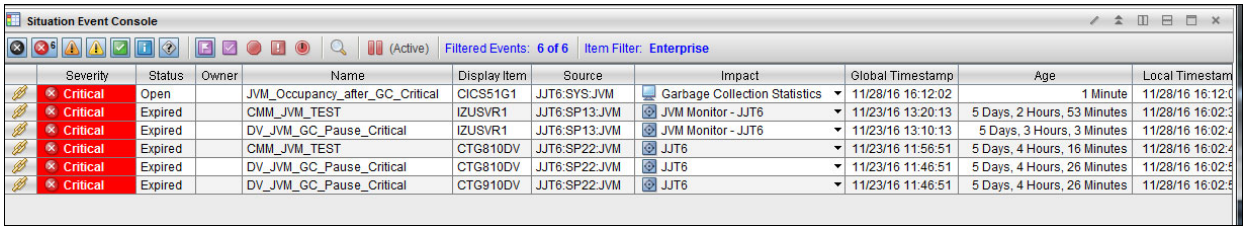


Figure 3-3 Situation Event Console in the Tivoli Enterprise Portal

Figure 3-4 shows the data for High Heap Occupancy.

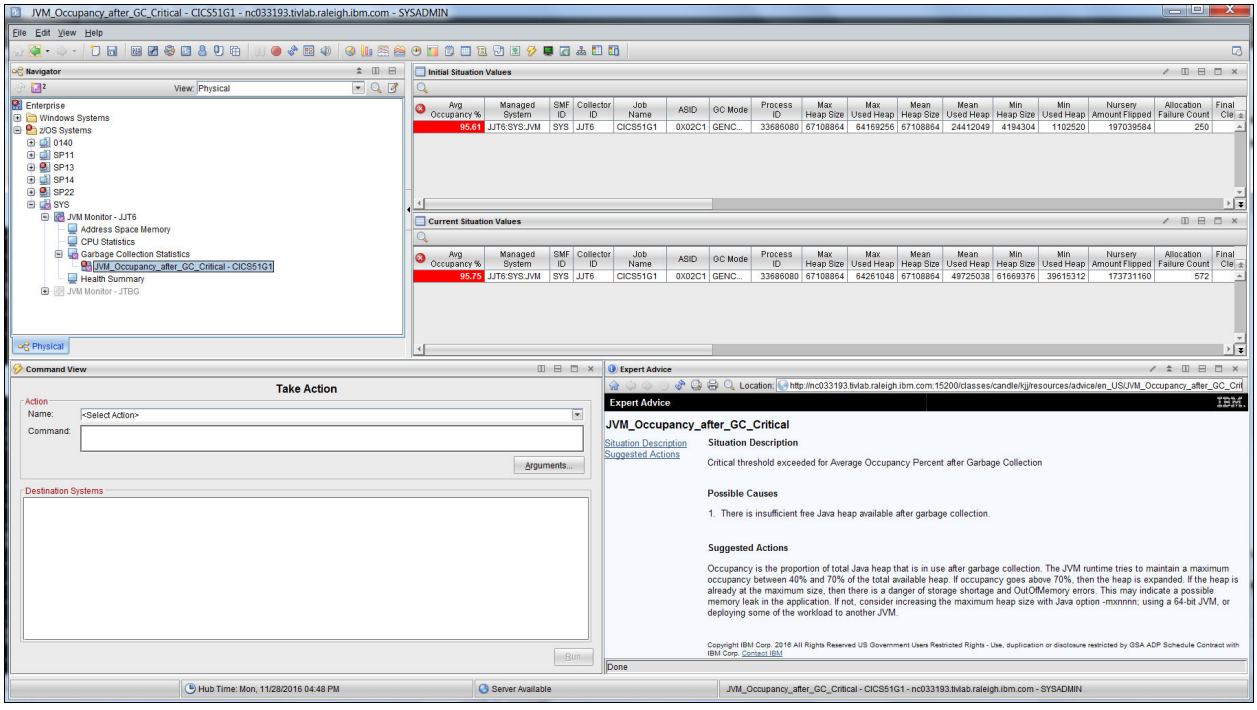


Figure 3-4 Situation Event Data for High Heap Occupancy in the Tivoli Enterprise Portal

Similar information is available in the enhanced 3270 UI (Figure 3-5).

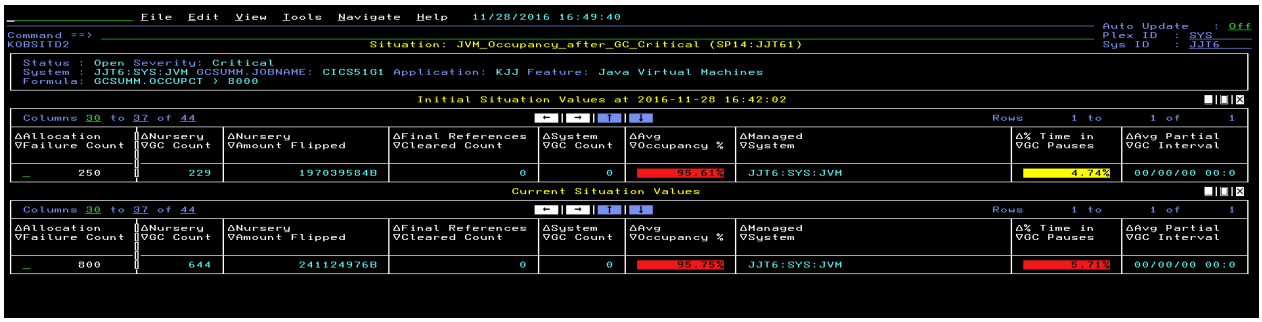


Figure 3-5 Situation Event Data in the Enhanced 3270 UI

The situation event data includes *expert advice* for what might be causing the issue. Heap occupancy after GC is a measure of how much of the total available JVM heap memory is committed to active tasks. By default, the JVM triggers a garbage collection when the heap in use reaches 80% of the current heap size.

The garbage collection process frees any storage that does not have any references to it. If after the garbage collection has completed, there is still more than 80% of the heap in use, and the heap cannot be expanded further (because it is at the maximum allowable heap size specified by the **-Xmx** JVM option), then garbage collection thrashes the JVM.

You can see in the situation data in Figure 3-5, that the percentage of time spent in garbage collection was 4.74% when the situation was triggered. At 95% heap occupancy, Java stops dispatching new threads, and many active threads will suffer OutOfMemoryError exceptions. The JVM will also be consuming more CPU than it normally does because of the additional garbage collection going on.

Operators at the event console can then acknowledge the situation event and route it to a subject matter expert (SME) such as a CICS system programmer.

3.3 Monitoring extended storage in IMS

After we configured IMS for monitoring, OMEGAMON began warning us about low extended region free memory when the JMP region had been running for about 12 hours. The Native Memory Summary workspace in the e3270UI (KJJNMEM) highlights the extended region free percentage if it goes below 10%. We were using IMS V14 without PTF PI64142, which meant we could not use a 64-bit JVM. If extended free storage went much lower, we would start to see OutOfMemoryErrors or abends that would crash the JMP.

Because we had history enabled, we were able to see the free storage percentage slowly diminish over this time. Where was this storage going? Was it a memory leak in an application? A JVM component? Or was it IBM Language Environment® managed storage or memory used for z/OS services?

The historical data showed no substantial increase in any JVM native memory category, nor in z/OS managed virtual storage used for LSQA, SWA, and subpools 229 and 230. However, it did reveal a continuous increase in Language Environment managed storage. The Language Environment heap size grew constantly throughout this period even though the Language Environment Heap Allocated grew only slightly. Most of the storage was Language Environment free heap memory.

We consulted with IMS L2 support and they suggested running the JVM with LE RPTSTG(ON) to create a report of memory allocation. Setting Language Environment runtime options can be accomplished in a couple of ways, such as setting the environment variable _CEE_RUNOPTS, but we elected to add a CEEOPTS DD to the JMP started task JCL. This DD statement specified a member of the IMS.PROCLIB in which we could place the Language Environment options.

In addition to RPTSTG(ON), we added RPTOPTS(ON), which would print a report of the Language Environment options effective for the process. Both reports would be printed to DD name CEEDUMP, which we allocated to a JES SYSOUT queue.

When the IMS JMP was stopped after several hours, we found a very large number of fragmented free storage areas of numerous sizes. There were no memory leaks of in-use memory.

Language Environment normally reuses free storage segments for memory requests of the same size. If none are available, Language Environment allocates segments of the correct size, even if there are free segments that are larger than the requested amount. The result is that the address space is filled with Language Environment free memory segments. At some point, this process can reach a dynamic equilibrium and grow no further, but because we needed quite a large Java heap for our application, we could not afford to waste memory.

The solution allows Language Environment to return free segments to the operating system allowing them to be consolidated. This process is done by specifying HEAP(,,,,FREE) in the Language Environment runtime options. We added this option to the CEEOPTS file allocated to our JMP, and restarted the region.

The result was no further increase in extended storage utilization. Even after 48 hours, the free extended memory percentage remained at 72%. All Java dependent regions should include the HEAP(,,,,FREE) LE option because the default IMS Language Environment runtime options do not include this setting by default.

The product-provided workspaces for native memory do not include any summary history views that show the progress of memory usage over time in one workspace. However, it is really easy to create your own enhanced 3270UI workspaces to create custom reports.

Start by examining the source for a workspace that contains data that you want to include in your custom report. You can use the **View** → **Workspace Source** menu items (or fast path “v.s” on the menu select field or command line).

We wanted to display Language Environment heap and region free percent values from history. The KJJNMEM workspace queries the KJJ.ADDRMEM table, so we created a new member in the library allocated to DD name UKANWENU, which is for a custom panel containing what is shown in Example 3-1.

Example 3-1 Enhanced 3270 UI workspace definition

```
<WORKSPACE>
HEADER='Region History'
NAV1TEXT='SMF ID  '
NAV2TEXT='Coll ID '
SET ZOMEGLOCK1=YES
SET ZOMEGLOCK2=YES
IMBED=KJJNAVI
HISTORY=AGENT
/* ********************************** */
/* *  Subpanel - Request Summary       * */
```

```
/* ************************************* */

<SUBPANEL>
HEADER='Jobname: &JOBNAME JVM Pid: &JVMPID'
NAME=QRY_ONE
TYPE=SUMMARY
WHENNOTEXT="No requests"
/* ******************** */
/* *  Data Query      * */
/* ******************** */

QUERYTYPE=ROUTER
QUERYMODE=LIVE
QUERYREGTYPE=DRA
QUERYWHEN=RETURN

QUERY='SELECT WRITETIME,
ORIGINNODE,
SMFID,
COLLID,
JOBNAME,
ASID,
JVMPID,
HEAPSIZE,
HEAPALLOC,
HEAPFREE,
LDAALLOC,
LDASIZE,
LDARSRVD,
LDAFREPCT,
LDAALCPCT
FROM KJJ.ADDRMEM HISTORY()
WHERE ORIGINNODE = "&KJJONODE"
AND JOBNAME = "&JOBNAME"
AND JVMPID = &JVMPID'

DISPLAYCOLS='WRITETIME(DATETIME),
HEAPSIZE(WIDTH=8),
HEAPALLOC(WIDTH=8),
HEAPFREE(WIDTH=8),
LDAFREPCT(WIDTH=8,CAPTION="Region\Free")'

STATICCOLS=1
<WORKSPACEEND>
```

The important thing to note is that the query specifies the HISTORY() modifier on the FROM clause so that the data is retrieved from the product history repositories.

We called this new workspace KJJRGNH, and we invoked it from the KJJNMEM panel by using the shortcut “=KJJRGNH” on the command line. It is important to invoke it from the KJJNMEM panel because our custom workspace uses variables that are set in the KJJNMEM workspace.

The resulting workspace is shown in Figure 3-6.

File Edit View Tools Navigate Help 12/12/2016 16:54:36

Command ==>
KJJRGNH

Region History

Jobname: IMSEUJM5 JVM Pid: 17236720

Columns 2 to 5 of 5

◊Recording Time	Heap Size	Heap Allocated	Heap Free	Extended Region Free %
12/12/16 12:40:00	408690928	21908432	386782496	48%
12/12/16 12:45:00	408690928	21910096	386780832	47%
12/12/16 12:50:00	425501488	21921312	403580176	47%
12/12/16 12:55:00	429044728	21930664	407114064	45%
12/12/16 13:00:00	451226016	21941392	429284624	45%
12/12/16 13:05:00	463551968	21951968	441600000	44%
12/12/16 13:10:00	465002080	21953472	443048608	43%
12/12/16 13:15:00	482600224	21964688	460635536	43%
12/12/16 13:20:00	482600224	21968176	460632048	41%
12/12/16 13:25:00	510034392	21978552	488055840	41%
12/12/16 13:30:00	514532376	21988944	492543432	40%
12/12/16 13:35:00	525027672	21991120	503036552	40%
12/12/16 13:40:00	540185112	22001360	518183752	39%
12/12/16 13:45:00	541700856	22003920	519696936	38%
12/12/16 13:50:00	569418984	22023104	547395880	37%
12/12/16 13:55:00	569418984	22025632	547393352	36%
12/12/16 14:00:00	586633544	22036072	564597472	35%
12/12/16 14:05:00	612947736	22132832	590814904	33%
12/12/16 14:10:00	619404504	22134736	597269768	33%
12/12/16 14:15:00	637341016	22145776	615195240	32%
12/12/16 14:20:00	637341016	22147216	615193800	32%
12/12/16 14:25:00	657928616	22158400	635770216	31%
12/12/16 14:30:00	660423728	22168672	638255056	29%
12/12/16 14:35:00	677057808	22170752	654887056	29%
12/12/16 14:40:00	697215696	22181304	675034392	28%
12/12/16 14:45:00	702255168	22183512	680071656	28%
12/12/16 14:50:00	728546640	22193952	706352688	26%
12/12/16 14:55:00	731939088	22195784	709743304	25%
12/12/16 15:00:00	749921808	22206192	727715616	25%
12/12/16 15:05:00	754244408	22216088	732028320	24%
12/12/16 15:10:00	755108928	22217560	732891368	24%
12/12/16 15:15:00	770818032	22228208	748589824	23%
12/12/16 15:20:00	772563488	22229664	750333824	23%
12/12/16 15:25:00	791943904	22239816	769704088	21%
12/12/16 15:30:00	822174528	22254352	799920176	19%
12/12/16 15:35:00	855375960	22264984	833110976	17%
12/12/16 15:40:00	868959120	22275200	846683920	16%
12/12/16 15:45:00	877109016	22277264	854831752	16%
12/12/16 15:50:00	898125128	22287400	875837728	14%
12/12/16 15:55:00	898125128	22289176	875835952	14%
12/12/16 16:00:00	920251976	22307664	897944312	13%
12/12/16 16:05:00	923972584	22318136	901654448	13%
12/12/16 16:10:00	924902736	22319320	902583416	13%
12/12/16 16:15:00	941793216	22329632	919463584	11%
12/12/16 16:20:00	954930256	22331808	932598448	11%
12/12/16 16:25:00	975754576	22342384	953412192	9%
12/12/16 16:30:00	975754576	22343400	953411176	9%
12/12/16 16:35:00	996759472	22353840	974405632	8%

Figure 3-6 JVM region history for selected IMS address space

3.4 Using OMEGAMON history to identify memory leaks

Memory leaks in applications can be hard to detect. Often the first symptom would be an unexpected abend or “Out Of Memory” error in the address space that then might require extensive memory dump or log analysis before determining the root cause.

With the Java memory management model using a garbage collector process, developers often believe that memory leaks are difficult to create with Java. However, it is entirely possible for the Java heap to be slowly exhausted by a poorly written application. When developing new applications in Java, being able to identify any memory leaks early in testing increases confidence that a production-deployed version will perform to required capabilities.

In addition to following the real-time KPIs for garbage collection frequency, JVM pause time and heap occupancy (the amount of free space in the Java heap after a garbage collection has been performed) we used the historical data collection function to analyze the trends in JVM behavior and trap any memory leak.

First, we enabled historical data collection for several attribute groups including “JVM Garbage Collection” with a 5-minute collection interval and distributed this to all our managed systems. Figure 3-7 shows the collections that we enabled.

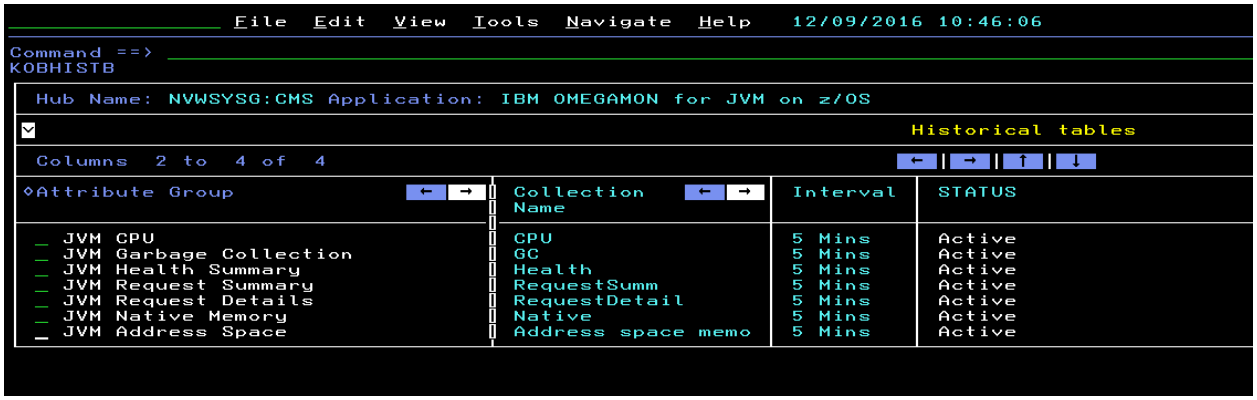


Figure 3-7 Attribute groups configured for historical data collection

After running our test application for some time, the Tivoli Enterprise Portal gave us a warning about high average heap occupancy in one of our monitored JVM. Was this another example of an acute shortage of Java heap because of unconstrained workload or was it a chronic problem due to a memory leak?

History views are available in both the Tivoli Enterprise Portal and the Enhanced 3270UI. However, the Tivoli Enterprise Portal GUI provides real-time situation awareness and also provides compelling charts and graphs that highlight a problem such as a memory leak more visually as shown in Figure 3-8.

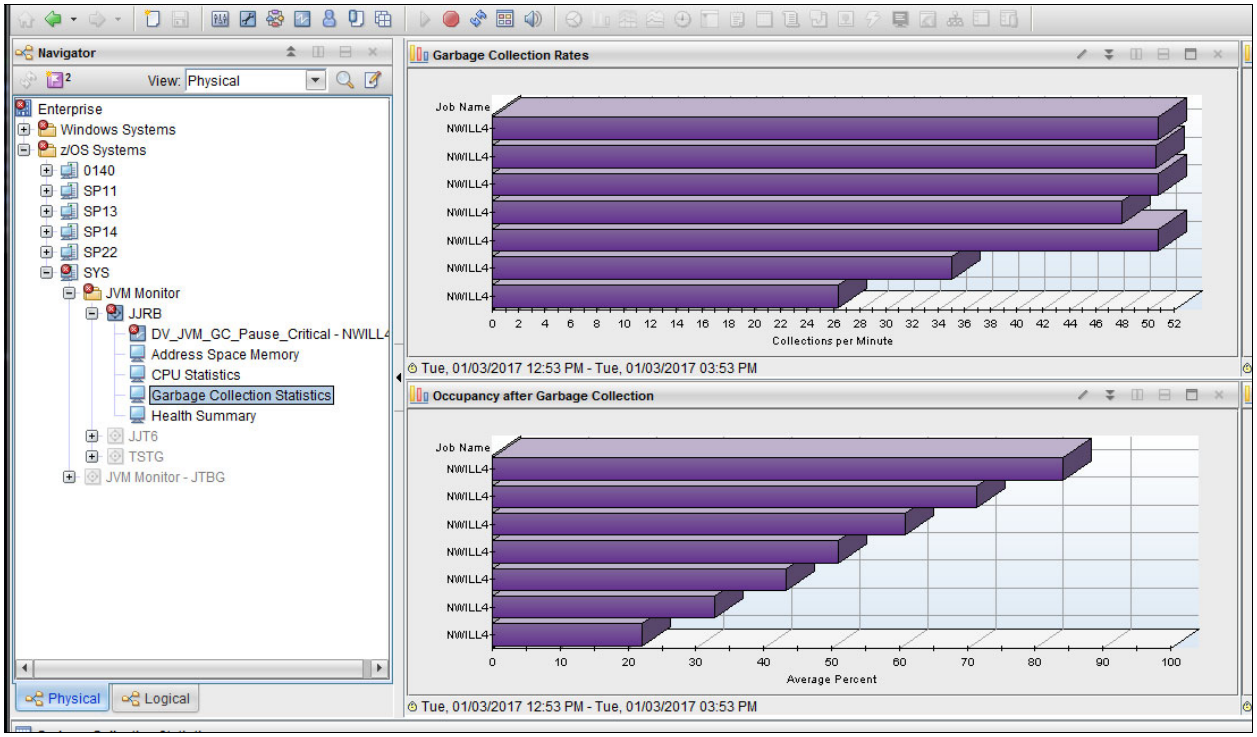


Figure 3-8 Historical Garbage Collection Statistics Tivoli Enterprise Portal workspace indicating the growth of JVM heap occupancy over time due to memory leak

A quick view of Garbage Collection history sealed the analysis. The heap had been steadily growing since the application was started and the number of garbage collections being performed also increased as the JVM attempted to free heap memory. Clearly, this was a memory leak.

By seeing this information quickly, we as operators can take prompt action. First, we can bring up new applications or application servers before the current problem causes the out of memory error. Second, we have identified already the root cause and can capture key information that can be assigned to the appropriate application developers for resolution.

3.5 Monitoring z/OS Connect Service Request Times

Recognizing the importance of APIs in the role of application modernization, OMEGAMON for JVM on z/OS V5.4.0 added a feature specifically to monitor service request times. z/OS Connect EE defines individual RESTful API calls as services. Each service has a URL associated with it that clients use to make API calls. z/OS Connect then transforms these JSON API requests into the native application calls that CICS, or IMS or DB2 understand, and transforms the results back into a JSON stream that the mobile or cloud application understands (Figure 3-9).

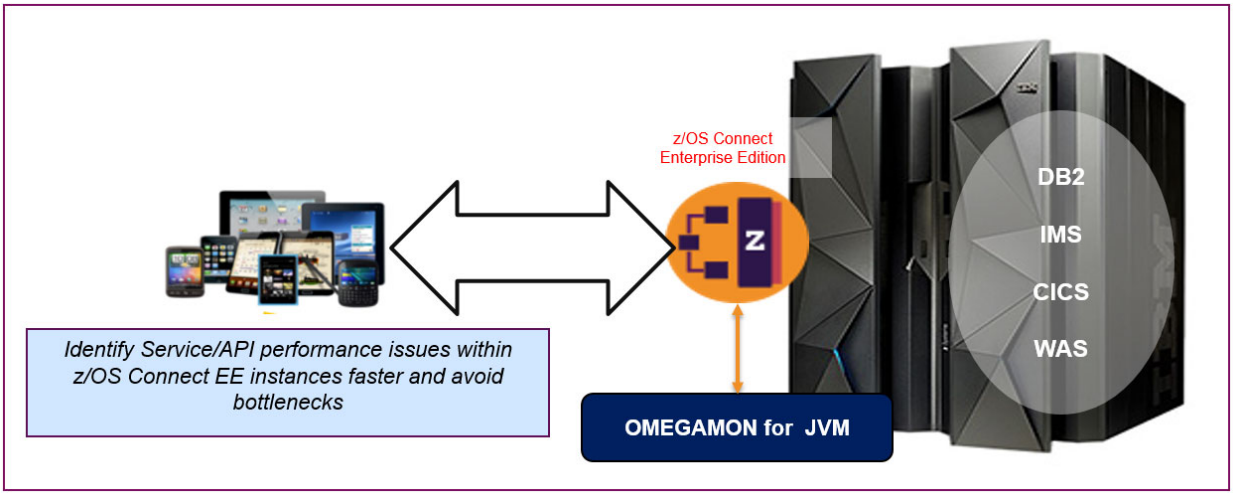


Figure 3-9 z/OS Connect EE and OMEGAMON Topology

OMEGAMON for JVM examines every client interaction with z/OS Connect, and accumulates a summary of the worst performing (longest response time) services. Generally speaking, transactions that run quickly with fast response and minimal degradation are of little interest to operations management and application owners. Only those requests that risk breaching service level agreements or create bottlenecks in an application need to be exposed.

OMEGAMON for JVM maintains a rolling 5-minute window of the top ten worst requests, with the total number of requests during that period, as well as the average, the longest, and the shortest duration requests during that period.

In our demonstration environment, a servlet makes the z/OS Connect API calls to an API called “catalog.” This service makes calls to CICS programs that query a database of stationary products. The service can also invoke programs to emulate customer orders for stationary products. The stand-alone z/OS Connect Liberty server uses WebSphere Optimized Local Adaptor (WOLA) to connect to CICS. This contrived example runs really quickly.

Running on a large z13 machine with little other workload to impact it, we were able to demonstrate a smoothly running system with rapid response time. Refer to Figure 3-10 and Figure 3-11.

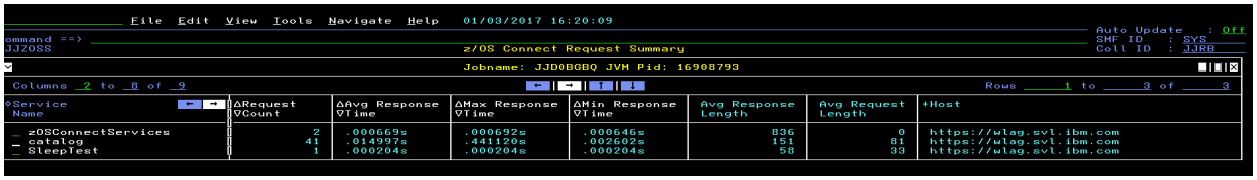


Figure 3-10 z/OS Connect Request Summary

Figure 3-11 shows the detailed snapshot of the system.

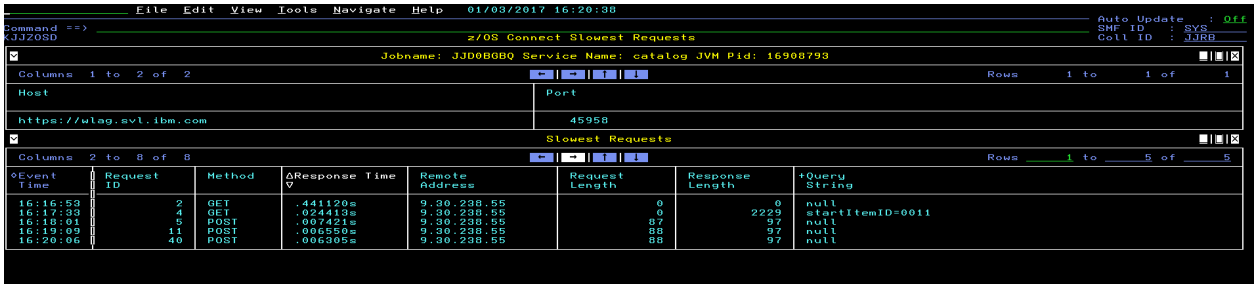


Figure 3-11 z/OS Connect Slowest Request Detail

The z/OS Connect workspaces in OMEGAMON also shows requests that query the available services on the server (shown as a service called zOSConnectServices).

Had there been extraordinary response time, we would have used OMEGAMON for JVM to determine whether the JVM was undergoing problems, and if not, use other members of the OMEGAMON product family to pinpoint the source of the issue.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

Online resources

These websites are also relevant as further information sources:

- IBM OMEGAMON for JVM on IBM Knowledge Center
  http://www.ibm.com/support/knowledgecenter/SSMTJ5
- IBM OMEGAMON for JVM on z/OS Product Home Page
  http://www.ibm.com/software/products/en/omegamon-jvm-monitor
- IBM developerWorks Answers (filtered for “OMEG” tag)
  https://developer.ibm.com/answers/topics/omeg/
- IBM developerWorks Service Management Connect z System Community Blog
  http://www.ibm.biz/zITSMBlog
- IBM z Systems IT Service Management Operational Excellence
  http://www.ibm.com/systems/z/solutions/operational-excellence/index.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

REDP-5429-00

ISBN 0738456020

Printed in U.S.A.