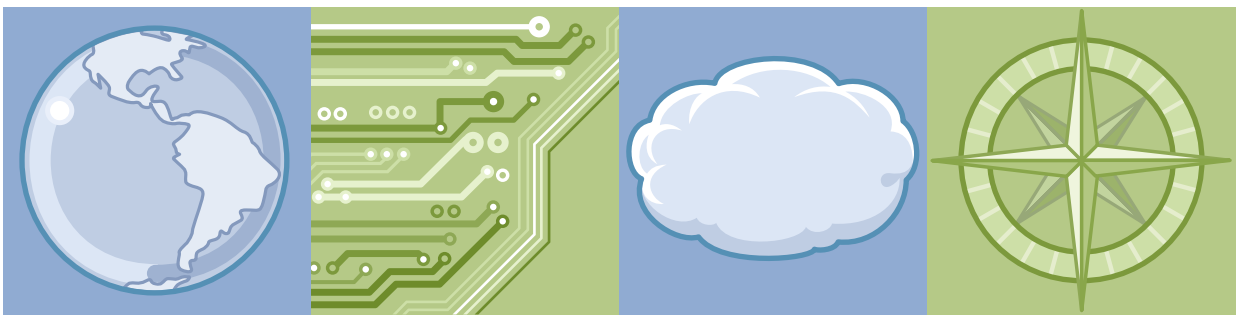
IBM Training

Student Exercises

WebSphere Application Server V8.5.5 Performance Tuning

Course code WA815 ERC 2.2



IBM Cloud

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

AFS™	AIX®	CICS®
DB™	DB2®	developerWorks®
Express®	HACMP™	IMS™
Informix®	Insight™	MVS™
PartnerWorld®	RACF®	Rational®
RDN®	Redbooks®	Tivoli®
WebSphere®	WPM®	z/OS®

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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

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

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

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Performance View® is a trademark or registered trademark of Kenexa, an IBM Company.

Other product and service names might be trademarks of IBM or other companies.

June 2017 edition

Contents

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

AFS™	AIX®	CICS®
DB™	DB2®	developerWorks®
Express®	HACMP™	IMS™
Informix®	Insight™	MVS™
PartnerWorld®	RACF®	Rational®
RDN®	Redbooks®	Tivoli®
WebSphere®	WPM®	z/OS®

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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

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

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

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Performance View® is a trademark or registered trademark of Kenexa, an IBM Company.

Other product and service names might be trademarks of IBM or other companies.

-

The exercises are mostly independent of successful completion of previous exercises, except where noted.

-
-
-
-
-
-
-
-
-
-
-

Exercises description

This course includes the following exercises:

- Exercise 1: POD configuration
- Exercise 2: Apache JMeter basics
- Exercise 3: DayTrader Benchmark installation
- Exercise 4: Using Apache JMeter to load test DayTrader
- Exercise 5: Performance monitoring tools
- Exercise 6: Exploring GC policies and monitoring JVM performance
- Exercise 7: Tuning the JVM
- Exercise 8: Troubleshooting JVM problems
- Exercise 9: Tuning JDBC connection pools and enabling servlet caching
- Exercise 10: Application profiling with Java Health Center
- Exercise 11: Load testing an application server cluster

In the exercise instructions, you can check off the line before each step as you complete it to track your progress.

Most exercises include required sections that should always be completed. It might be necessary to complete these sections before you can start subsequent exercises. Some exercises might also include optional sections that you might want to complete if you have sufficient time and want an extra challenge.

Exercise 1. POD configuration

What this exercise is about

This exercise covers the setup of your exercise POD. The term “POD” describes the group of three machines that work together for the lab exercises. A set of scripts configure the three machines to work together.

What you should be able to do

At the end of this exercise, you should be able to:

- Use the POD of three machines for the lab exercises

Introduction

The exercise uses a set of three machines, which must work together. This group of machines is collectively referred to as a POD. Individually, the machines are referred to as hostA, hostB, and hostC. All three machines start as identical images, but have unique PUBLIC IP addresses.

This exercise uses the ResetScripts icon on hostA’s desktop to start a script (1_setup_POD-hostA**)** which configures all three machines. The script first asks for the PUBLIC IP address for each of the machines (these PUBLIC IP addresses are the IP addresses that have external access to the machines). The script then generates static INTERNAL IP addresses for communications between the hosts within the POD and maps those addresses to the host names hostA, hostB, and hostC.

After the network is configured on hostA, the setup script configures the appropriate WebSphere profiles and then makes calls to hostB and hostC (using their PUBLIC IP addresses) to complete their configurations.

After the script finishes, the POD is ready to use for the course exercises. All further exercise instructions can run on hostA, so there is no need to remote desktop into hostB or hostC.

Requirements

Access to the three lab machines is necessary. Aside from needing the PUBLIC IP addresses for all three hosts, remote desktop access to at least hostA is required.

Important

New versions of IBM Support Assistant tools

A new course image was released in February 2016. It has an updated version of IBM Support Assistant Team Server 5.0.2.1 and updated tools with valid security certificates.

You might notice slight variations between some of the screen captures in the exercises and how the new versions of the tools appear. In general, the functions that are required for these exercises are unaffected in this edition of the course material.

Exercise instructions

Section 1: Gather your IP addresses for hostA, hostB, and hostC

These exercises use a collection of three machines. This collection is referred to as a pod and must be configured to work together.

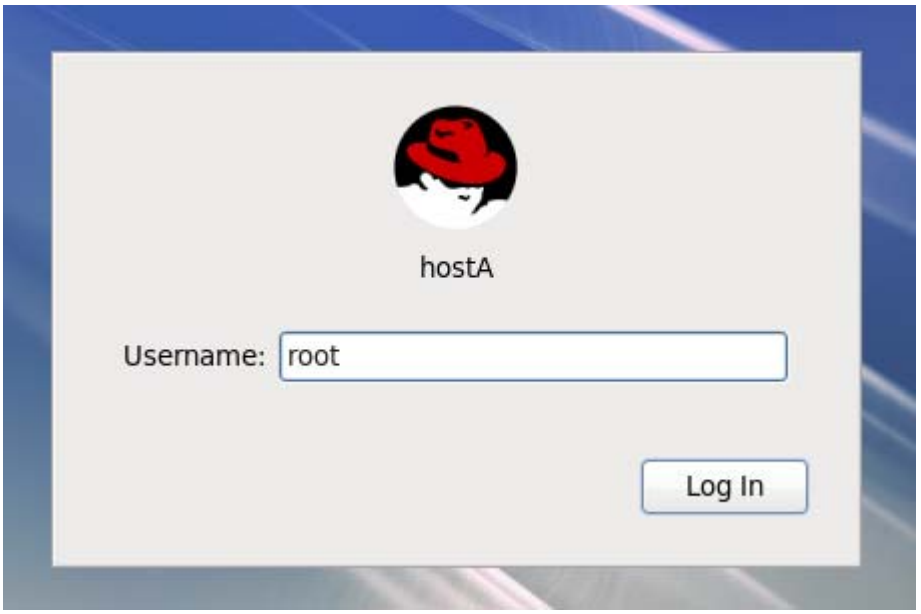
The process of configuring the pod is done through a script that requires the PUBLIC IP address for each of the three hosts (hostA, hostB, and hostC). Depending on how the lab exercises are delivered, these addresses were already supplied to you or you must find them on your three machines.

If you already know the IP addresses for your hosts, skip this section and proceed to Section 2.

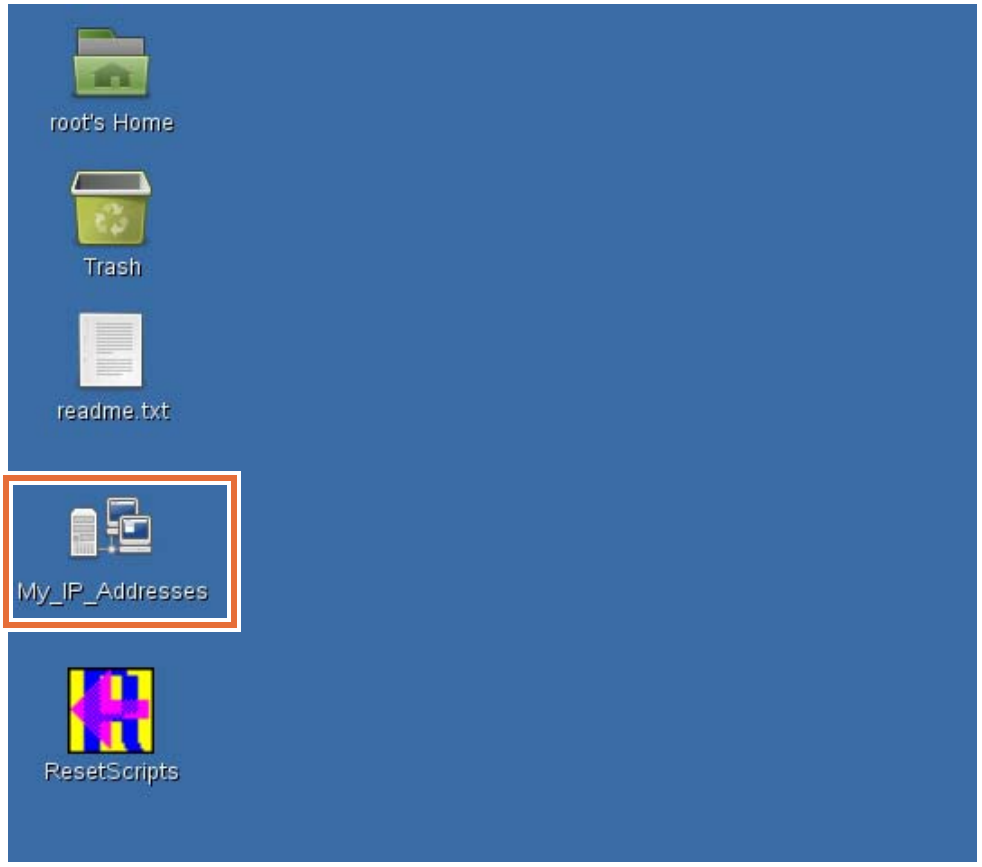
__ 1. Connect to hostA.

__ a. Connect to the desktop for **hostA**. If challenged to log in, use the following credentials:

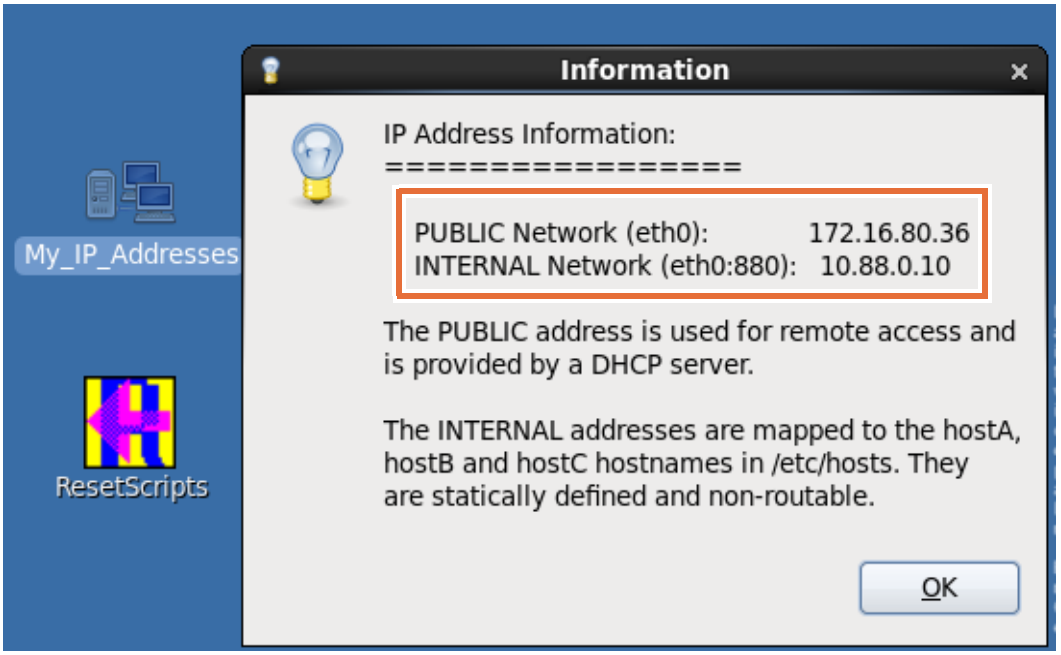
```
Username: root
Password: web1sphere
```



__ b. On the desktop, double-click **My_IP_Addresses**.



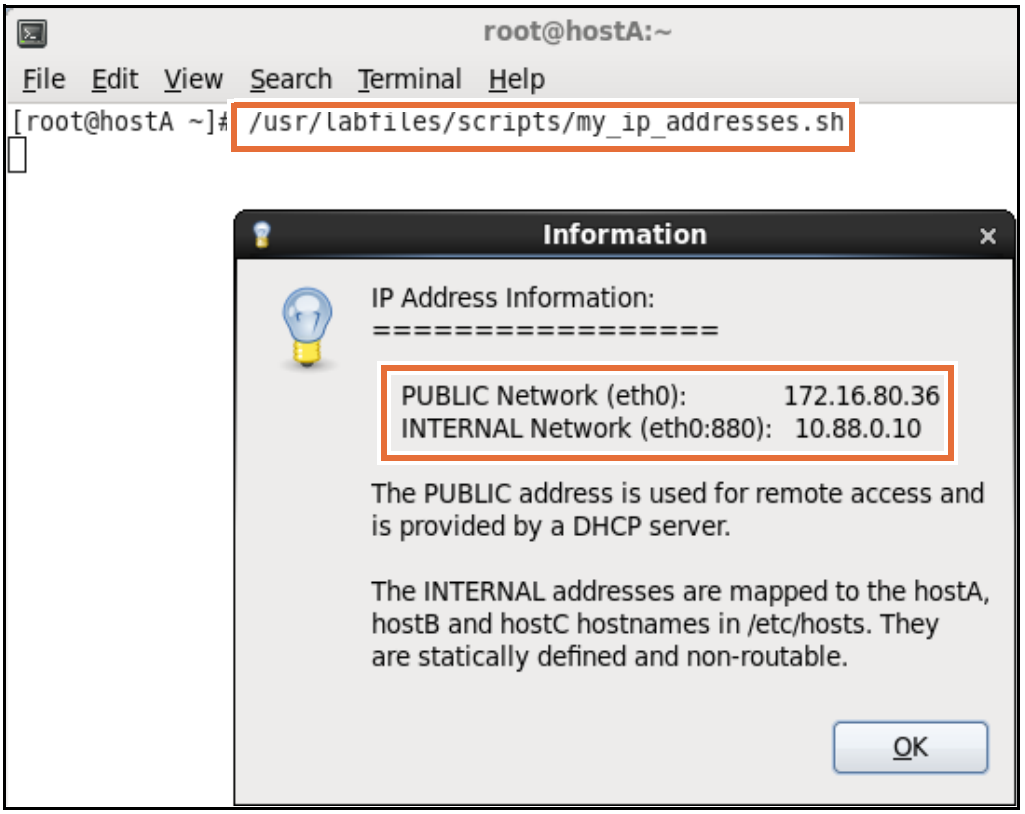
__ c. A window opens and lists **two IP addresses**.



Information

It is also possible to run the `my_ip_addresses.sh` script from the command line:

`/usr/labfiles/scripts/my_ip_addresses.sh`



__ d. Take note of the PUBLIC IP address that is displayed and enter it in the table.

__ e. Click **OK** to close the window.

__ 2. Repeat the previous steps for **hostB**.

__ 3. Repeat the previous steps for **hostC**.

__ 4. Enter the **PUBLIC IP addresses** for each of the hosts in the POD.

Table 1: POD PUBLIC IP addresses

Host name	PUBLIC IP address
hostA	
hostB	
hostC	

Section 2: Configuring the lab machines (POD)

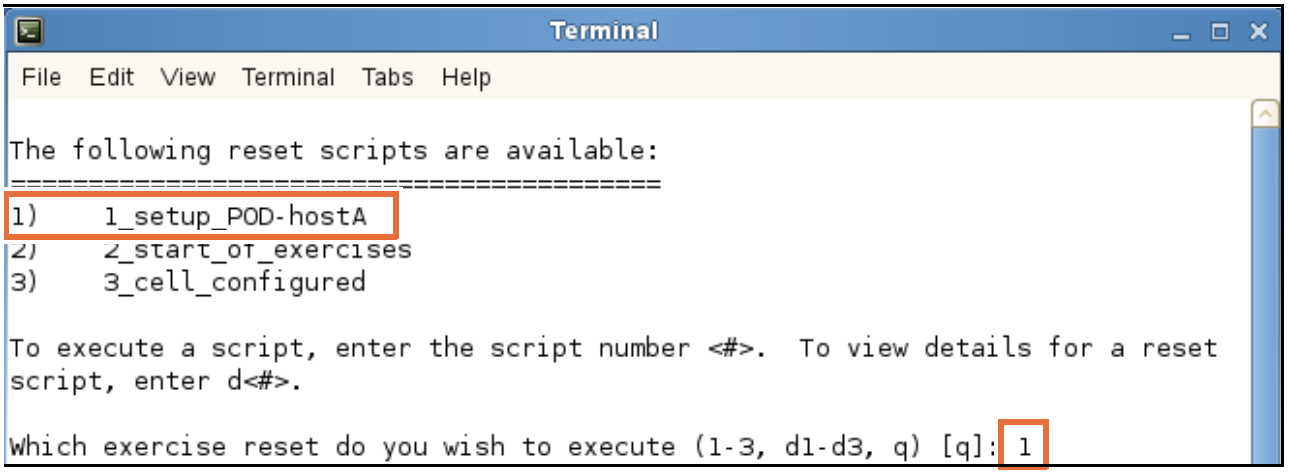
This section goes through the process of configuring the lab machines (hostA, hostB, and hostC) to work together as a POD. A single script is run on hostA, which in turn makes remote calls to hostB and hostC to configure those machines.

__ 1. Connect to the desktop for hostA and run **ResetScripts** to configure the POD.

The mechanism for connecting to the desktop depends on how the class is delivered. If you did not receive the steps to connect to the remote desktop, your instructor can assist.

__ a. From the desktop, double-click **ResetScripts**.

__ b. A command window shows the list of possible reset scripts available. Enter **1** to select the **1_setup_POD-hostA** script, and press **Enter**.



Important

Reset scripts

The only reset script that works for the WA815 course is (1) `setup POD-hostA`. Do not run scripts (2) or (3) at any time during this course. Running these scripts makes your images unusable. You

can run the `setup POD-hostA` script later in the course, if you need to start over. However, you must then redo the following exercises:

- *Exercise 3. DayTrader Benchmark installation*: Sections 1 and 2.
- *Exercise 4. Use Apache JMeter to load test DayTrader*: Sections 1 and 2.

Note

If necessary, it is possible to run the setup_POD-hostA script again. For example, If the PUBLIC IP addresses change during the exercises, running this setup script can help correct the configuration. Running the setup POD script returns the WebSphere configuration to the beginning of the exercises. Therefore, it would be necessary to use the ResetScripts again to reset the state to the appropriate state.

__ c. When asked **Do you want to continue (y/n)**, press Enter to accept the default of **y**.

```
Which exercise reset do you wish to execute (1-3, d1-d3, q) [q]: 1

Running script: /usr/labfiles/reset/reset_scripts/reset_1_setup_POD-hostA.sh

=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

This script will reset the profiles to a state expected at the

    1_setup_POD-hostA

Do you want to continue (y/n) [y]:
```

__ 2. Enter the PUBLIC IP addresses for each of the three hosts in the POD.

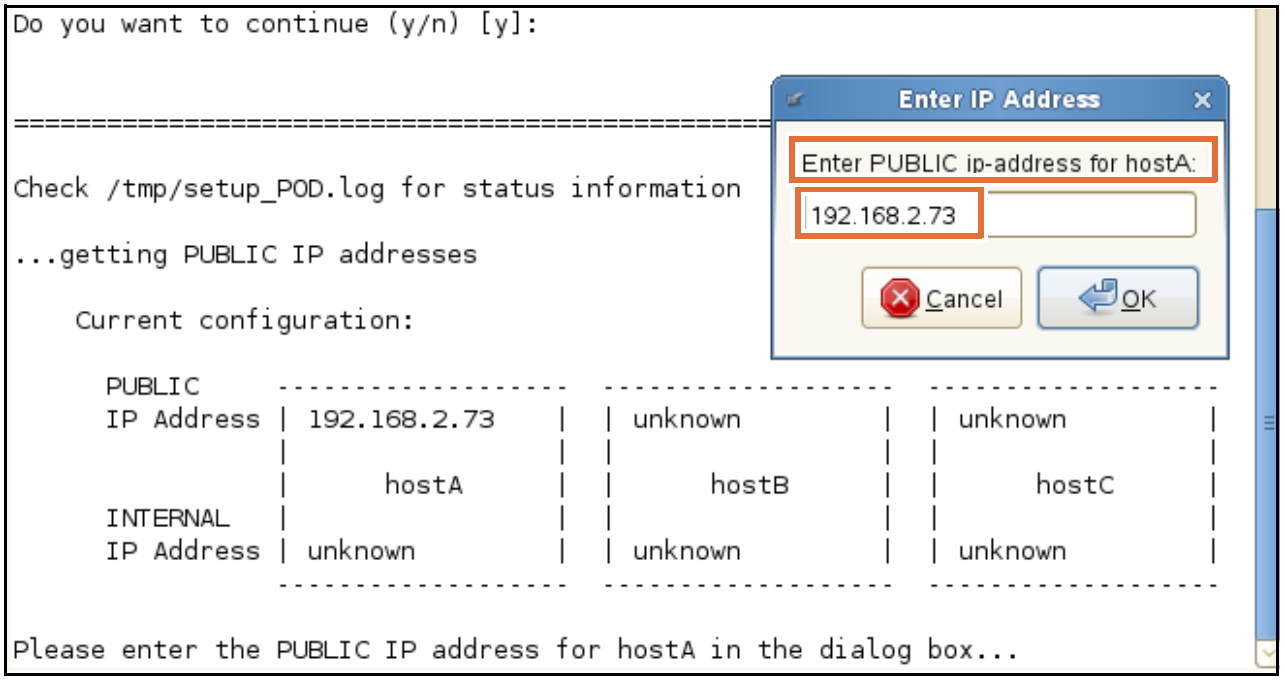
__ a. In the main terminal window, a POD diagram that shows the current IP addresses is displayed. Notice that most of the IP addresses are listed as unknown.

Information

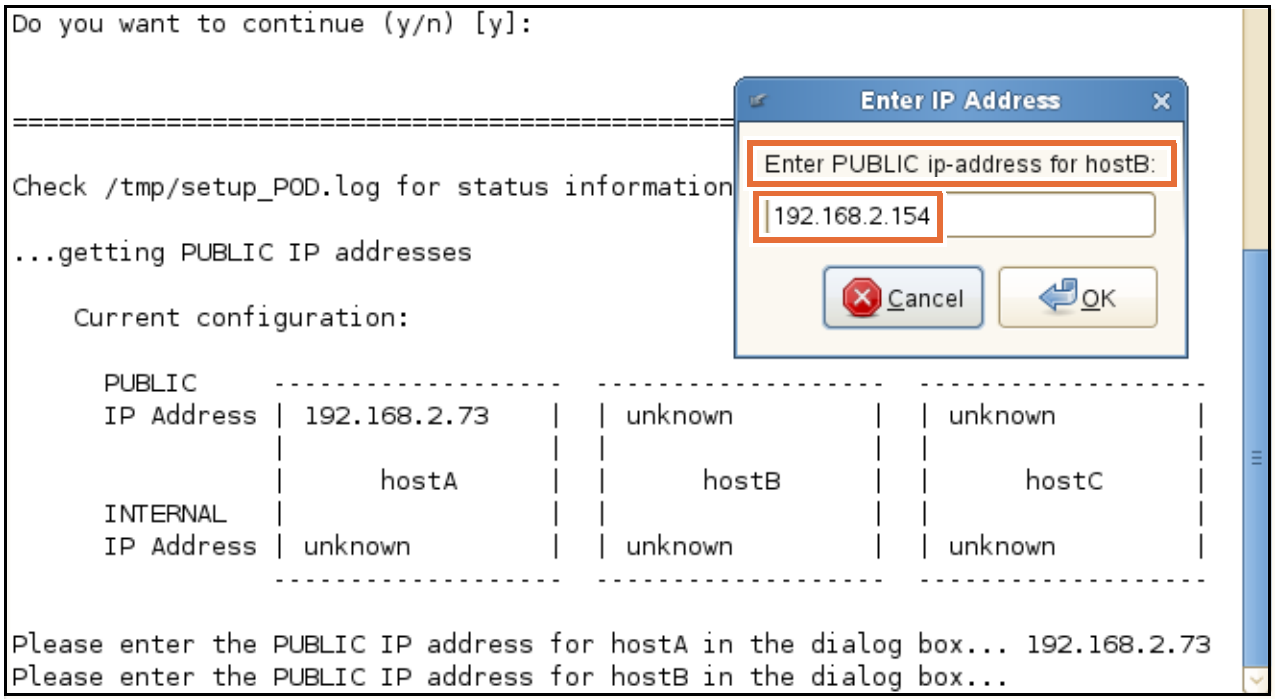
For debugging information, monitor the log file with the following command:

```
tail -f /tmp/setup_POD.log
```

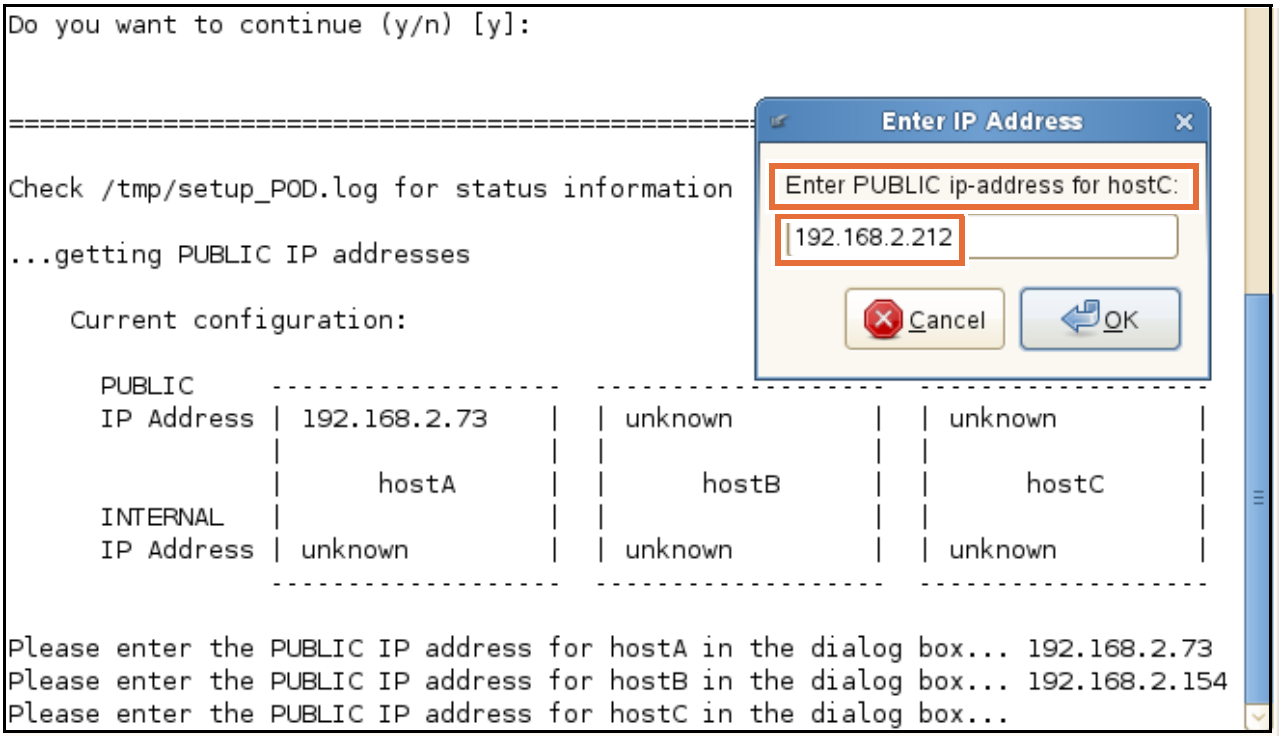
__ b. A dialog also box opens asking for the **PUBLIC IP address** for **hostA**. Because this script is running on hostA, the value is prefilled. If the value is not correct, enter the current PUBLIC IP address (`eth0`). Click **OK** to accept.



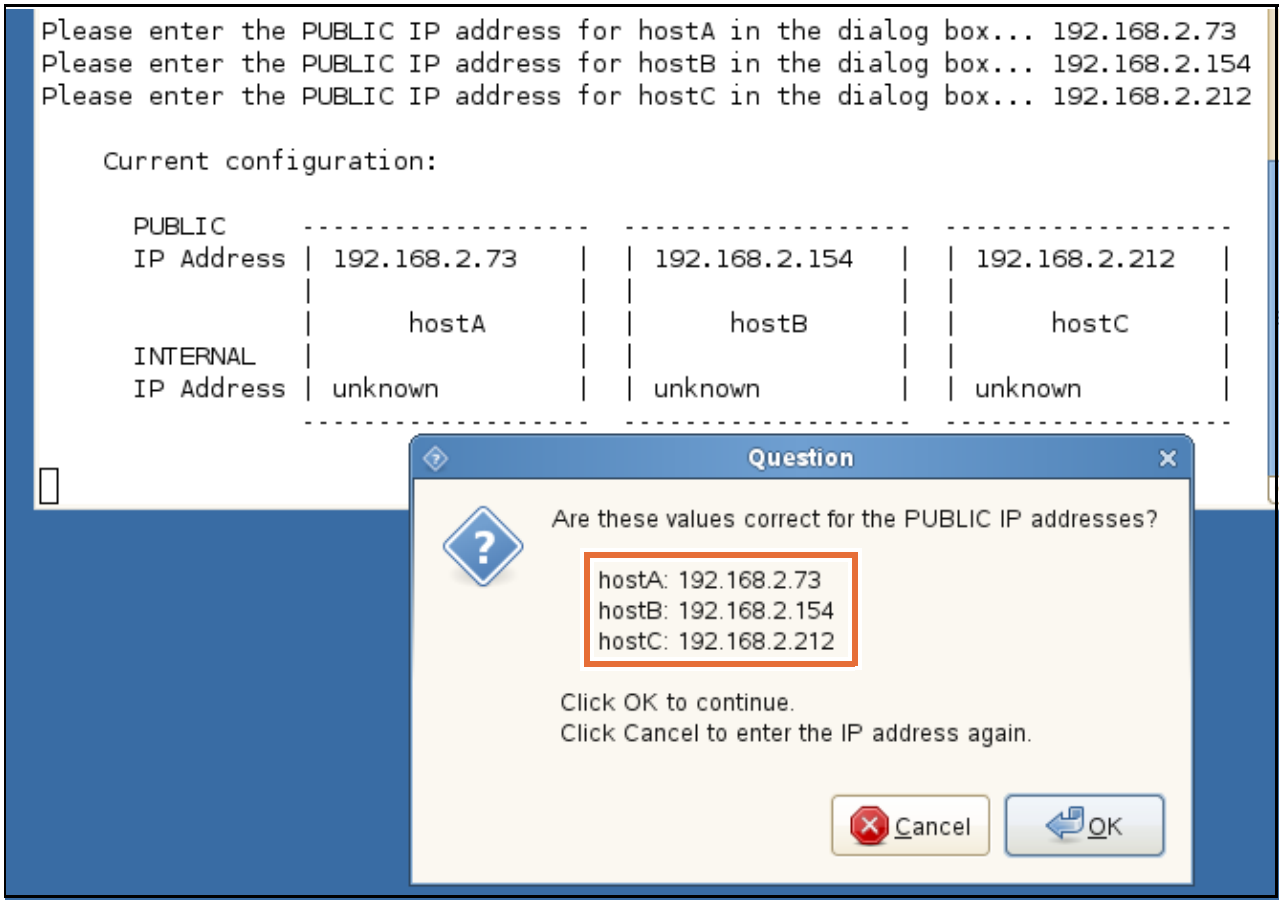
__ c. Enter the **PUBLIC IP address** for **hostB** and click **OK**.



__ d. Enter the **PUBLIC IP address** for **hostC** and click **OK**.



__ e. A window is displayed allowing you to confirm the addresses that were entered. If they are correct, click **OK**. If any updates are needed, click **Cancel**. Notice that the main terminal window also lists the PUBLIC IP addresses in the POD diagram.

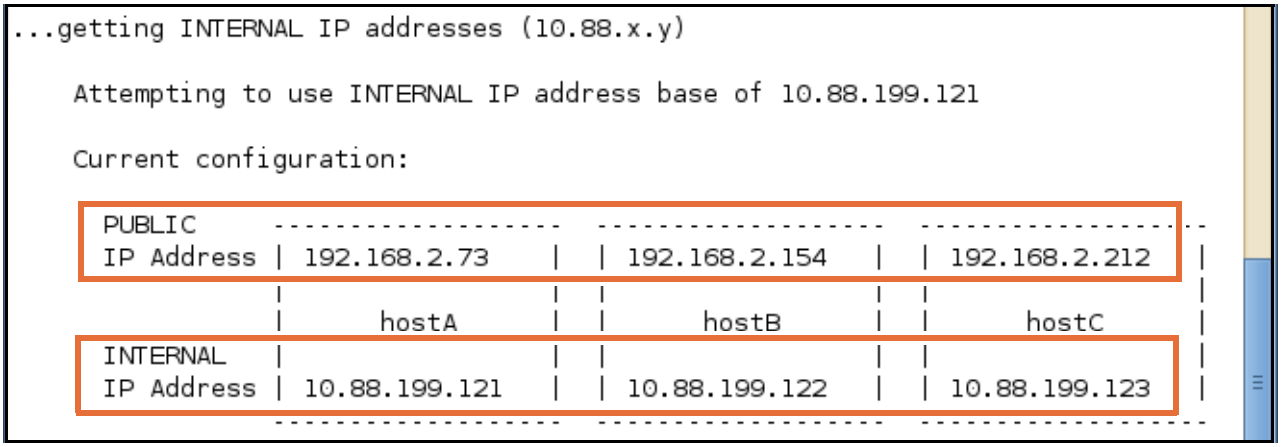


__ 3. Configure the INTERNAL IP addresses.

The PUBLIC IP addresses are used to access the lab machines from across the network. The INTERNAL IP addresses are statically defined on `eth0:880` (a virtual network card)

and used for communications between the POD members. Using static IP addresses ensures that the POD members can communicate even if a DHCP address changes.

__ a. Notice that the ResetScripts window displays the message **"getting INTERNAL IP addresses (10.88.x.y)"**. After a few moments, a new POD diagram is displayed with both the **PUBLIC** and **INTERNAL** IP addresses displayed



__ b. Wait a few moments for the reset to complete.

Information

What is going on?

During this step, the script chooses a set of random numbers (x and y) to define the INTERNAL IP address for hostA in the form 10.88.x.y. The addresses for hostB and hostC are assigned 10.88.x.(y+1) and 10.88.x.(y+2). For example:

```
hostA - 10.88.199.121
hostB - 10.88.199.122
hostC - 10.88.199.123
```

Each of the three addresses is pinged to ensure that it is not in use. If a response comes back, a new INTERNAL address is chosen for each machine until no conflicts are found.

__ c. If prompted for a **keyring password**, enter **web1sphere** in both password fields and click **Create**.

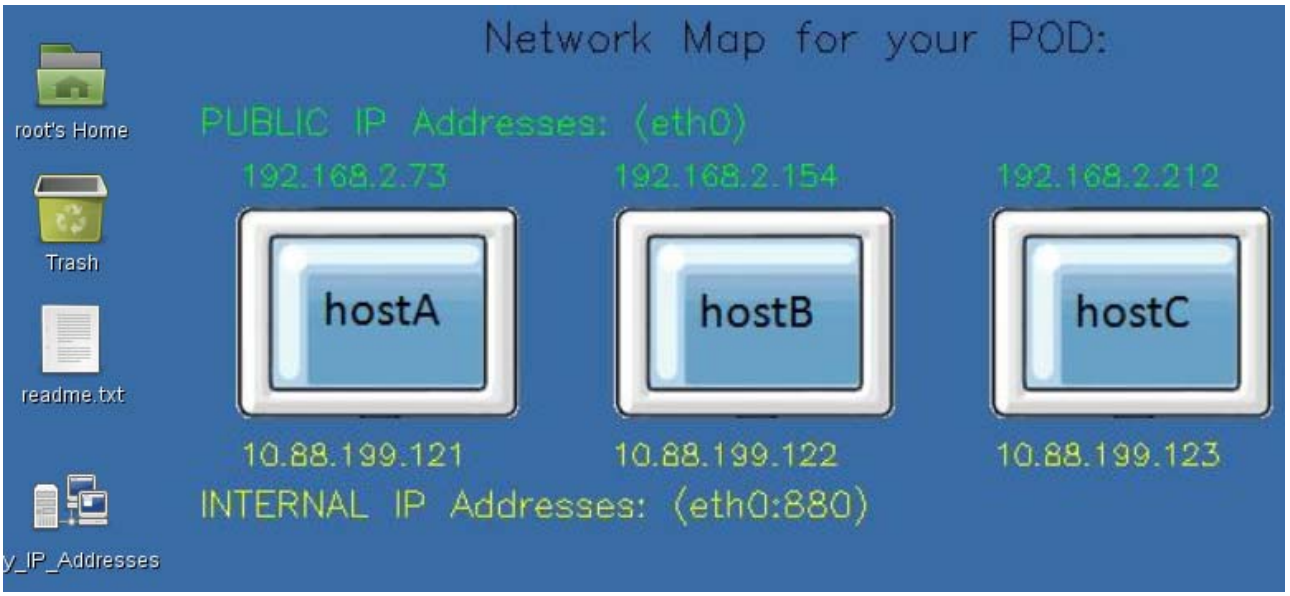
__ d. Notice the messages: **Remotely configuring hostB** and **Remotely configuring hostC**. These messages indicate that the configuration scripts ran on the other POD machines.

__ e. Wait until you see the message Reset Complete, then press Enter.

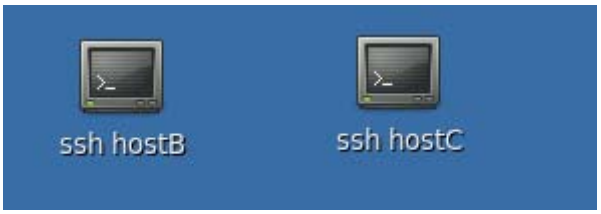
```
  Current configuration:

    PUBLIC     -------------------  -------------------  -------------------
    IP Address | 192.168.2.73    |  | 192.168.2.154   |  | 192.168.2.212   |
               |                 |  |                 |  |                 |
               |      hostA      |  |      hostB      |  |      hostC      |
    INTERNAL   |                 |  |                 |  |                 |
    IP Address | 10.88.199.121   |  | 10.88.199.122   |  | 10.88.199.123   |
               -------------------  -------------------  -------------------

Remotely configuring hostB (192.168.2.154)
Remotely configuring hostC (192.168.2.212)

Reset Complete.  Press Enter to continue...
```

__ 4. Confirm that the final POD diagram as part of the wallpaper on hostA.

__ a. After the POD setup is complete, the final POD diagram is displayed as part of the desktop. If you do not see the diagram, log off and log in again.



__ 5. Also, notice that you have two icons on the desktop of hostA, **ssh hostB** and **ssh hostC**. You can use these icons to get an ssh terminal window for hostB and hostC in the remaining exercises.



Important

Use hostA for most of your work.

In most of the exercises that follow, you work mainly from hostA unless directed otherwise.

End of exercise

Exercise review and wrap-up

The exercise used the ResetScripts to configure the student POD (hostA, hostB, and hostC).

Exercise 2. Apache JMeter basics

What this exercise is about

This exercise provides a basic introduction to using Apache JMeter to create test plans that simulate multiple users who access a web-based application. You also learn how to monitor performance metrics such as response time and throughput by using some JMeter listeners. Finally, you learn how to use the JMeter Test Script Recoder to generate test scripts by navigating a website.

What you should be able to do

At the end of this exercise, you should be able to:

- Start Apache JMeter
- Create a simple test plan
- Run the test plan and monitor application performance
- Use JMeter to record a test script by navigating a remote website

Introduction

The Apache JMeter desktop application is open source software, a 100% pure Java application that is designed to load test functional behavior and measure performance. It can be used to simulate a heavy load on a server, group of servers, network, or object to test its strength or to analyze overall performance under different load types.

For detailed information about how to use Apache JMeter, go to the website `http://jmeter.apache.org/index.html`

Requirements

This exercise requires that Apache JMeter is installed on the lab image. The image for this course has Apache JMeter installed.

Exercise instructions

Preface

- Most steps in this exercise are done on **hostA** unless otherwise specified.

Note

<profiles_root> = /opt/IBM/WebSphere/AppServer/profiles

Section 1: Start the WebSphere servers

__ 1. On **hostA**, start the Deployment Manager.

__ a. Open a terminal window and enter the following command:
cd <*profiles_root*>/Dmgr/bin

__ b. Enter the command:
./startManager.sh

__ 2. On **hostB**, start the node agent and server1.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
cd <*profiles_root*>/profile1/bin

__ c. Enter the command:
./startNode.sh

__ d. After the node agent starts successfully, start server1 by entering the command:
./startServer.sh server1

Section 2: Start JMeter and create a simple test plan

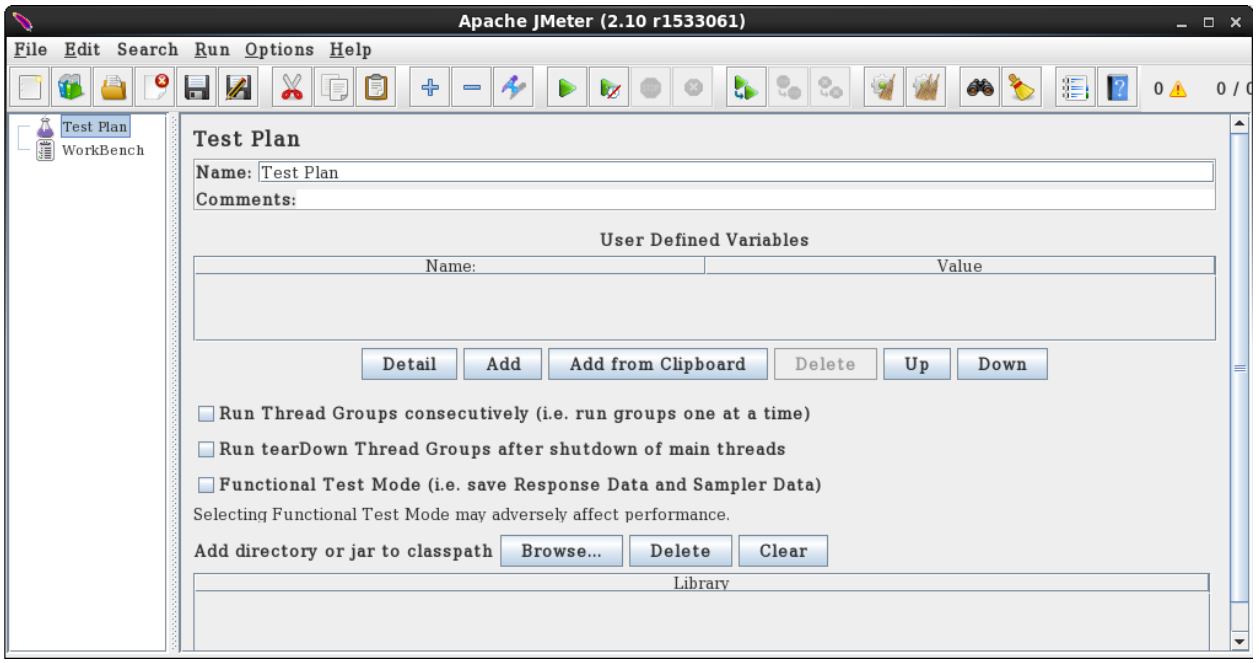
All the steps in this section are done from **hostA**.

__ 1. Start JMeter.

__ a. Open a terminal window and enter the command:
cd /opt/apache-jmeter/bin

__ b. Enter the command:
./jmeter.sh &

.



The JMeter GUI consists of a hierarchical tree view, a workspace, and toolbars.

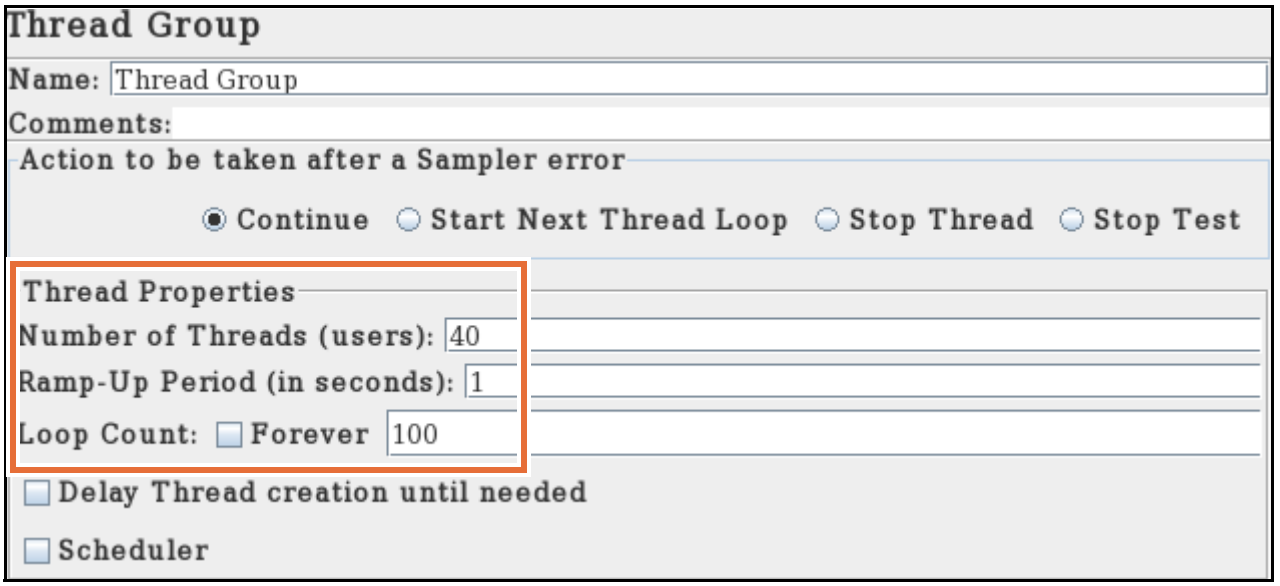
__ 2. Add a Thread Group. Thread group elements are the beginning points of any test plan. The thread group element controls the number of threads (users) JMeter uses to run the test.

__ a. In the tree, right-click **Test Plan** and click **Add > Threads (Users) > Thread Group**.

__ b. Notice in the tree that a Thread Group is added under the Test Plan. Select **Thread Group**.

__ c. Enter the following properties for the thread group.

- Number of Threads (users): **40**
- Ramp-Up period: **1**
- Loop Count: **100**



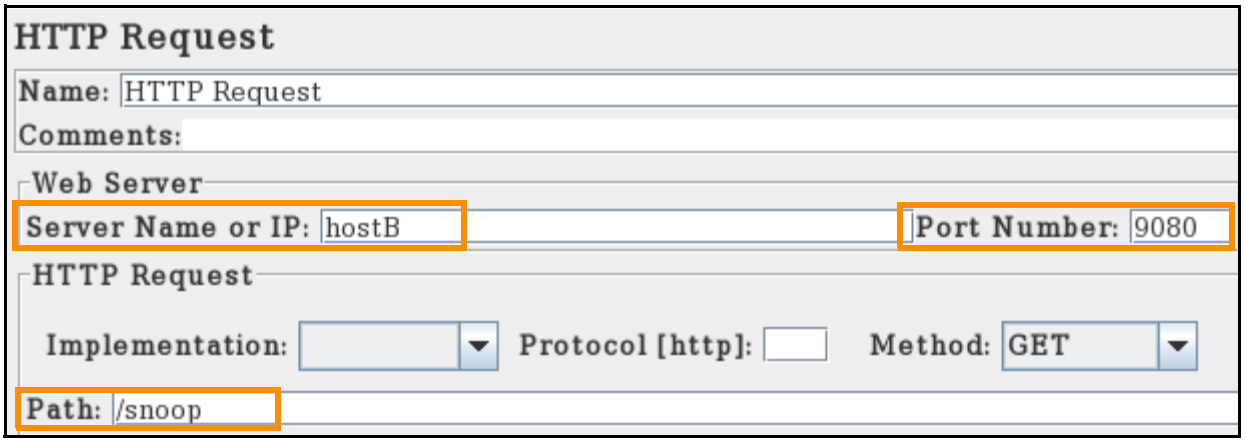
__ 3. Add a sampler for HTTP requests. Samplers tell JMeter to send requests to a server and wait for a response.

__ a. Right-click **Thread Group** and click **Add > Sampler > HTTP Request**.

__ b. Notice that the HTTP Request sampler is added under the Thread Group in the tree. Select **HTTP Request**.

__ 4. Enter the following data for the HTTP Request.

- Server Name: **hostB**
- Port Number: **9080**
- Path: **/snoop**



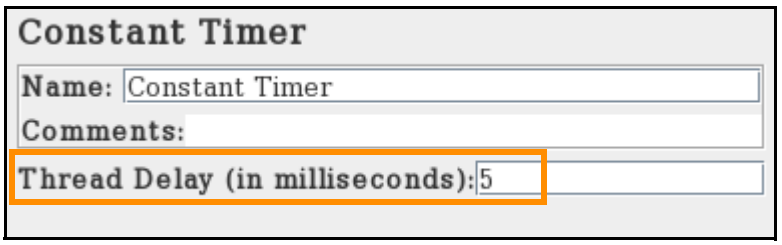
__ 5. Add a timer for the HTTP Request sampler. A timer adds “think time” in between requests. The timer simulates a user who clicks a page, and then pauses to read some information on the page, before making another request.

__ a. Right-click **HTTP Request** in the tree and click **Add > Timer > Constant Timer**.

__ b. Notice that the **Constant Timer** is added under HTTP Request in the tree. Select Constant Timer.

__ c. Enter the following property for the Constant Timer.

- Thread Delay: **5**

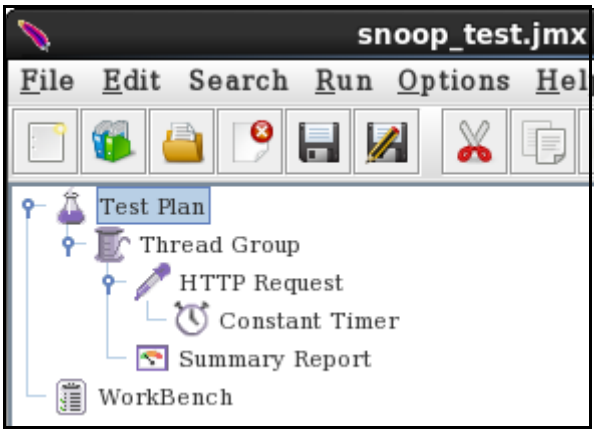


__ 6. Add a listener. Listeners display the data that JMeter gathers about the test plan while JMeter is running.

__ a. Right-click the **Thread Group** in the tree and click **Add > Listener > Summary Report**.

__ b. Notice that the **Summary Report** is added under Thread Group in the tree. The Summary Report shows number of samples, average response times, and throughput while the test runs.

__ 7. Verify that your test plan looks like the following tree.



__ 8. Save the test plan.

__ a. Click **File > Save Test Plan as**.

__ b. Navigate to **`/usr/labfiles/Test_Plans`**, and name the plan **`snoop_test.jmx`**

__ c. Click **Save**.

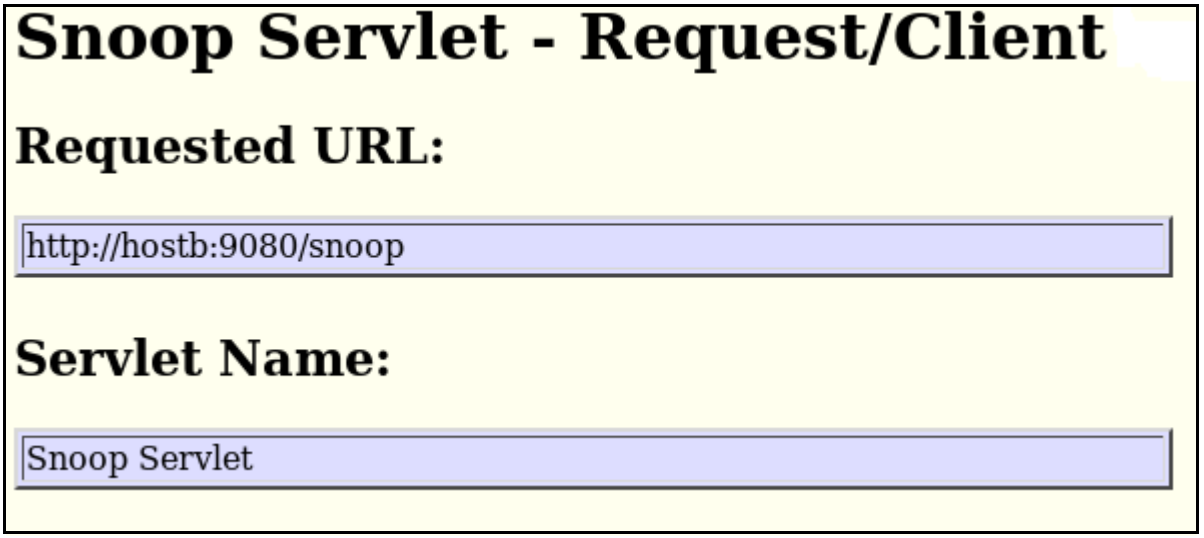
Section 3: Run the snoop test plan

In this section, you run the simple test plan, and view the performance results by using the JMeter summary report listener.

__ 1. Verify that the snoop application can be accessed by using a browser.

__ a. From **hostA**, start a web browser and enter the URL **`http://hostB:9080/snoop`**

__ b. Verify that you see the familiar snoop servlet page.

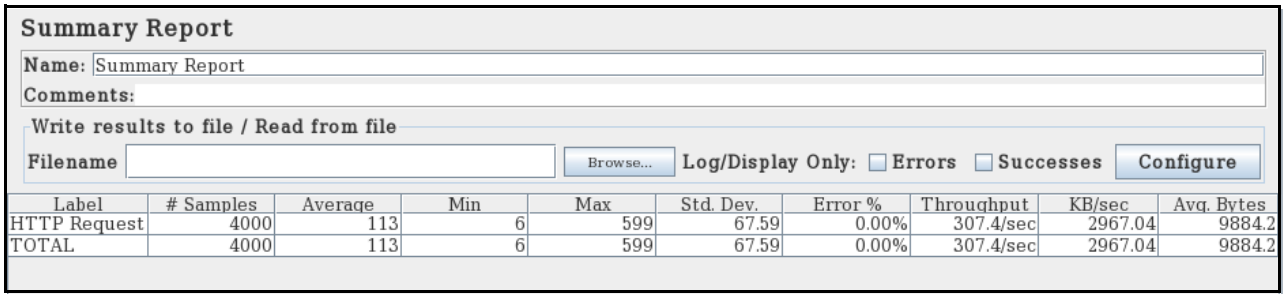


__ 2. Run the test in JMeter.

__ a. Click Summary Report in the tree to show the Summary Report page in the workspace.

__ b. Click **Run > Start.** Alternatively, you can click the green arrow on the toolbar to run the test.

__ c. As the test runs, observe the response time and throughout values in the Summary report.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
HTTP Request	4000	113	6	599	67.59	0.00%	307.4/sec	2967.04	9884.2
TOTAL	4000	113	6	599	67.59	0.00%	307.4/sec	2967.04	9884.2

__ d. The test stops after 4000 samples (No. of users x Loop count). The average response time is in milliseconds and the throughput is the number of requests per second.

__ e. Record your results here for the first test run.

- Response time (ms)_____
- Throughput (req/sec)_____

__ 3. Run the snoop test a second time.

__ a. Clear the results from the last test by clicking **Run > Clear**. Alternatively, click the "broom" icon to clear the results.

__ b. Click the green arrow to run the test again and observe the response time and throughput.

__ c. After the test completes record your results for the second test.

- Response time (ms)_____
- Throughput (req/sec)_____

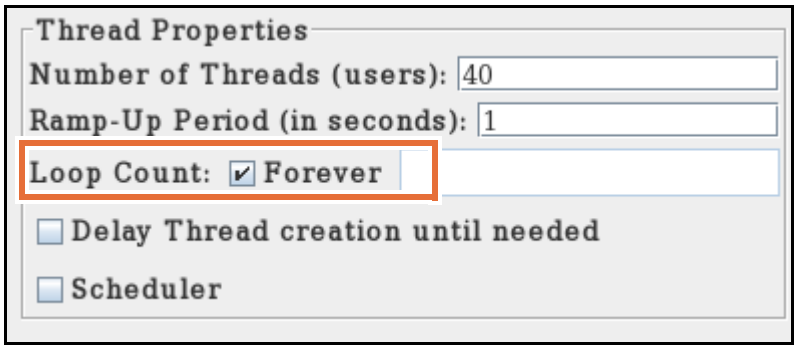
__ d. Compare the performance results for the two tests. Most likely you notice a faster response time and an increased throughput for the second test. One reason for the improved results is that the first test run serves to "warm-up" the application server. Java methods are compiled and some caching might occur.

__ 4. Modify the Thread Group elements.

__ a. In the tree, select **Thread Group**.

__ b. Make the following changes to the Thread Properties.

- Loop Count: **Forever**

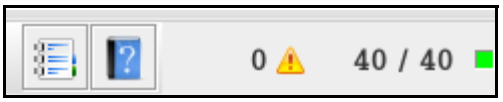


__ c. Clear the results from the last test by clicking **Run > Clear** or the "broom" icon.

__ d. Click the green arrow to run the test.

__ e. As the test runs, observe the response time and throughout values in the summary report. Initially both the response time and the throughput increase as the test runs. Eventually the performance reaches a steady state.

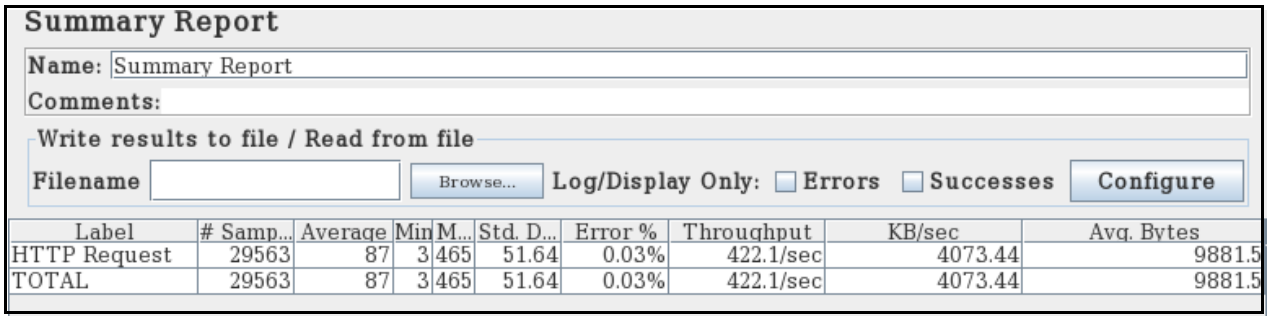
__ f. Notice the icons in the upper left taskbar. They show if any errors are logged, the number of threads that are running (40/40), and the status of the test run.



If there are any errors, you can click the yellow triangle icon to open a log viewer pane.

__ g. Run the test until throughput no longer increases. Since the loop count is forever, you must stop the test manually. You can stop the test by clicking **Run > Stop** or the red Stop icon.

__ h. You might notice that when you stop a test run manually, there is a percentage error. These errors can happen if some of the threads are stopped before they complete successfully.



Label	# Samp...	Average	Min	M...	Std. D...	Error %	Throughput	KB/sec	Avg. Bytes
HTTP Request	29563	87	3	465	51.64	0.03%	422.1/sec	4073.44	9881.5
TOTAL	29563	87	3	465	51.64	0.03%	422.1/sec	4073.44	9881.5

However, if you see a nonzero Error during a test run, it might be due to errors in the server environment. You then need to troubleshoot these errors.

__ 5. Stop JMeter.

__ a. Save any changes to the test plan by clicking **File > Save**, or Ctrl + S.

__ b. Click **File > Exit**.

Section 4: Create a test plan that uses an external web server

In this section, you are going to modify your test plan so that HTTP requests are sent to an external web server and then the HTTP plug-in routes them to the application server.

__ 1. Start JMeter.

__ a. Open a terminal window and enter the command:
`cd /opt/apache-jmeter/bin`

__ b. Enter the command:
`./jmeter.sh &`

__ 2. Open the **`snoop_test.jmx`** test plan.

__ a. Click **File > Open**.

__ b. Browse for **`/usr/labfiles/Test_Plans/snoop_test.jmx`**

__ c. Click **Open**.

__ 3. Modify the HTTP Request Element.

__ a. In the tree, click **HTTP Request**.

__ b. Change the HTTP Request as shown.

- Server Name: **hostA**

- Port Number: **80**

__ 4. Save the modified test plan.

__ a. Click **File > Save Test Plan as**.

__ b. Enter the name: **snoop_testvialHS.jmx**

__ c. Click **Save**.

__ 5. Before you can run the new test, you must ensure that the HTTP server is running and that the snoop applications is mapped to the web server.

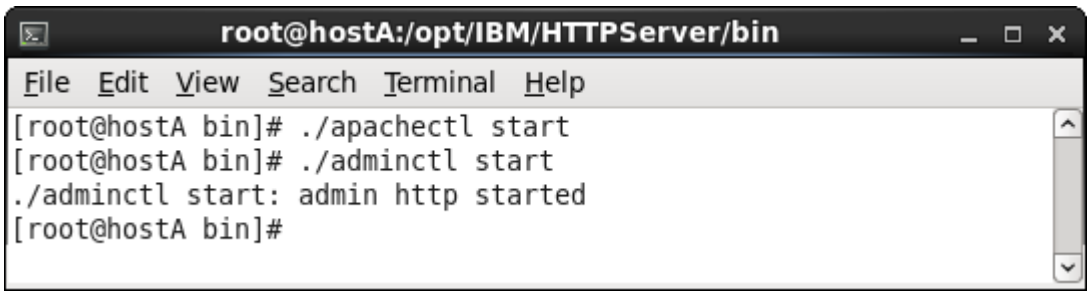
__ 6. Start the HTTP Server.

__ a. On hostA, open a terminal window and
`cd /opt/IBM/HTTPServer/bin`

__ b. Enter the following commands:

`./apachectl start`

`./adminctl start`



__ 7. Map the snoop application to the web server.

__ a. Open a Firefox web browser and enter the URL **`http://hosta:9060/ibm/console`**

Information

Browser message: This Connection is Untrusted.

You might see this message several times throughout the course. Follow these steps to handle this message.

i. Click **I understand the Risks**.

ii. On the next screen, click **Add Exception...**

iii. The next screen provides detail about this security exception. Examine this screen; then click **Confirm Security Exception**.

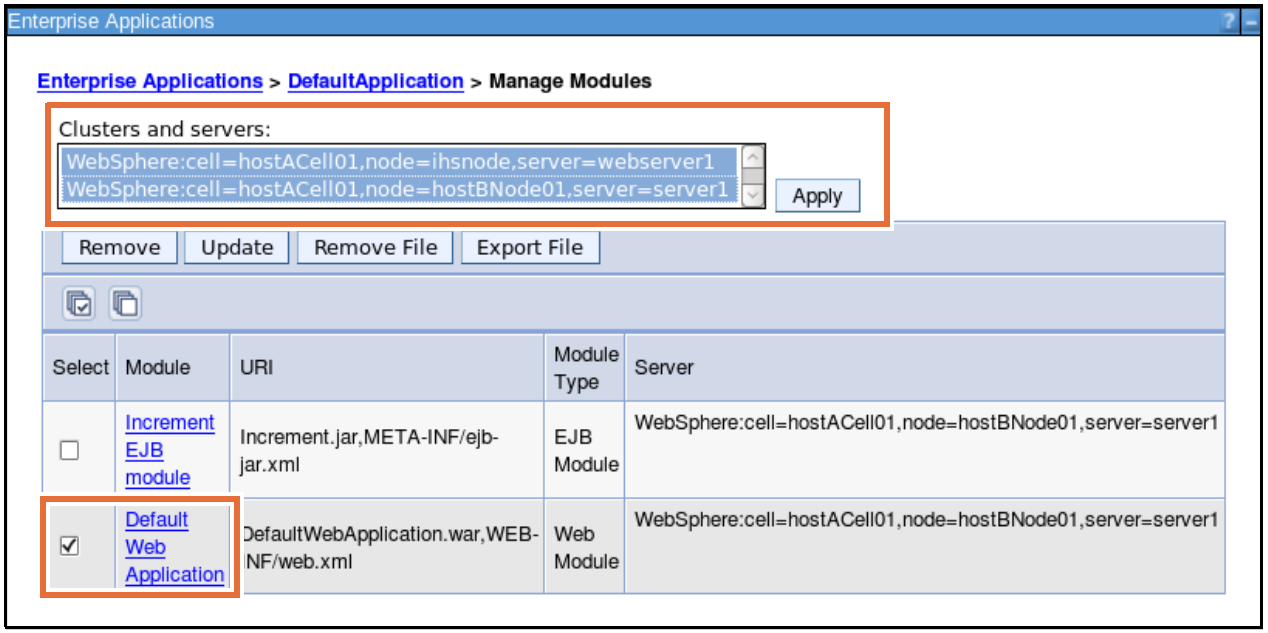
__ b. Log in to the administrative console as user: **wasadmin** with password: **web1sphere**.

__ c. Click **Applications > Application Types > WebSphere enterprise applications**.

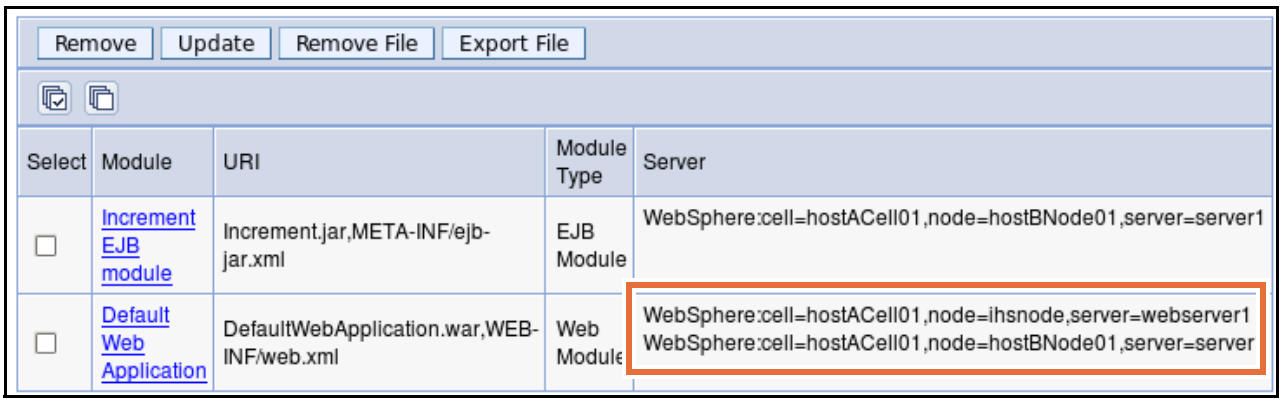
__ d. Click **DefaultApplication**.

__ e. Under Modules, click **Manage Modules**.

__ f. Select the **Default Web Application** and map it to **server1** and **webserver1**.



__ g. Click **Apply**.



__ h. Click **OK** and save the changes.

__ i. Use the administrative console to regenerate and propagate the plug-in for **webserver1**.

__ 8. Run the test plan.

__ a. Return to JMeter and select the **Summary Report**.

__ b. Click the green arrow to run the test.

__ c. Run the test until the throughput appears to reach a steady state. Record the performance results.

- Response Time (ms)_____
- Throughput (req/sec)_____

__ d. Compare the performance results with the previous test. Because requests are now routed through the web server and plug-in, you might see a higher response time and a lower throughput.

__ e. Save the test plan by clicking **File > Save**.

__ f. Stop JMeter by clicking **File > Exit**.

__ 9. Stop server1.

__ a. In the terminal window, enter the command:
cd <*profiles_root*>/profile1/bin

__ b. Stop server1 by entering the command:
./stopServer.sh server1 -user wasadmin -password web1sphere

Section 5: Use the JMeter HTTP(S) Test Script Recorder

In this section, you learn how to record test scripts by navigating the pages of a website.

__ 1. Start JMeter.

__ a. Open a terminal window and enter the command:
cd /opt/apache-jmeter/bin

__ b. Enter the command:
./jmeter.sh &

__ 2. In the tree, right-click **Test Plan** and click **Add > Threads (Users) > Thread Group**.

__ 3. In the tree, right-click **Thread Group** and click **Add > Config Element > HTTP Request Defaults**.

__ 4. In the HTTP Request Defaults pane, enter the following data.

- Server Name: **http://jmeter.apache.org/**

__ 5. Add a recording controller.

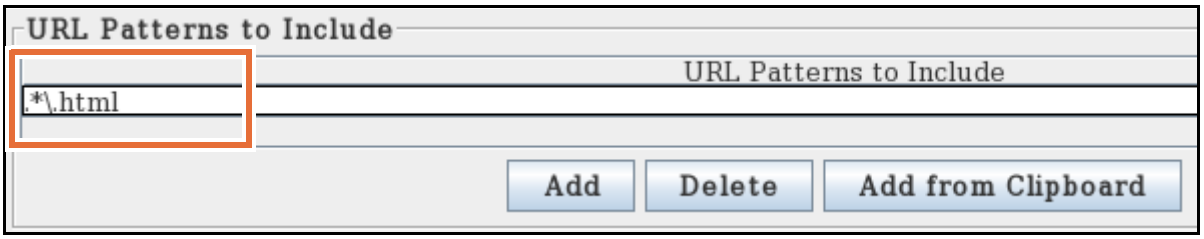
__ a. Right-click **Thread Group** and click **Add > Logic Controller > Recording Controller**.

__ 6. Add an HTTP(S) Test Script Recorder.

__ a. In the tree, right-click **WorkBench** and click **Add > Non-Test Elements > HTTP(S) Test Script Recorder**.

__ b. In the HTTP(S) Test Script Recorder pane, under **URL Patterns to Include**, click **Add** to create a blank entry.

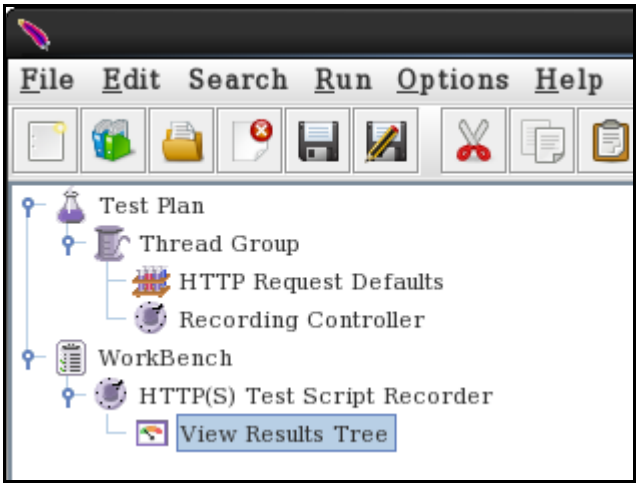
__ c. Enter the following pattern:
.*\.html



__ 7. Add a listener.

__ a. Right-click **HTTP(S) Test Script Recorder** and click **Add > Listener > View Results Tree**.

__ b. Your expanded tree should like the following tree.



__ 8. Start the script recorder.

__ a. Select **HTTP(S) Test Script Recorder**, and click the **Start** at the bottom of the pane.

__ 9. Configure the browser to use the Test Script Recorder.

__ a. Start the Firefox browser.

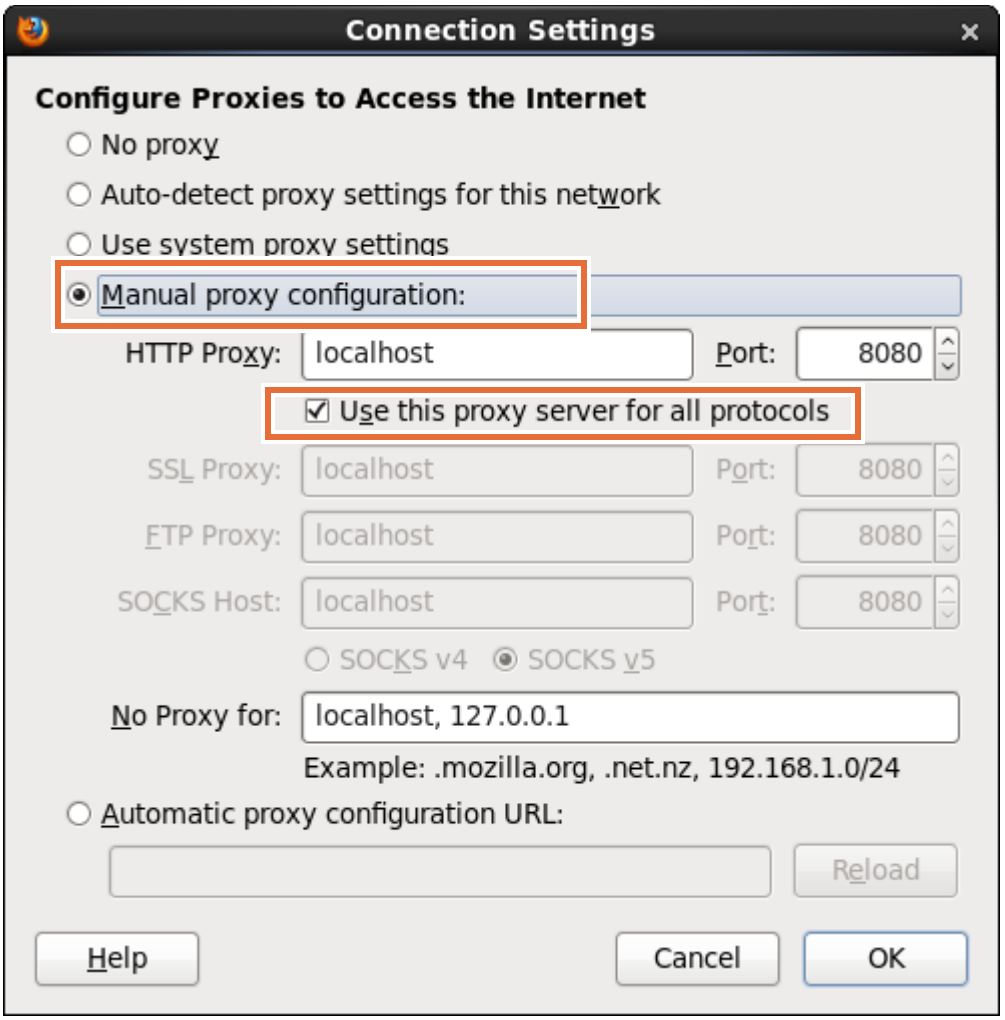
__ b. On the toolbar, click **Edit > Preferences**.

__ c. On the **Advanced** tab, click the **Network** tab, then click **Settings**.

__ d. On the Connects Settings pane, select or enter the following settings.

- Select **Manual proxy configuration**
- HTTP Proxy: **localhost**
- Port: **8080**

- Select **Use this proxy server for all protocols**



__ e. Click **OK**.

__ f. Click **Close** on the Firefox Preference pane.

__ 10. Record navigation of the JMeter website.

__ a. Open a new Firefox browser, and enter the web address: **`http://jmeter.apache.org/index.html`**

__ b. Click several of the links on the JMeter pages. For example, click **User Manual** and some topics from the manual.

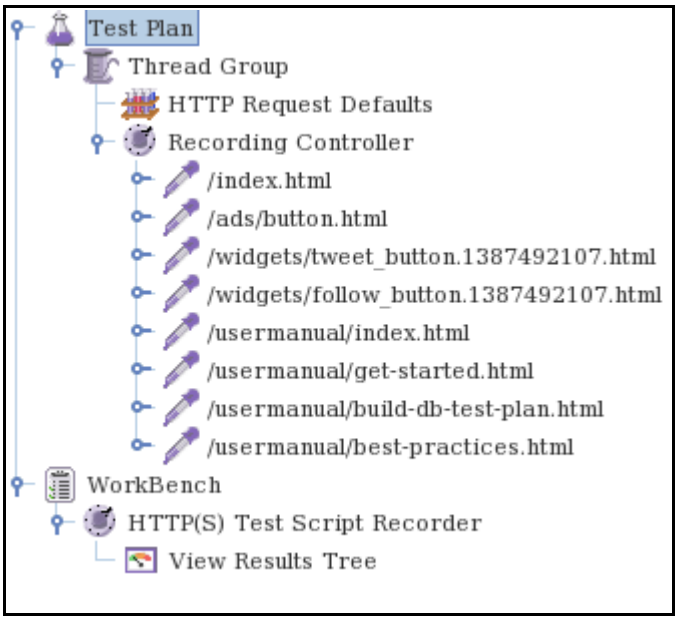
__ c. Close the browser window, and return to JMeter.

__ 11. Stop the Script Recoder.

__ a. From JMeter, select **HTTP(S) Test Script Recorder**, and click the **Stop** at the bottom of the pane.

__ 12. Explore the samplers that were recorded.

__ a. In the JMeter tree, expand Recording Controller. Notice the different samplers that correspond to each html that you navigated to.

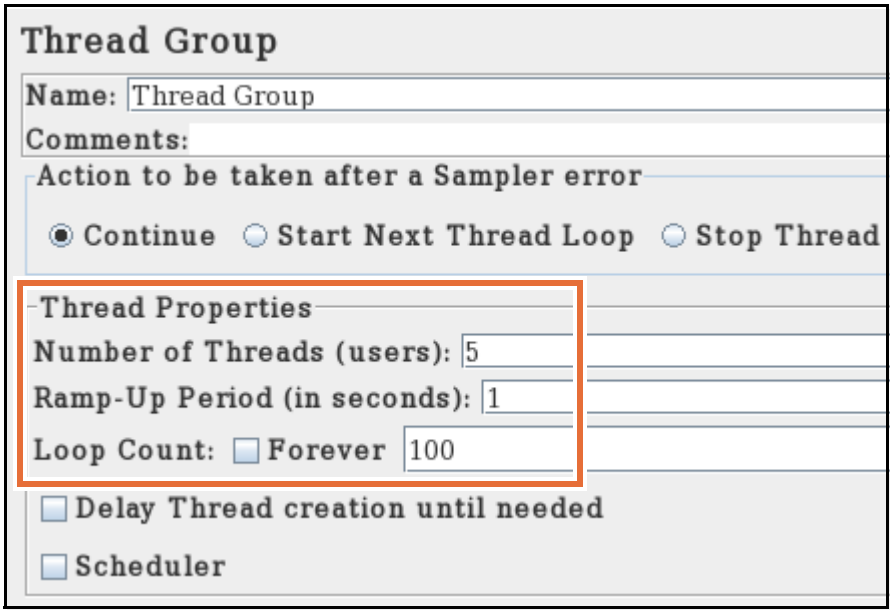


__ 13. Configure a test to run the script.

__ a. Add a Summary Report under Thread Group by right-clicking **Thread Group** and clicking **Add > Listener > Summary Report.**

__ b. Click **Thread Group** and make the following property changes.

- Number of Threads (users): **5**
- Ramp-Up Period: **1**
- Loop Count: **100**



__ 14. Save the test plan.

__ a. From JMeter, click **File > Save Test Plan as**.

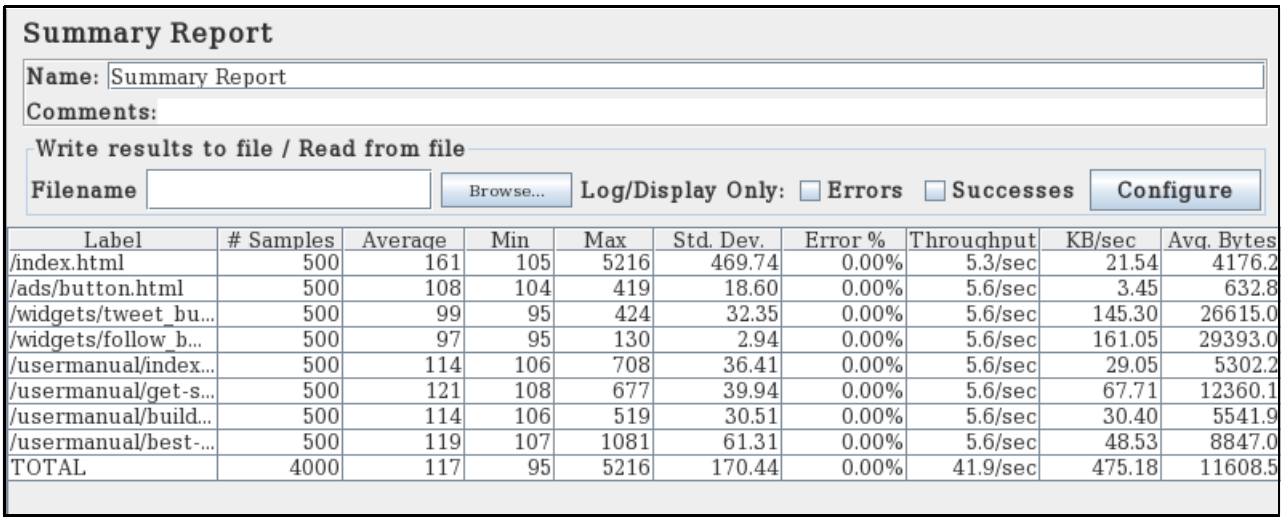
__ b. Navigate to **`/usr/labfiles/Test_Plans`** and save the plans as **`Script_Recorder_test.jmx`**

__ 15. Run the test plan.

__ a. In the JMeter tree, click **Summary Report**.

__ b. Click **Run > Start** or the green arrow to start the test.

__ c. When the test completes, you should see performance results in the Summary Report.



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
/index.html	500	161	105	5216	469.74	0.00%	5.3/sec	21.54	4176.2
/ads/button.html	500	108	104	419	18.60	0.00%	5.6/sec	3.45	632.8
/widgets/tweet_bu...	500	99	95	424	32.35	0.00%	5.6/sec	145.30	26615.0
/widgets/follow_b...	500	97	95	130	2.94	0.00%	5.6/sec	161.05	29393.0
/usermanual/index...	500	114	106	708	36.41	0.00%	5.6/sec	29.05	5302.2
/usermanual/get-s...	500	121	108	677	39.94	0.00%	5.6/sec	67.71	12360.1
/usermanual/build...	500	114	106	519	30.51	0.00%	5.6/sec	30.40	5541.9
/usermanual/best-...	500	119	107	1081	61.31	0.00%	5.6/sec	48.53	8847.0
TOTAL	4000	117	95	5216	170.44	0.00%	41.9/sec	475.18	11608.5

__ 16. Stop JMeter.

__ a. Click **File > Exit**.

__ 17. Unconfigure the browser to use the Test Script Recorder.

__ a. Start the Firefox browser.

__ b. On the toolbar, click **Edit > Preferences**.

__ c. On the **Advanced** tab, click the **Network** tab, then click **Settings**.

__ d. On the Connects Settings pane, select or enter the following settings.

- Select **Use system proxy settings**.

__ e. Click **OK**.

__ f. Click **Close** on the Firefox Preferences pane.

__ g. Close all browsers.

End of exercise

Exercise review and wrap-up

This exercise provided a basic introduction to using Apache JMeter to create test plans that simulate multiple users that access web-based applications. You learned how to start the Apache JMeter, create a simple test plan, run the test plan, and monitor application performance. You also learned how to use JMeter to record a test script by navigating a remote website.

Exercise 3. DayTrader Benchmark installation

What this exercise is about

The purpose of this exercise is to install the Day Trader Benchmark example application.

What you should be able to do

At the end of this exercise, you should be able to:

- Set up the WebSphere Application Server ND environment
- Install and configure the DayTrader application
- Test and explore the DayTrader application

Introduction

The DayTrader Benchmark is an end-to-end web application that simulates an online stock brokerage. DayTrader uses Java EE components such as servlets, JSP files, enterprise beans (EJBs), message-driven beans (MDBs), and Java database connectivity (JDBC) to provide a set of user services. These services include: login/logout, stock quotations, buy, sell, and account details, which are accessed through standards-based HTTP and web services protocols.

The installation and configuration process for DayTrader basically consists of the following steps.

1. Creating the DayTrader database on the chosen database server
2. Configuring the JDBC driver and data source to communicate with the database
3. Creating the JMS/Messaging resources to support JMS queues, topics, and MDBs
4. Installing the DayTrader EAR file

Before proceeding, it is important to identify the database server that is used to host the DayTrader database. The scripts are written to support several database vendors that include DB2, Oracle, Informix, and Derby. Every effort is made to ease the database configuration process. Nevertheless, some familiarity with the database software is required.

Requirements

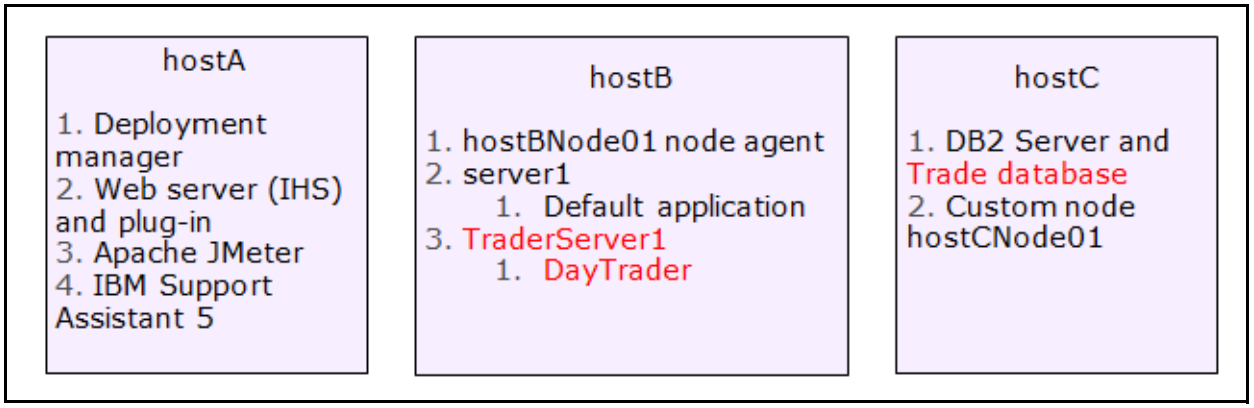
This exercise does not rely on the completion of **Exercise 2: Apache JMeter basics**. However, it is suggested that you do complete that exercise.

You need the compressed file, **`DayTrader3Install.zip`**, for DayTrader 3.0 (Java EE 6). This file is in the **`/usr/labfiles`** directory of your lab image.

Exercise instructions

Preface

- It is important that you complete this exercise successfully as several of the following exercises rely on the DayTrader application.
- Most steps in this exercise are done on **hostA** unless otherwise specified.
- After you complete this exercise, you have the TradeServer1 and the DayTrader application on **hostB** and the DayTrader database on **hostC**.



Note

<profiles_root> = /opt/IBM/WebSphere/AppServer/profiles

Section 1: Start the WebSphere servers and create TradeServer1

If the Deployment Manager and node agent are not already running, start them now. Otherwise, start with Step 3.

__ 1. On **hostA**, start the Deployment Manager.

__ a. Open a terminal window and enter the command:
cd <*profiles_root*>/Dmgr/bin

__ b. Enter the command:
./startManager.sh

__ 2. On **hostB**, start the node agent.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
cd <*profiles_root*>/profile1/bin

__ c. Enter the command:
./startNode.sh

__ 3. Create an application server called **TradeServer1**.

__ a. On **hostA**, start a web browser and enter the web address **`http://localhost:9060/ibm/console`**

__ b. Log in to the administrative console with the user ID: **`wasadmin`** and Password: **`web1sphere`**

__ c. Click **Servers > Server Types > WebSphere application servers**.

__ d. Click **New**.

__ e. Select node: **`hostBNode01`**. This option is the default.

__ f. For the Server name, enter: **`TradeServer1`**

__ g. Click **Next.**

__ h. Accept the default template and click **Next.**

__ i. Make sure that **Generate Unique Ports** is selected and click **Next.**

__ j. Read the Summary and click **Finish**.

__ k. In the messages pane, click the **Preferences** link.

__ l. Select the **Synchronize changes with Nodes** check box.

__ m. Click **Apply**. This console preference is now set for the rest of the lab exercises.

__ n. **Save** the changes.

__ o. Wait for the node to synchronize and click **OK**.

__ 4. Map TradeServer1's web container port 9081 to the default_host virtual host.

__ a. From the administrative console, click **Environment > Virtual hosts > default_host**.

__ b. Click **Host Aliases**.

__ c. Click **New**.

__ d. Enter Port **`9081`** and click **OK**.

__ e. **Save** the changes.

__ f. Wait for the node to synchronize and click **OK.**

Section 2: Install the DayTrader application

In this section, you create the DayTrader database and deploy the DayTrader application.

__ 1. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ 2. Create the **tradedb** database.

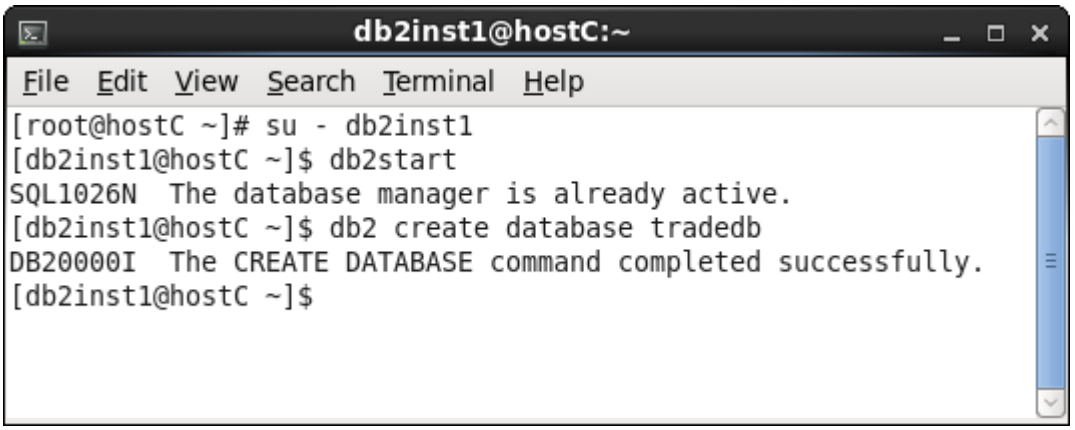
__ a. Open a terminal window and enter the command: **`su - db2inst1`**

Note

Your VMware image might be configured to have DB2 started automatically when the system starts up. Verify that DB2 is started with your instructor. If it is started, you can skip Step **2 b**.

__ b. Start DB2 with the command: **db2start**

__ c. Create the tradedb database: **db2 create database tradedb**



__ d. Wait a few minutes for the database to be created successfully.

__ e. Log out as **db2inst1** by typing: **logout**

__ 3. Start the DayTrader installation and configuration scripts.

__ a. On **hostA**, start a new terminal window and enter the command:
cd /usr/labfiles/DayTrader3-EE6

__ b. On one line enter the command:

/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin/wsadmin.sh -f daytrader_singleServer.py -user wasadmin -password web1sphere

__ 4. Specify the script information. The Jython script asks a series of questions to configure the necessary JDBC and JMS resources. As you progress through the interactive steps, enter the following information and press the **Enter** key.

Note

The first time that you run a Jython script several JAR files must be processed. So you must wait a few minutes before the script runs.

__ a. Global security is enabled: **true**

__ b. Select the node: **hostBNode01**

__ c. Select the server: **TradeServer1**

__ d. Select the JDBC provider type: **DB2 Universal**

__ e. Select the EJB deployment target: **DB2UDB_V82**

__ f. Enter the location of JDBC driver (JAR) files:
/home/db2inst1/sqllib/java/db2jcc.jar;/home/db2inst1/sqllib/java/db2jcc_license_cu.jar

__ g. Enter the database name: **tradedb**

__ h. Enter the database host name: **hostC**

__ i. Enter the database port number: **50000**

__ j. Enter the database user name: **db2inst1**

__ k. Enter the database password: **was1edu**

__ l. Use file store for JMS Provider: **true**

__ m. ME file store location: **default**

__ n. Enter a valid administrative user name: **wasadmin**

__ o. Enter a valid administrative password: **web1sphere**

__ 5. The script begins creating all of the application resources and installs the DayTrader application.

__ a. Scroll through the terminal window and verify that you see no error messages or warnings.

__ b. Verify that the script completes with the message:

DayTrader Installation Completed!!!

Troubleshooting

DayTrader installation failure

If the DayTrader installation fails, or you need to reinstall it for any reason, follow these steps to uninstall it and clean up the environment.

- Uninstall DayTrader by running the script with the `-uninstall` option.

```
/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin/wsadmin.sh -f
daytrader_singleServer.py -uninstall -user wasadmin -password
web1sphere
```

- Clean up the environment by running the script with the `-cleanup` option.

```
/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin/wsadmin.sh -f
daytrader_singleServer.py -cleanup -user wasadmin -password
web1sphere
```

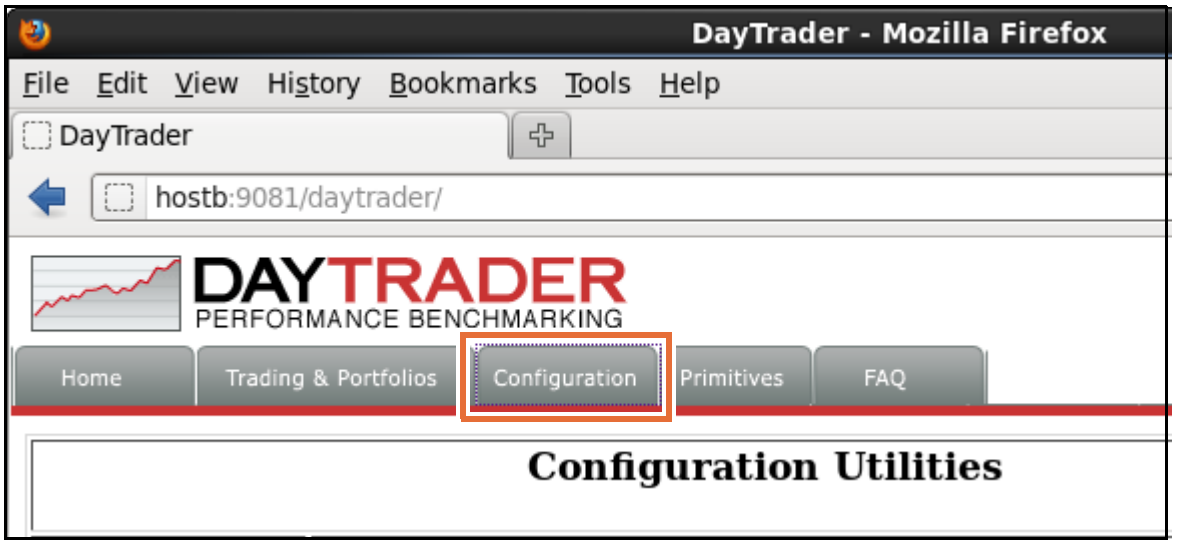
__ 6. Start **TradeServer1** from the administrative console.

__ 7. Examine the server's **SystemOut** and **SystemErr** logs to see whether any error or warning messages are recorded.

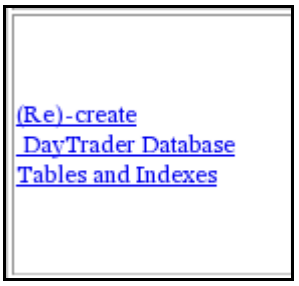
__ 8. After the DayTrader application is installed and started, it is accessed by going to the following web address: **http://hostb:9081/daytrader**
However, before accessing any of the trading application functions, the required database tables must be created and populated with an initial set of accounts, stock quotations, and holdings. Follow the simple steps that are outlined next to configure the database tables and populate them with data.

__ a. Open a web browser and enter the web address: **http://hostb:9081/daytrader**

__ b. Click the **Configuration** tab.



__ c. Scroll down and click the **(Re)-create Day Trader Database Tables and Indexes** link.



Information

(Re)-create DayTrader Database Tables and Indexes

This link is used to (a) initially create or (b) drop and re-create the DayTrader tables. A DayTrader database should exist before doing this action. The existing DayTrader tables, if any, are dropped, and then new tables and indexes are created.

The initial population size consists of 15000 accounts and 10000 stock quotations. These values can be updated by using the “Configure DayTrader runtime parameters” link on the “Configuration” tab.

__ d. Wait for the database tables and indexes to be created. You should see the following messages when the process is complete.

TradeBuildDB: ** Dropping and Recreating the DayTrader tables... ******

TradeBuildDB: ** DayTrader tables successfully created! ******

__ e. Close the browser and open a new browser.

__ f. From the administrative console, restart the application **DayTrader3-EE6**.

__ g. Open a web browser and enter the web address **`http://hostb:9081/daytrader`**

__ h. Click the **Configuration** tab.

__ i. Click the **(Re)-populate DayTrader Database** link.

__ j. Wait several minutes for the operation to complete. You should see messages similar to the following message when the process is complete.

TradeBuildDB: ** Creating 10000 Quotes ******

.....s:0 -

.....s:10.....s:20.....s:30.....s:40.....s:50.....s:60.....s:70.....s:80.....s:90.....s:100 -

.

.

****** Registering 15000 Users ******

Account# 1 userID=uid:0 has 8 holdings.

Account# 51 userID=uid:50 has 3 holdings.

.

.

Account# 451 userID=uid:450 has 10 holdings.

__ k. You also see a detailed description of the current DayTrader configuration. Read the description and close the browser when you are finished.

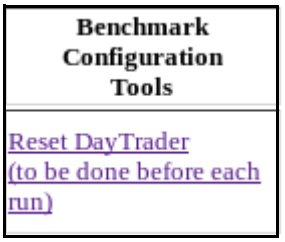
Section 3: Explore the DayTrader application

In this section, you explore the DayTrader application and get familiar with some of the utilities that it provides.

__ 1. Open a web browser and enter the web address: **`http://hostb:9081/daytrader`**

__ 2. On the DayTrader welcome page, click the **Configuration** tab.

__ 3. Click the link **Reset DayTrader**.



Information

Reset DayTrader link

Reset the DayTrader run time to a clean starting point by logging off all users, removing new registrations, and other general cleanup. For consistent results, this URL is run **before each** DayTrader test.

__ 4. You see the following information.

DayTrader Scenario Runtime Statistics

Trade Reset completed successfully

Modify runtime configuration

Benchmark runtime configuration summary	Value
Run-Time Mode	**Full EJB3**
Order-Processing Mode	**Synchronous**
Scenario Workload Mix	**Standard**
Web Interface	**JSP**
Active Traders / Trade User population	**15000 / 15000**
Active Stocks / Trade Stock population	**10000 / 10000**

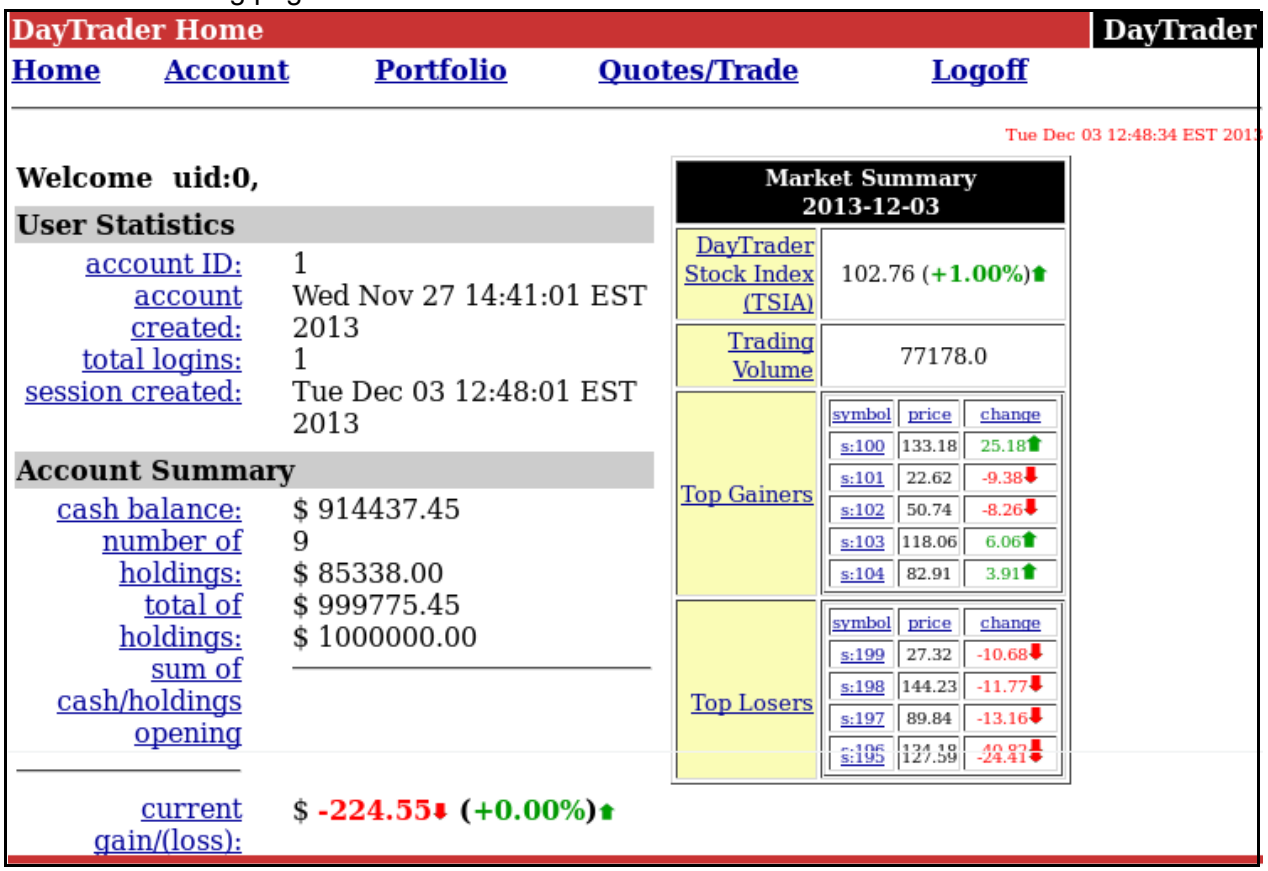
Benchmark scenario verification

Run Statistic	Scenario verification test	Expected Value	Actual Value	Pass/Fail
Active Stocks	Active stocks should generally equal the db population of stocks	10000	**10000**	Pass
Active Traders	Active traders should generally equal the db population of traders	15000	**15000**	Pass

__ 5. Try logging in and trading some stock.

__ a. Click the **Trading & Portfolios** tab.

__ b. On the Login page, the user name (uid:0) and password (***) fields are prefilled. Click **Log in**, then click **Home**. You see account information similar to what is shown on the following page.



__ c. Click the links at the top of the page: **Home**, **Account**, **Portfolio**, and **Quotes/Trade**. Explore the information that is provided on each page.

__ d. Buy stock by clicking **Quotes/Trade**. The Quotes page shows several rows of stock listings. In the trade column, click **buy** for any stock you like. You see a **New Order** confirmation of your stock purchase.

__ e. Sell stock by clicking the **Portfolio** link. You see a table of all of the stocks in the portfolio. In the trade column, click **sell** for any stock you want to sell. You see a confirmation of your sell order.

__ f. When you are done exploring, click **Logoff**.

__ g. Close the browser.

__ 6. Map the DayTrader's web module to the **web server**.

__ a. From the administrative console, click **Applications > Application Types > WebSphere enterprise applications**.

__ b. Click **DayTrader3-EE6**.

__ c. Click **Manage Modules**.

__ d. Ensure that only the **DayTrader Web** check box is selected.

__ e. In the Clusters and servers list, press **Ctrl** and select both **WebSphere:Cell=hostACell01,node=ihsnode, server=webserver1** and **WebSphere:Cell=hostACell01,node=hostBNode01,server=TradeServer1**

__ f. Click **Apply**.

__ g. Verify that the mapping is correct.

Module	URI	Module Type	Server
DayTrader Enterprise Bean Definitions	dt-ejb.jar,META-INF/ejb-jar.xml	EJB Module	WebSphere:cell=hostACell01,node=hostBNode01,server=TradeServer1
DayTrader Web	web.war,WEB-INF/web.xml	Web Module	WebSphere:cell=hostACell01,node=hostBNode01,server=TradeServer1 WebSphere:cell=hostACell01,node=ihsnode,server=webserver1
Rest.war	Rest.war,WEB-INF/web.xml	Web Module	WebSphere:cell=hostACell01,node=hostBNode01,server=TradeServer1

__ h. Click **OK**.

__ i. **Save** the changes.

__ j. Wait for the node to synchronize and click **OK**.

__ 7. If not already started, start the web server on **hostA**.

__ a. In a terminal window, enter the command:
`cd /opt/IBM/HTTPServer/bin`

__ b. Enter the following commands:

`./apachectl start`

`./adminctl start`

__ 8. Verify that the DayTrader can be accessed with the web server and web container port.

__ a. Open a new web browser and enter the address: **`http://hostb:9081/daytrader`**

__ b. Open a new web browser and enter the address: **`http://hosta/daytrader`**

__ c. Close the DayTrader browsers.

Information

Recall that the web server is running on hostA and that TradeServer1 is running on hostB, so the URLs must point to the correct host for the request to get routed correctly.

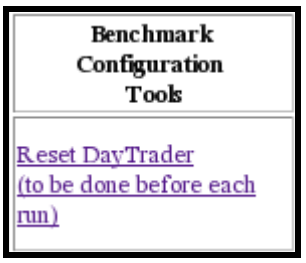
Section 4: Run the DayTrader test scenario

The Test DayTrader Scenario is going to be run by using Apache JMeter in subsequent exercises. This section gives an idea of what application functions the scenario does.

__ 1. Open a new web browser and enter the address: **`http://hosta/daytrader`**

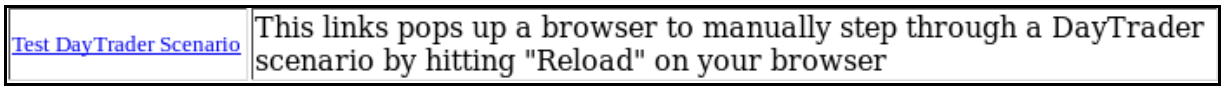
__ 2. Click the **Configuration** tab.

__ 3. Under Benchmark Configuration Tools, click the **Reset DayTrader** link.

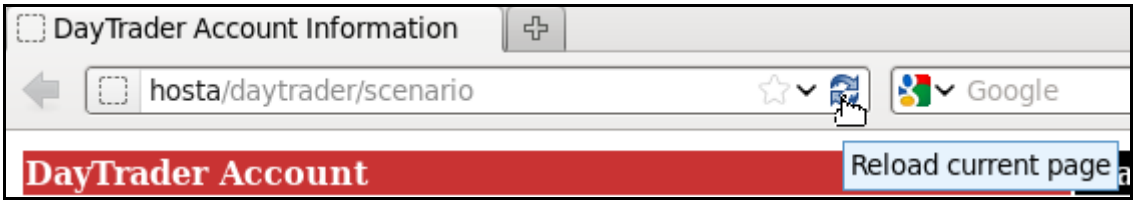


This function resets the DayTrader run time to a clean starting point by logging off all users, removing new registrations, and other general cleanup. For consistent results, this URL should be run before each Trade run.

__ 4. Click the **Configuration** tab again. Scroll down and click the **Test DayTrader Scenario** link. This link pops up a browser to manually step through a DayTrader scenario by pressing the Reload icon on your browser. Initially you see User Statistics and an Account Summary for a randomly selected user ID.



__ 5. Click your browser's **Reload** icon repeatedly to step through the DayTrader scenario.



__ 6. Wait for each page to load. As you step through the scenario, you see a list of Quotes and a Buy Order. Eventually you return to the DayTrader Login page.

__ 7. Close the browser.

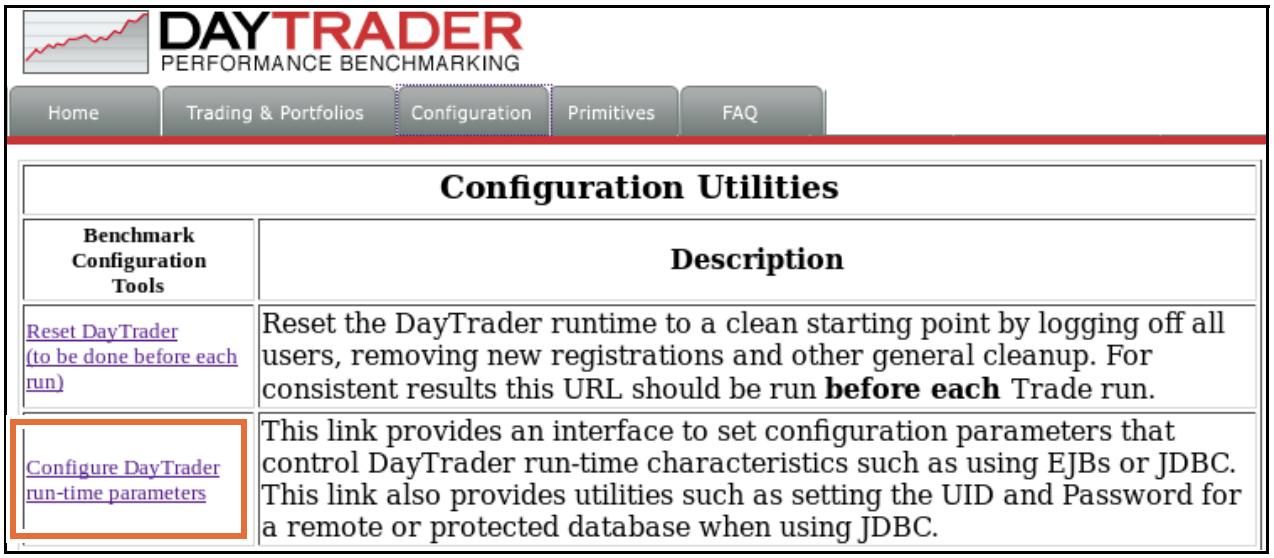
__ 8. If you choose not to proceed to the next section, stop **TradeServer1**.

Section 5: Optional: Explore some of the DayTrader configuration options

The DayTrader application provides a number of configuration options. In this section, you explore some of these options.

__ 1. Open a new web browser and enter the address: **`http://hostb:9081/daytrader`**

__ 2. Click the **Configuration** tab and click the link **Configure DayTrader run-time parameters**.



__ 3. Runtime Mode determines server implementation of the TradeServices to use in the DayTrader application Enterprise beans, which include Session, Entity, and Message beans or Direct mode, which uses direct database and JMS access. Examine the different **runtime Modes** by clicking the **DayTrader FAQ** link.

- Full EJB3
- Direct (JDBC)
- Session (EJB3) to Direct

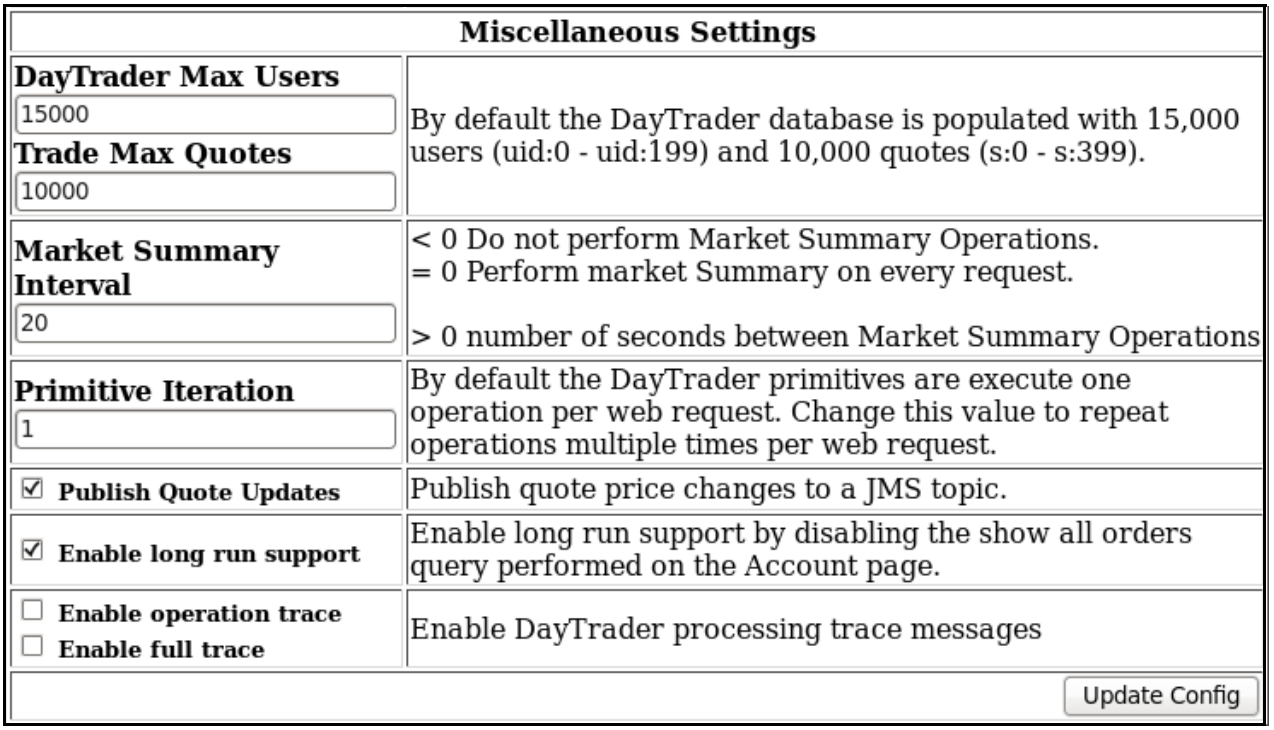
__ 4. Order Processing Mode determines the mode for completing stock purchase and sell operations. Synchronous mode completes the order immediately. Asychronous_2-Phase does a 2-phase commit over the EJB Entity/DB and MDB/JMS transactions. Examine the **Order Processing Modes** by clicking the **DayTrader FAQ** link.

- Synchronous
- Asynchronous_2-Phase

__ 5. The Scenario Workload Mix setting determines the runtime workload mix of DayTrader operations when driving the benchmark through TradeScenarioServlet. Examine the **Scenario Workload Mix** by clicking the **DayTrader FAQ** link.

- Standard
- High-Volume

__ 6. Explore the **Miscellaneous Settings** to learn about other configuration options.



Miscellaneous Settings	
DayTrader Max Users 15000 **Trade Max Quotes** 10000	By default the DayTrader database is populated with 15,000 users (uid:0 - uid:199) and 10,000 quotes (s:0 - s:399).
Market Summary Interval 20	< 0 Do not perform Market Summary Operations. = 0 Perform market Summary on every request. > 0 number of seconds between Market Summary Operations
Primitive Iteration 1	By default the DayTrader primitives are execute one operation per web request. Change this value to repeat operations multiple times per web request.
☑ **Publish Quote Updates**	Publish quote price changes to a JMS topic.
☑ **Enable long run support**	Enable long run support by disabling the show all orders query performed on the Account page.
☐ **Enable operation trace** ☐ **Enable full trace**	Enable DayTrader processing trace messages
	Update Config

The Market Summary Interval is set by default to 20 seconds. Therefore, every 20 seconds the market summary data is updated and displayed in the browser.

Market Summary
2014-01-19

DayTrader Stock Index (TSIA)	102.76 (+1.00%)
Trading Volume	77178.0

Top Gainers

symbol	price	change
s:100	133.18	25.18
s:101	22.62	-9.38
s:102	50.74	-8.26
s:103	118.06	6.06
s:104	82.91	3.91

Top Losers

symbol	price	change
s:199	27.32	-10.68
s:198	144.23	-11.77
s:197	89.84	-13.16
s:196	134.18	-40.82
s:195	127.59	-24.41

This summary contains a list of the top five gaining and losing stocks and also the current stock index and trading volume. This activity requires the execution of several database lookups and can delay the loading of the user's home page. With servlet caching, the **marketSummary.jsp** can be cached, virtually eliminating these expensive database queries to improve the response time for the user home page. In a later exercise, you are going to learn how to enable servlet caching for the market summary.

__ 7. When you are done exploring the DayTrader application, stop **TradeServer1.**

End of exercise

Exercise review and wrap-up

In this exercise, you learned how to install the DayTrader Benchmark sample application. Preliminary setup of the environment required you to create the DB2 database tradedb; then you ran an interactive Jython script to install and configure the DayTrader application.

After the application is installed, you learned how to work with the DayTrader, do other configuration tasks and log in as a user to trade stocks.

Exercise 4. Using Apache JMeter to load test DayTrader

What this exercise is about

In this exercise, you lean how to use Apache JMeter to run a ramp up test on the DayTrader application. At the end of this exercise, you have a test plan that you can use as a baseline for future performance tuning exercises.

What you should be able to do

At the end of this exercise, you should be able to:

- Tune and back up the DayTrader database
- Run a ramp-up test on DayTrader to find the saturation point
- Monitor CPU usage on the application server host and the database server host
- Establish a baseline of performance for the DayTrader application server

Introduction

To begin applying performance tuning advice to an application server, it is necessary to ensure consistent performance results and have a performance baseline for comparison. The DayTrader application uses the tradedb database, which is running on a dedicated DB2 server host in the lab environment. You are going to run some basic DB2 database maintenance commands to ensure acceptable performance of the database server. You then back up the database so that it can be restored for future tests.

To establish a baseline of performance for the DayTrader application server, you run a ramp up test and increase the number of concurrent users until the application server host is using 100% CPU. The number of concurrent users that drives CPU to 100% becomes your baseline test plan, and the measure response time and throughput becomes your baseline of performance.

Requirements

This exercise depends on the successful completion of **Exercise 3: DayTrader Benchmark installation**. The DayTrader application must be installed on hostA, and the DayTrader database must be created on hostC.

Exercise instructions

Preface

- Most steps in this exercise are done on **hostA** unless otherwise specified.
- It is important that you complete this exercise successfully as several of the following exercises rely on the database backup and the test plan that you create.

Note

<profiles_root> = /opt/IBM/WebSphere/AppServer/profiles

Section 1: Tune the DayTrader database and back it up

This section shows you the basic DB2 database performance commands that you must run to maintain the tradedb database so that you can achieve consistent performance results. After the database is tuned, you back it up so that you can restore it to a consistent state for future testing.

__ 1. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ 2. Enter the command: **`su - db2inst1`**

__ 3. Update statistics on all tables in the tradedb database by entering the db2 commands ***or*** running the script, do not do both.

__ a. Following are the db2 commands for your reference. Go to **step b**, and run the script.

```
db2 connect to tradedb

db2 reorgchk update statistics on table all

db2 connect reset

db2rbind tradedb -l rbind.log all
```

__ b. Run the script: **`/usr/labfiles/db2scripts/db2updatestats.sh`**

__ c. Wait until you see the messages:

```
DB20000I  The SQL command completed successfully.

Rebind done successfully for database 'TRADEDB'.
```

__ 4. Gather more statistics on specific tables by entering the db2 commands ***or*** running the script. Do not do both.

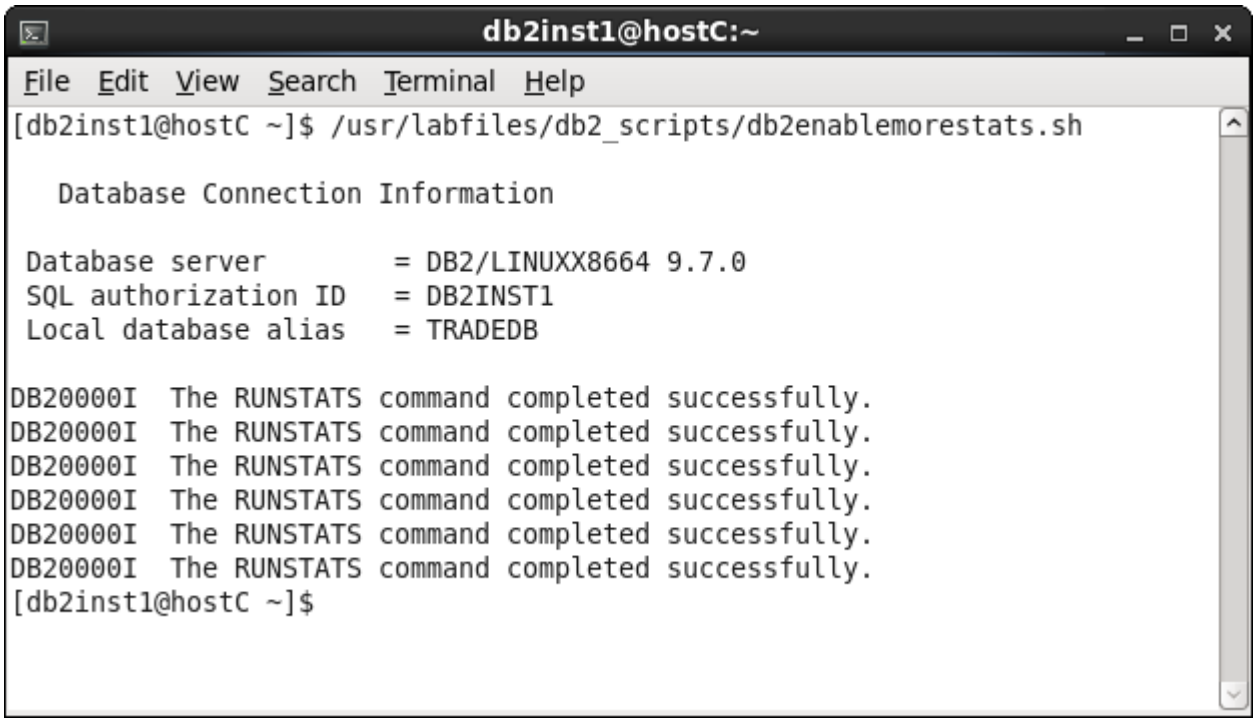
__ a. Following are the db2 commands for your reference. Go to **step b**, and run the script.

```
db2 connect to tradedb

db2 runstats on table db2inst1.accountejb with distribution and detailed
indexes all

db2 runstats on table db2inst1.accountprofileejb with distribution and
detailed indexes all
```

```
db2 runstats on table db2inst1.holdingejb with distribution and detailed
indexes all

db2 runstats on table db2inst1.keygenejb with distribution and detailed
indexes all

db2 runstats on table db2inst1.orderejb with distribution and detailed
indexes all

db2 runstats on table db2inst1.quoteejb with distribution and detailed
indexes all
```

__ b. Run the script: **/usr/labfiles/db2scripts/db2enablemorestats.sh**

__ c. Wait until you see the messages:



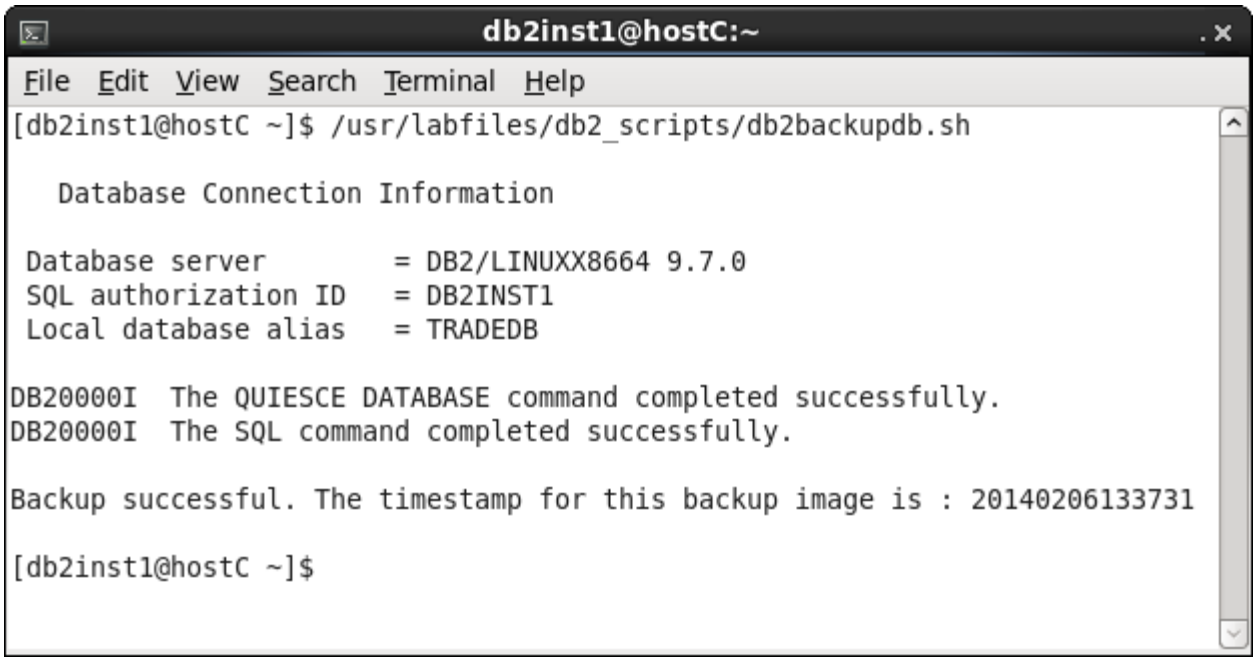
__ 5. Back up the tradedb database by running the following db2 commands ***or*** run the script. Do not do both.

__ a. Following are the db2 commands for your reference. Go to **step b**, and run the script.

```
db2 connect to tradedb

db2 quiesce database immediate force connections

db2 connect reset

db2 backup database tradedb to "/home/db2inst1" with 2 buffers buffer 1024
parallelism 1 without prompting
```

__ b. Run the script: **/usr/labfiles/db2scripts/db2backupdb.sh**

__ c. Wait until you see similar messages:



__ 6. When the backup completes, you see the message:

Backup successful. The timestamp for this backup image is : <time_stamp>

Record your timestamp here _____.

__ 7. Enter the following commands to unquiesce the database ***or*** run the script. Do not do both.

__ a. Following are the db2 commands for your reference. Go to **step b**, and run the script.

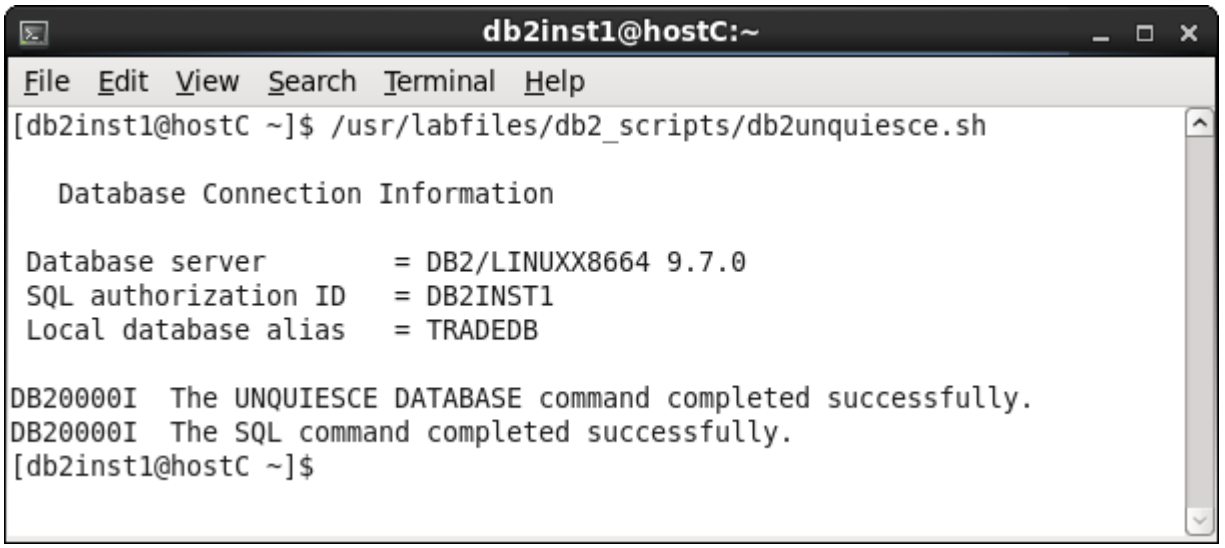
db2 connect to tradedb

db2 unquiesce database

db2 connect reset

__ b. Run the script: **/usr/labfiles/db2scripts/db2unquiesce.sh**

__ c. Wait until you see the messages:

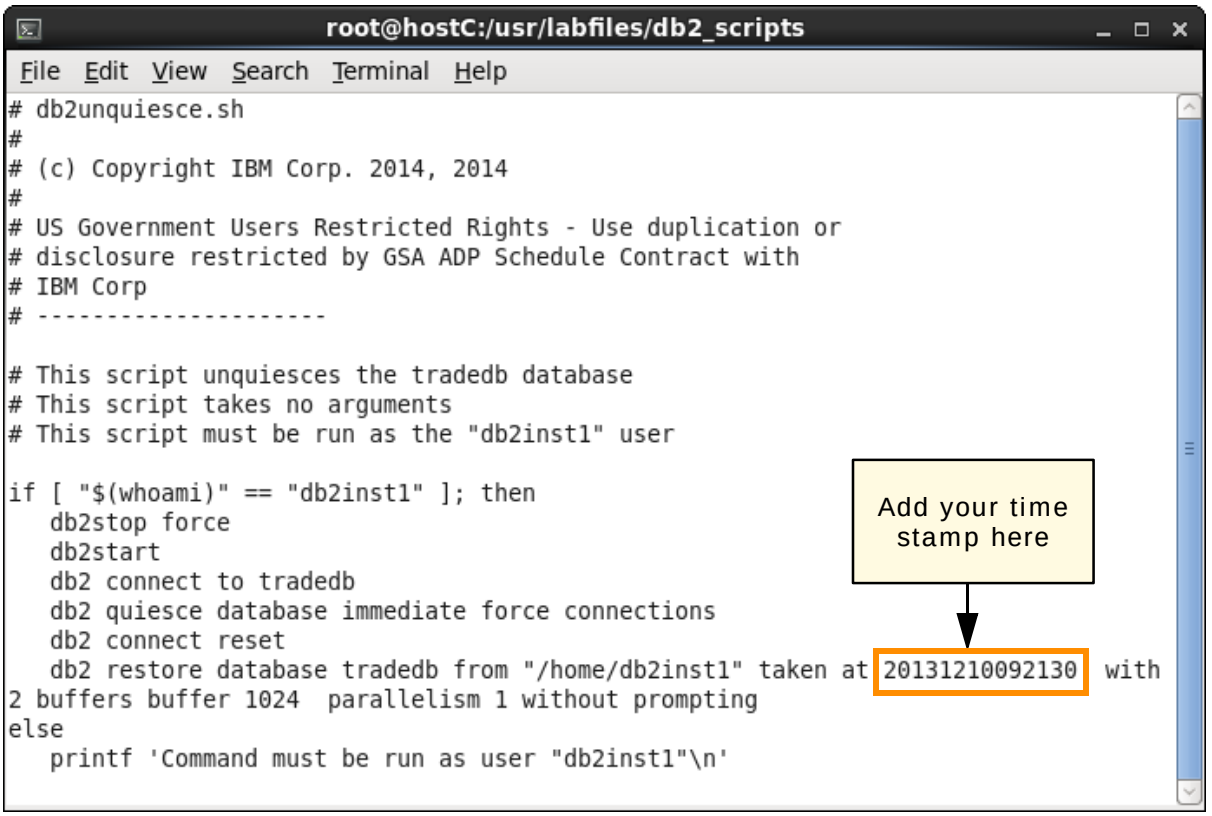


__ 8. Edit the Trade database restore script.

__ a. On **hostC**, use a text editor such as **vi** to open the script, **`/usr/labfiles/db2scripts/db2restore.sh`**

__ b. Replace **<time_stamp>** with the value that you recorded in the previous step.

__ c. For example:



__ d. Save the modified script.

Section 2: Run the ramp up test on DayTrader to find the saturation point

In this section, you restore the tradedb database and use Apache JMeter to begin running the ramp up test on the DayTrader application server.

__ 1. On **hostA**, start the Deployment Manager.

__ a. Open a terminal window and enter the command:
`cd <profiles_root>/Dmgr/bin`

__ b. Enter the command:
`./startManager.sh`

__ 2. On **hostB**, start the node agent.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
`cd <profiles_root>/profile1/bin`

__ c. Enter the command:
`./startNode.sh`

__ 3. In the next several steps, you reset the TradeServer1 to a default application server configuration.

__ 4. Log in to the administrative console as user **`wasadmin`** with password **`web1sphere`**.

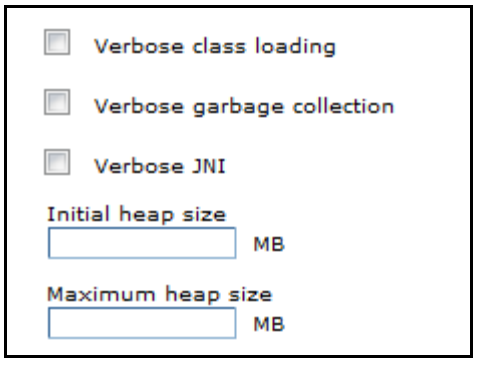
__ 5. Set the initial and maximum heap sizes to their default values.

__ a. Click **Server Type > WebSphere application servers > TradeServer1**.

__ b. On the configuration tab for TradeServer1, click **Java and Process Management > Process definition**.

__ c. Under **Additional properties**, click **Java Virtual Machine**.

__ d. Click the **Configuration** tab, and make sure the values for **initial heap size** and **maximum heap size** are cleared. This action sets the initial heap size to the default of 50 MB and the maximum heap size to the default of 256 MB.



__ e. Click **OK**.

__ 6. Set the Trade data source to its default property values.

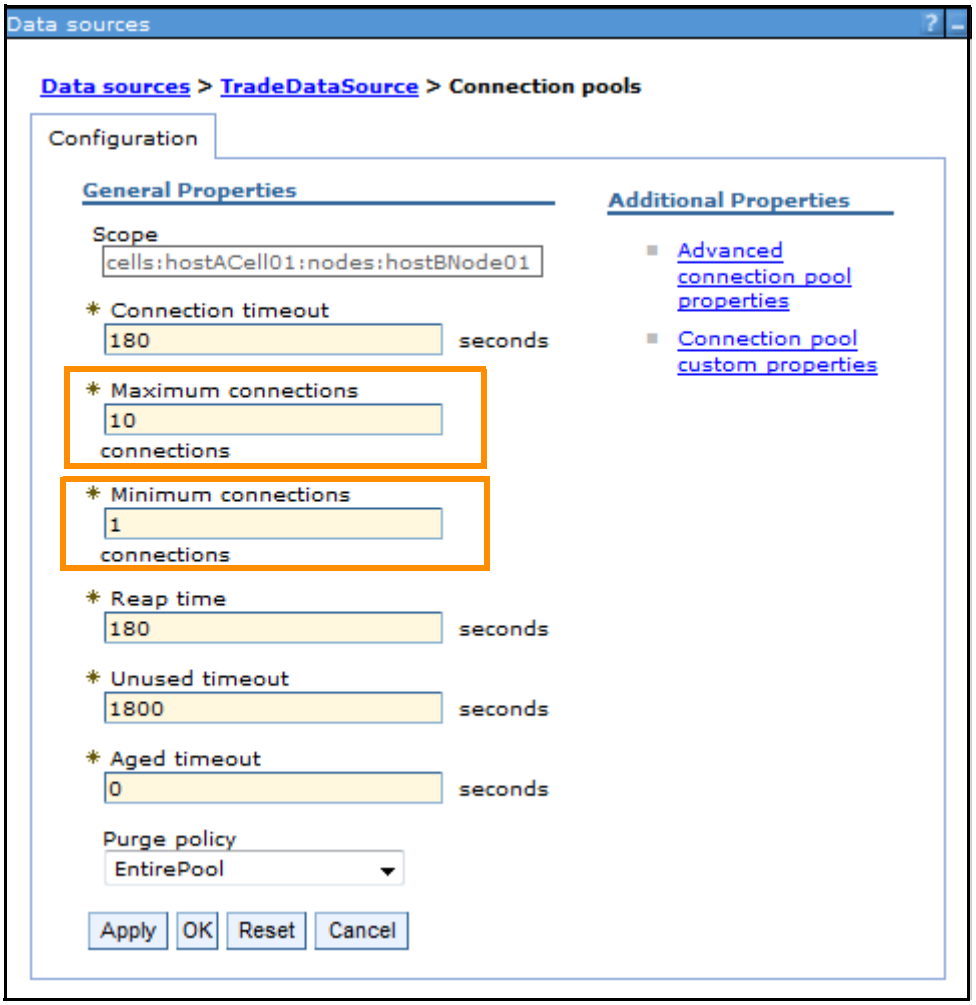
__ a. Click **Resources > JDBC > Data sources**.

__ b. Verify that the scope is set to **All Scopes**.

__ c. Scroll down in the table of data sources and click **TradeDataSource**.

__ d. Under Additional Properties, click **Connection Pool Properties**. The connection pool properties are displayed.

__ e. Set the **maximum connections** to **10**, and set the **minimum connections** to **1**.



__ f. Click **OK**.

__ g. Click **Save** to save the configuration.

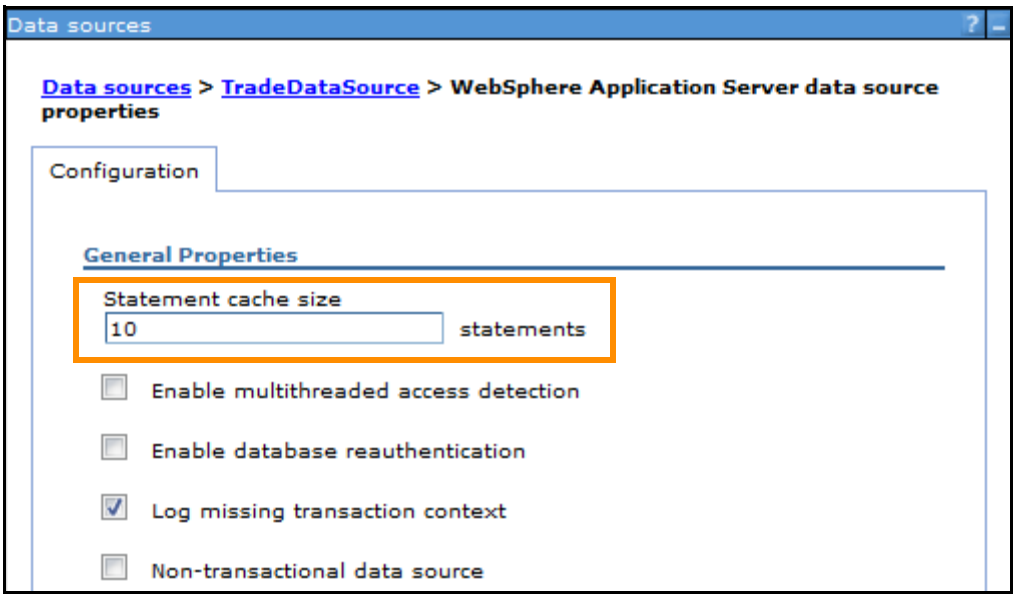
__ h. Wait for the node to synchronize, and click **OK**.

__ i. Click **Resources > JDBC > Data sources**.

__ j. Verify that the scope is set to **All Scopes**.

__ k. Scroll down in the table of data sources and click **TradeDataSource**.

__ l. Under Additional Properties, click **WebSphere Application Server data source properties.** Set the **Statement cache size** to **10.**



__ m. Click **OK.**

__ n. Click **Save** to save the configuration.

__ o. Wait for the node to synchronize, and click **OK**.

__ 7. Restore the tradedb database.

__ a. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ b. Open a terminal window and enter the command:

```
su - db2inst1
```

__ c. Run the following script to restore the tradedb database.

```
/usr/labfiles/db2scripts/db2restore.sh
```

__ d. Run the following script to unquiesce the database.

```
/usr/labfiles/db2scripts/db2unquiesce.sh
```

Note

DB2 warning message: `SQL2540W Restore is successful, however a warning "2539" was encountered during Database Restore while processing in No Interrupt mode.`

This message is nothing to worry about. It is a warning that you are overwriting an existing database. If the restore command did not specify WITHOUT PROMPTING, you would see the message in the terminal window, and you must answer YES to continue.

__ 8. Restart the **node agent** and start **TradeServer1** so that the configuration changes take effect.

__ a. From the administrative console, select **node agent** on **hostBNode01** and click **Restart**.

__ b. Wait for the node agent to start.

__ c. From the administrative console, select **TradeServer1** and click **Start**.

__ 9. Start the Apache JMeter and open a test plan.

__ a. On **hostA**, open a terminal window and enter the command:
`cd /opt/apache-jmeter/bin`

__ b. Enter the command:
`./jmeter.sh &`

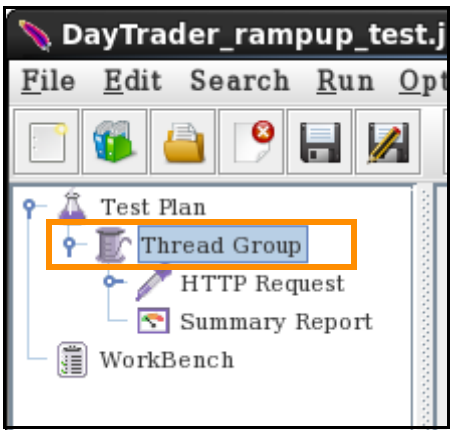
__ c. Load a JMeter script by clicking **File > Open**.

__ d. Navigate to **`/usr/labfiles/Test_Plans`** and select **`DayTrader_rampup_test.jmx`**.

__ e. Click **Open**, and wait for the test plan to load into JMeter.

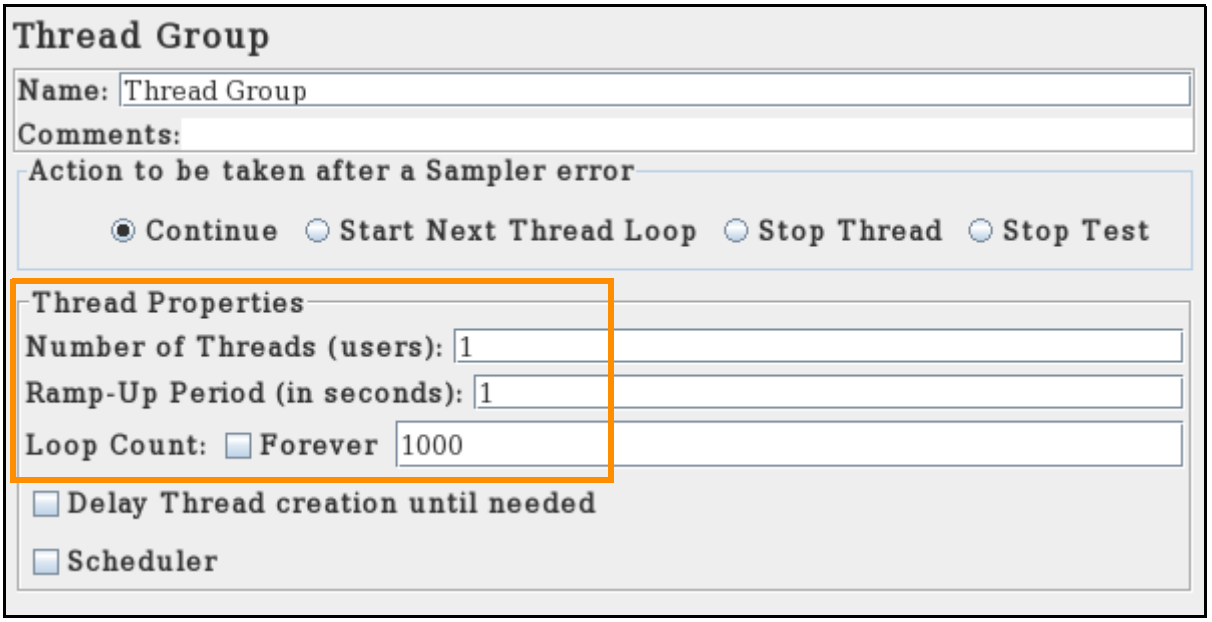
__ 10. Configure the **DayTrader_rampup_test**.

__ a. Click **Thread Group**.



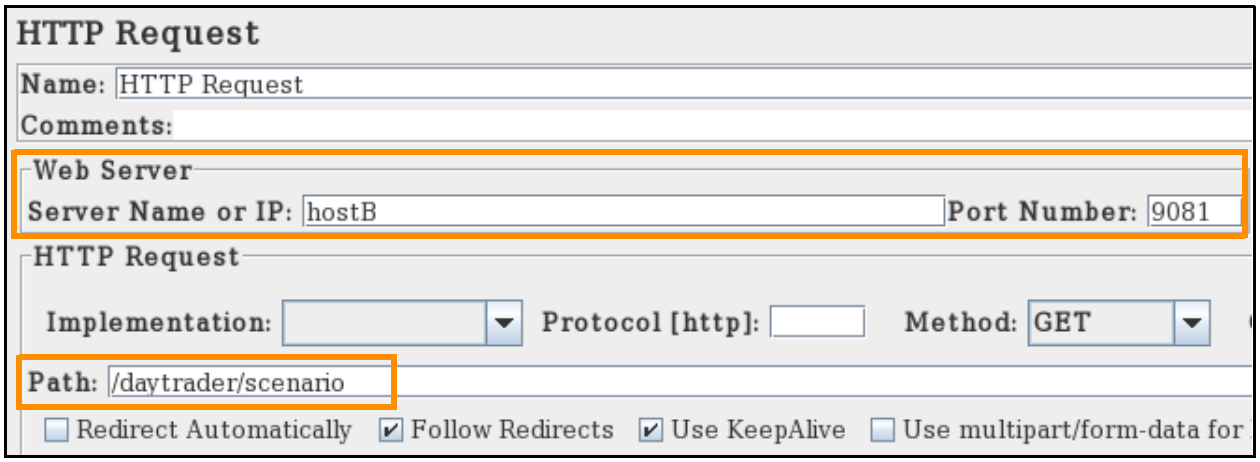
__ b. Configure the Thread Group with the following properties.

- Set **Number of Threads** (users) to **1**.
- Set **Ramp-up Period** (in seconds) to **1**.
- Set **Loop Count** to **1000**.



__ c. Click **HTTP Request** and set the following parameters.

- Server Name = **hostB**
- Port Number = **9081**
- Path = **/daytrader/scenario**



__ d. Save the test plan by clicking **File > Save**.

__ 11. Prepare to load test the DayTrader.

__ a. Start a web browser, and go to **`http://hostB:9081/daytrader`**

__ b. Click the **Configuration** tab, and click the **Reset DayTrader** link. Keep this browser open because you are going to reset the DayTrader for each run of the load test.

__ c. Verify that the reset completes successfully.

DayTrader Scenario Runtime Statistics

Trade Reset completed successfully

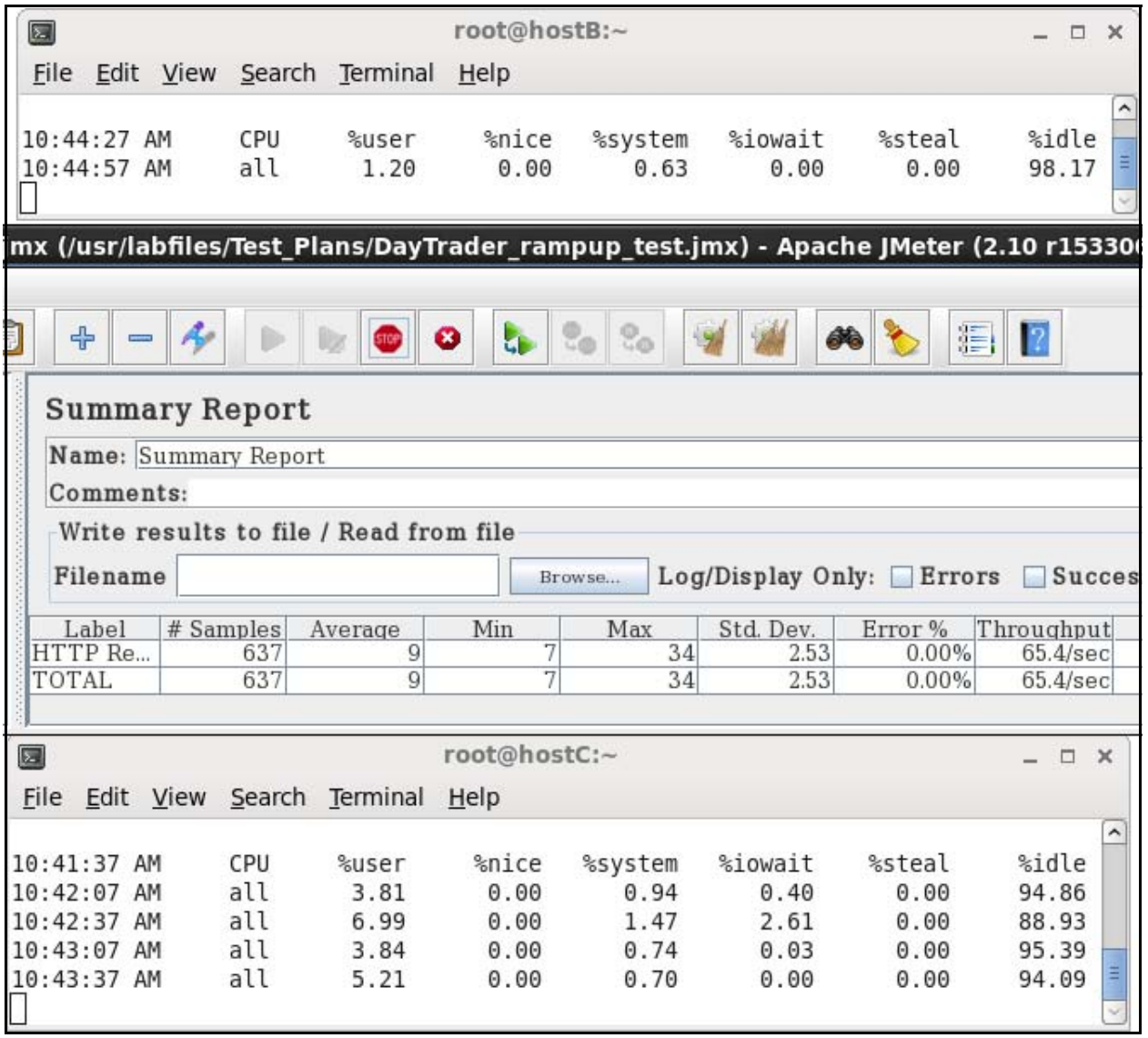
Modify runtime configuration

Benchmark runtime configuration summary	Value
Run-Time Mode	**Full EJB3**
Order-Processing Mode	**Synchronous**
Scenario Workload Mix	**Standard**
Web Interface	**JSP**
Active Traders / Trade User population	**15000 / 15000**
Active Stocks / Trade Stock population	**10000 / 10000**
Benchmark scenario verification	

__ d. In Apache JMeter, click **Summary Report**.

__ e. Open an ssh terminal on **hostB** and **hostC**. Type `sar 5` in each terminal window.

__ f. Arrange JMeter and the two terminal windows so that you can see the CPU usage for hostB (TradeServer1) and hostC (DB2)



__ 12. Run the test plan.

__ a. In JMeter, click **Run > Start** (or click the green arrow) to start the test.

__ b. As the test runs, you need to look at the following measurements.

- sar terminal for hostB CPU usage (user and system)
- sar terminal for hostC CPU usage (user and system)
- JMeter Summary Report: Average (response time in ms)
- JMeter Summary Report: Throughput (requests per second)

__ c. The test stops when the number of samples reaches 1000 (No. of Threads) X (Loop Count). Before the test stops, record the CPU usage (us, sy) for hostB and hostC.

__ d. After the test stops, record the **Average response time** and **Throughput**.

Information

Little’s Law

The Little’s Law column is a way to verify the validity of the response time and throughput for each test run. According to this law, **[response time (seconds)] X [throughput (request/second)]** is equal to the number of concurrent users. Round the product to the nearest whole number. In some cases, when the number of users is large, Little’s Law might not be exact, but it should be close.

Table 2: Ramp up test

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)	Little's Law resp x thru
warm-up						
1						
2						
5						
10						
20						
40						
80						
100						

__ e. The first run is a warm-up test. The response time and throughput are poor because methods are not yet compiled and caches are not yet populated.

__ f. Reset DayTrader, clear previous results, and run the test again with one user. Record the results when the test stops.

__ g. Increase the number of users as shown in the table. Basically, doubling the number for each run, and record the results. Remember to reset DayTrader before each test run.

Information

Finding the saturation point

The expected behavior that your data reveals is that the average response time increases and the throughput increases as the number of concurrent users increases. Eventually, the CPU usage for the application server reaches, or is very close to 100%. Subsequent test runs at 100% CPU for the application server might show increased throughput, but at some specific number of users, the throughput remains the same, or begins to decrease. This condition is known as the saturation point. Use the number of users at or near the saturation for your baseline test plan.

For example, if for 80 users, you see the highest throughput and the CPU usage for the application server is at, or near 100%, use the test plan with 80 users (threads). Change the loop count to 500, and save it as **/usr/labfiles/Test_Plans/DayTrader_baseline80_test.jmx**

__ h. Save your DayTrader baseline test as **/usr/labfiles/Test_Plans/DayTrader_baselineXX_test.jmx,** where **XX** is the number of users that showed the highest CPU usage.

Example

The results that are shown in the table are for comparison. Your data might be different, but you should notice increasing response time and throughput until the application server reaches 100% CPU.

Table 3: Example Ramp up test

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us,sy)	CPU% DB2 server (us,sy)	Little's Law resp x thru
warm-up	1000	27	30	98,2-100%	4,1	1
1	1000	10	63	14,4	4,1	1
2	2000	10	116	22,6	5,2	1
5	5000	14	226	51,9	7,5	3
10	10000	22	330	85,15-100%	12,8	7
20	20000	51	339	82,18-100%	14,9	17
40	40000	107	346	82,18-100%	14,10	37
80	**80000**	**219**	**351**	**81,17**	**16,10**	**77**
100	82600	802	123	95,5-100%	12,4	99

In this example, 80 users appear to be the saturation point.

Section 3: Verify your baseline

In this section, you run the test several times with the number of users that provided the best performance in the previous section. You should see consistent performance results.

__ 1. Restore the tradedb database.

__ a. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ b. Open a terminal window and enter the command: **`su - db2inst1`**

__ c. Run the following script to restore the tradedb database.

`/usr/labfiles/db2scripts/db2restore.sh`

__ d. Run the following script to unquiesce the database.

`/usr/labfiles/db2scripts/db2unquiesce.sh`

__ 2. Restart the node agent and TradeServer1.

__ 3. Run the **`DayTrader_baseline80_test.jmx`** three times and record your results.

Table 4: Verify baseline test

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)	Little's Law resp x thru
warm-up						

__ 4. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

Example

The results that are shown in the table are for comparison. Your data might be different.

Table 5: Example baseline verification

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% app server (us,sy)	CPU% DB2 server (us,sy)	Little's Law resp x thru
warm-up	40000	920	86	92,8	8,4	
80	40000	166	454	81,19	19,13	75
80	40000	168	449	82,18	20,12	75
80	40000	175	433	82,18	19,12	76

- Response time_____170_____(ms)

- Throughput _____445_____(req/sec)

__ 5. You can keep Apache JMeter running on hostA because you use it in the next several exercises.

Section 4: Cleanup

If this exercise is the last exercise you do for several hours, you should stop the WebSphere servers.

__ 1. On **hostA**, stop the Deployment Manager.

__ a. Open a terminal window and enter the following command:
cd <*profiles_root*>/Dmgr/bin

__ b. Enter the command:
./stopManager.sh -username wasadmin -password web1sphere

__ 2. On **hostB**, stop the node agent and TradeServer1.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
cd <*profiles_root*>/profile1/bin

__ c. Enter the command:
./stopNode.sh -username wasadmin -password web1sphere

__ d. Enter the command:
./stopServer.sh TradeServer1 -username wasadmin -password web1sphere

End of exercise

Exercise review and wrap-up

In this exercise, you leaned how to use Apache JMeter to run a ramp up test on the DayTrader application. You learned how to tune and backup the DayTrader database. You ran a ramp up test on DayTrader to find the saturation point. You monitored CPU usage on the application server host and the database server host. You use the results of the ramp up tests to establish a baseline of performance for the DayTrader application server.

Exercise 5. Performance monitoring tools

What this exercise is about

The purpose of this exercise is to familiarize you with the performance monitoring tools and performance advisor facilities that are available in WebSphere Application Server and IBM Support Assistant.

What you should be able to do

At the end of this exercise, you should be able to:

- Monitor performance by using Tivoli Performance Viewer
- Use the performance advisors, including runtime Performance Advisor and Performance Advisor in Performance Viewer
- Use the Health Center tool in the IBM Support Assistant to monitor a running JVM

Introduction

Monitoring the application server is an important part of the performance tuning cycle. This exercise shows you how to use monitoring tools that WebSphere Application Server and the IBM Support Assistant provide.

Tools such as the Tivoli Performance Viewer and Health Center not only monitor the performance of a running application server, they also provide advice and recommendations for improving performance.

Requirements

Exercise 3: Installing DayTrader and **Exercise 4: Use Apache JMeter to load test DayTrader** must be successfully completed before you can complete this exercise. At the end of Exercise 4, you created an Apache JMeter test plan, **`/usr/labfiles/Test_Plans/DayTrader_baselineXX_test.jmx`,** which you are going to use in this exercise.

Exercise instructions

Preface

- Most steps in this exercise are done on **hostA** unless otherwise specified.

Note

<*profiles_root*> = /opt/IBM/WebSphere/AppServer/profiles

Section 1: Start the WebSphere servers

If the Deployment Manager, node agent, and TradeServer1 are already running, stop and restart them now.

__ 1. On **hostA**, start the Deployment Manager.

__ a. Open a terminal window and enter the command:
cd <*profiles_root*>/Dmgr/bin

__ b. Enter the command:
./startManager.sh

__ 2. Restore the tradedb database.

__ a. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ b. Open a terminal window and enter the command:
su - db2inst1

__ c. Run the following script to restore the tradedb database.

/usr/labfiles/db2scripts/db2restore.sh

__ d. Run the following script to unquiesce the database.

/usr/labfiles/db2scripts/db2unquiesce.sh

Note

DB2 warning message: SQL2540W Restore is successful, however a warning "2539" was encountered during Database Restore while processing in No Interrupt mode.

This message is nothing to worry about. It is a warning that you are overwriting an existing database. If the restore command did not specify WITHOUT PROMPTING, you would see the message in the terminal window, and you must answer YES to continue.

__ 3. On **hostB**, start the node agent and TradeServer1.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
cd <*profiles_root*>/profile1/bin

__ c. Enter the command:
./startNode.sh

__ d. After the node agent starts, start TradeServer1 by entering the command:
./startServer.sh TradeServer1

Section 2: Run the DayTrader baseline test to warm up TradeServer1

In this section, you run the DayTrader baseline test that you developed in the previous lab exercise. You want to verify the baseline performance and warm up TradeServer1.

__ 1. Start Apache JMeter.

__ a. On **hostA**, open a terminal window and enter the command:
cd /opt/apache-jmeter/bin

__ b. Enter the command:
./jmeter.sh &

__ 2. Open the **DayTrader_baselineXX_test.jmx** test plan.

__ a. Click **File > Open**, and navigate to
/usr/labfiles/Test_Plans/DayTrader_baselineXX_test.jmx

__ b. Click **Open**.

__ c. In the JMeter tree, click **Thread Group** and change the Loop Count to **100**.

__ 3. Prepare to load test the DayTrader.

__ a. Start a web browser, and go to **http://hostB:9081/daytrader**

__ b. Click the **Configuration** tab, and click the **Reset DayTrader** link. Keep this browser open as you are going to reset the DayTrader for each run of the load test.

__ c. Verify that the reset completes successfully.

DayTrader Scenario Runtime Statistics

Trade Reset completed successfully — Modify runtime configuration

Benchmark runtime configuration summary	Value
Run-Time Mode	Full EJB3
Order-Processing Mode	Synchronous
Scenario Workload Mix	Standard
Web Interface	JSP
Active Traders / Trade User population	15000 / 15000
Active Stocks / Trade Stock population	10000 / 10000
Benchmark scenario verification	

__ d. In Apache JMeter, click **Summary Report**.

__ 4. Run the test plan.

__ a. In JMeter, click **Run > Start** (or click the green arrow) to start the test.

__ b. Run the test three more times and record the performance results. Remember to reset DayTrader before each test run.

Table 6: Verify baseline test

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% app server (us+sy)	CPU% DB2 server (us+sy)	Little's Law resp x thru
warm-up						

__ 5. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

Section 3: Monitoring with the Tivoli Performance Viewer

The Tivoli Performance Viewer (TPV) enables administrators and programmers to monitor the overall health of WebSphere Application Server from within the administrative console. By using Tivoli Performance Viewer, you can view current activity or log Performance Monitoring Infrastructure (PMI) performance data for the following resources:

- System resources such as CPU usage
- WebSphere pools and queues such as a database connection pool
- Customer application data such as servlet response time

__ 1. From the DayTrader Configuration tab, click **Reset DayTrader**.

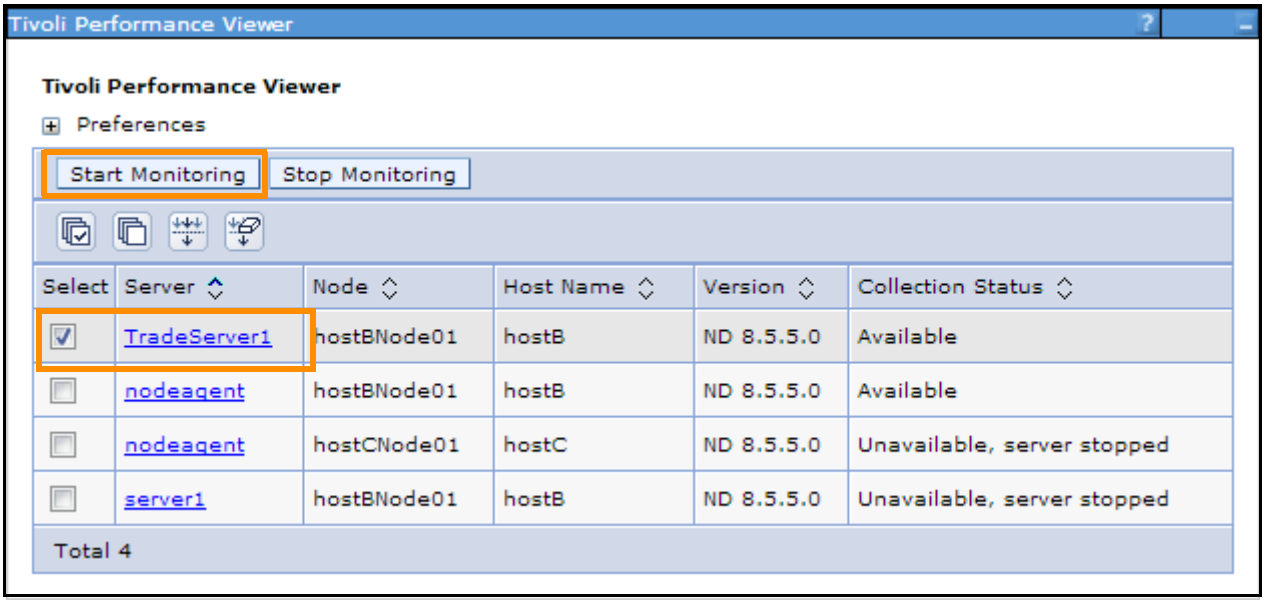
__ 2. In JMeter, change the loop count to **Forever** and run the Test Plan.

__ 3. Start monitoring TradeServer1 with the Tivoli Performance Viewer.

__ a. On **hostA**, log in to the administrative console.

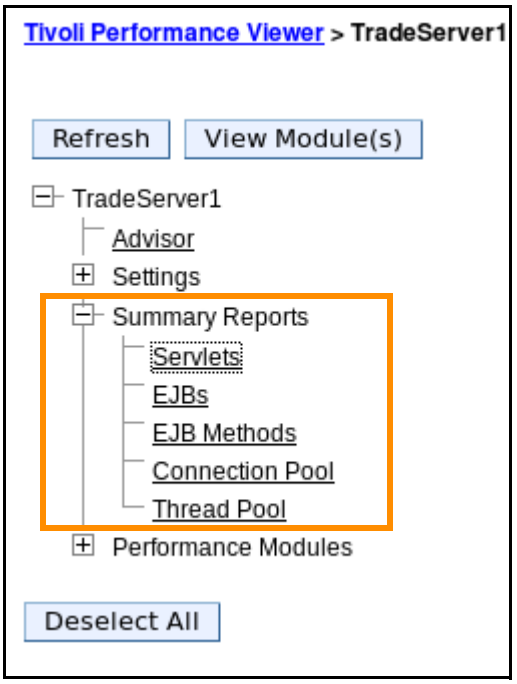
__ b. Click **Monitoring and Tuning > Performance Viewer > Current activity.**

__ c. Select the **TradeServer1** check box and click **Start Monitoring.**



__ d. When the collection status changes to Monitored, click the **TradeServer1** link.

__ e. In the Tivoli Performance Viewer pane, expand **Summary Reports > Servlets**.



__ 4. Click **Servlets** to examine the Servlets Summary Report.

Servlets Summary Report

Start Logging

Name	Application	Total Requests	Avg Resp Time (ms)	Total Time (ms)	Time
/marketSummary.jsp	DayTrader3-EE6#web.war	244,305	4.31	1,052,847	12:00:13 PM
/runStats.jsp	DayTrader3-EE6#web.war	5	2.4	12	12:00:13 PM
/tradehome.jsp	DayTrader3-EE6#web.war	244,305	8.645	2,111,954	12:00:13 PM
Faces Servlet	DayTrader3-EE6#web.war	0	0	0	12:00:13 PM
TradeAppServlet	DayTrader3-EE6#web.war	244,350	275.278	67,264,280	12:00:13 PM
TradeConfigServlet	DayTrader3-EE6#web.war	5	1,134.2	5,671	12:00:13 PM
TradeScenarioServlet	DayTrader3-EE6#web.war	244,350	280.563	68,555,576	12:00:13 PM
ax.ws.rs.core.Application	DayTrader3-EE6#Rest.war	0	0	0	12:00:13 PM
rspservlet	ibmasyncrsp.war	0	0	0	12:00:13 PM

Total 9

This report shows a performance snapshot of the DayTrader servlets and JSPs. For each, you can see the total requests and average response time (ms). The JSPs might be good candidates for caching with the dynamic caching feature.

__ 5. Click **EJBs** to examine the EJBs Summary Report.

EJBs Summary Report

Start Logging

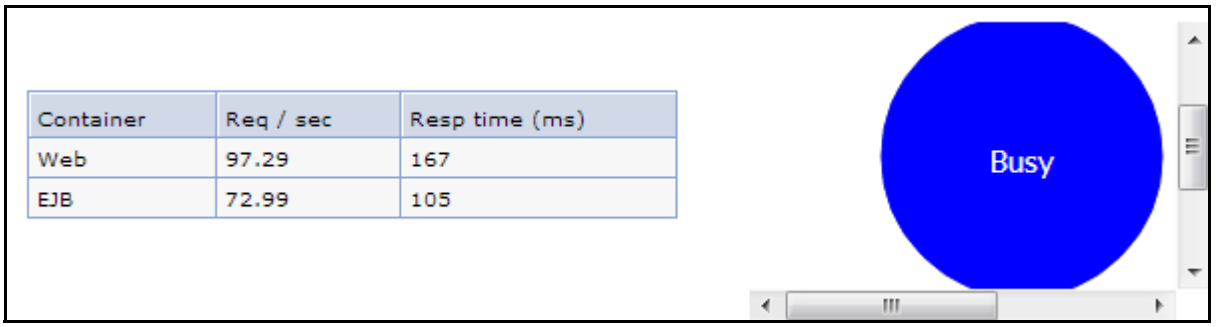
Name	Application	Method Calls	Avg Resp Time (ms)	Total Time (ms)	Time
DTBroker3MDB	DayTrader3-EE6#dt-ejb.jar	0	0	0	12:06:48 PM
DTStreamer3MDB	DayTrader3-EE6#dt-ejb.jar	0	0	0	12:06:48 PM
TradeSLSBBean	DayTrader3-EE6#dt-ejb.jar	768,989	106.008	81,518,952	12:06:48 PM

Total 3

This report shows a performance snapshot of the DayTrader EJBs. For each EJB, you can see the number of method calls and average response time.

__ 6. Click **Advisor** to view the Tivoli Performance Advisor. The advisor screen has three parts.

__ a. The top part of the screen shows a snapshot of how busy the server is. The pie chart shows busy and idle sectors for the CPU. In this graphic, the CPU is 100% busy. You also see a breakdown of the web container and EJB container requests and response times.

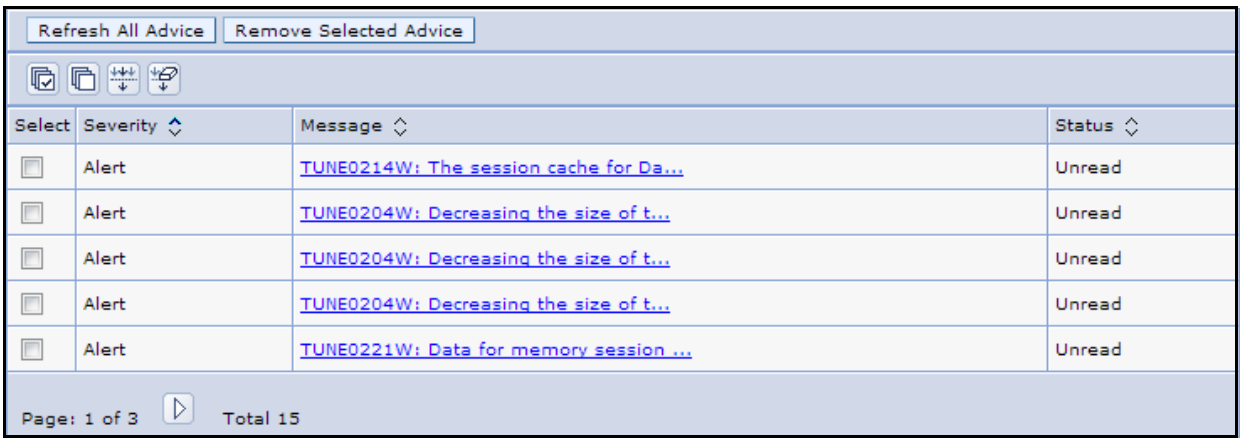


__ b. The middle part of the screen shows a snapshot of thread pools and data source connection pools.

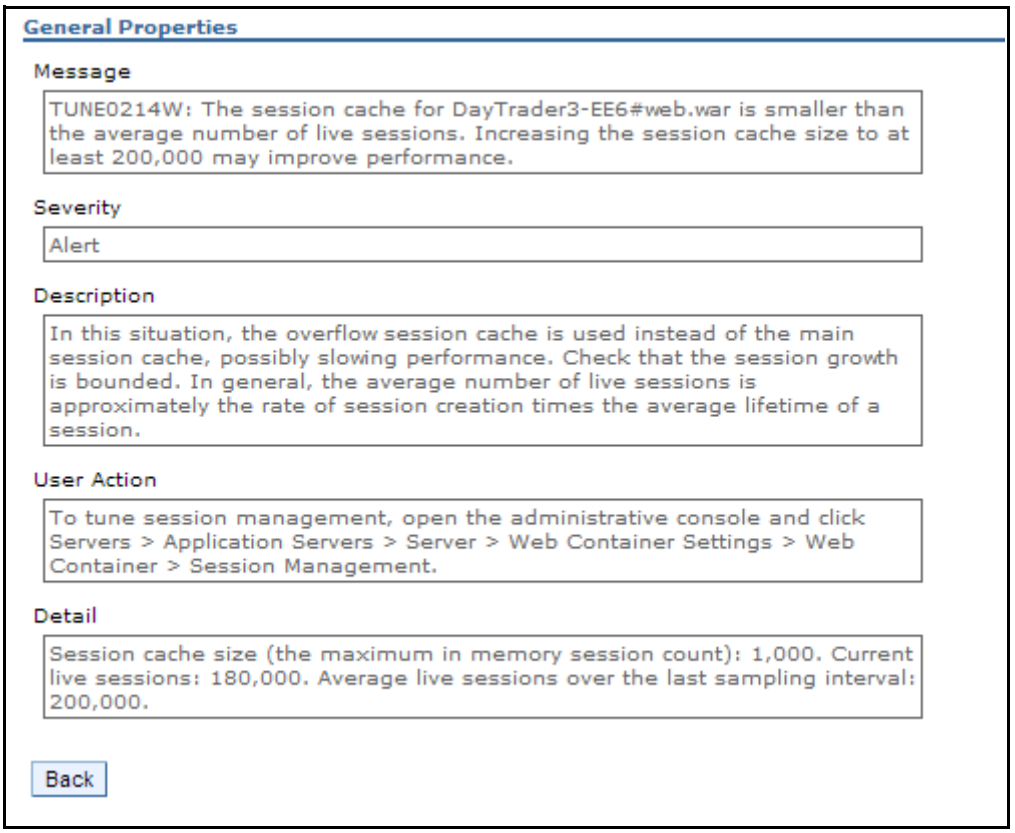
Resource	Busy	Idle	Total
AriesThreadPool	0	5	5
Default	3	4	7
HAManager.thread.pool	0	2	2
Message Listener	0	50	50
Object Request Broker	0	50	50
ProcessDiscovery	0	1	1
SIBFAPInboundThreadPool	0	50	50
SIBFAPThreadPool	0	50	50
SIBJMSRAThreadPool	0	41	41
SoapConnectorThreadPool	0	5	5
TCPChannel.DCS	2	1	3
WMQJCAResourceAdapter	0	50	50
WebContainer	45	5	50
jdbc/NoTxTradeDataSource	0	1	1
jdbc/TradeDataSource	8	2	10
jdbc/DefaultEJBTimerDataSource	0	0	0

In particular, notice the number of busy and idle web container threads and TradeDataSource connections. Web container threads and data source connection pools are common performance bottlenecks and are good candidates for tuning.

__ c. The bottom part of the screen shows a table of performance advice entries. Each entry has a severity level that can be Alert, Config, or Warning. You can sort the table by severity by clicking the Severity column. If no advice is listed, click **Refresh All Advice**.

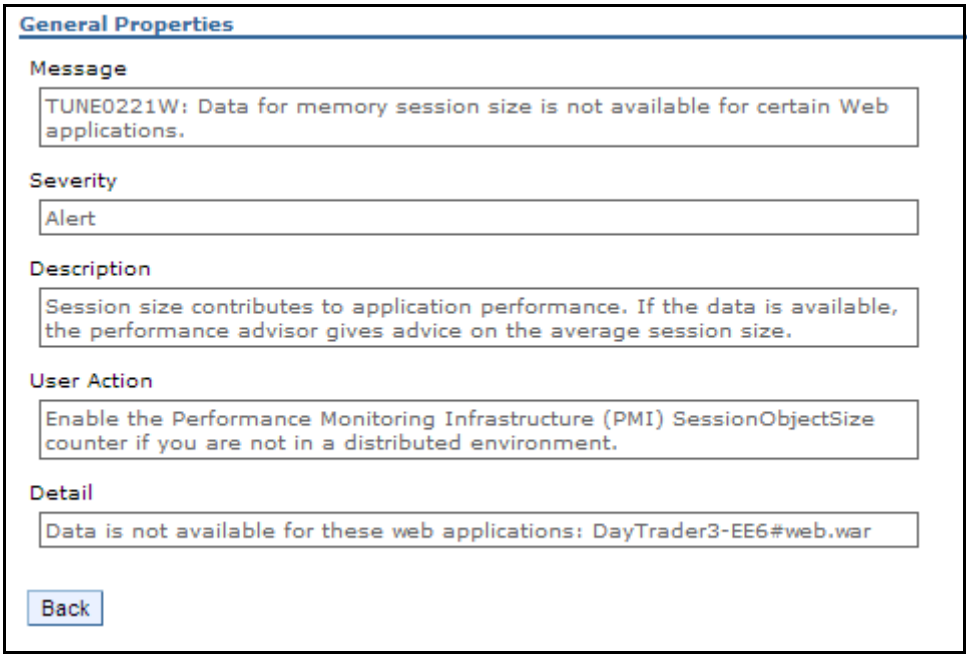


__ d. Click the message link for all of the advice entries to read the information.



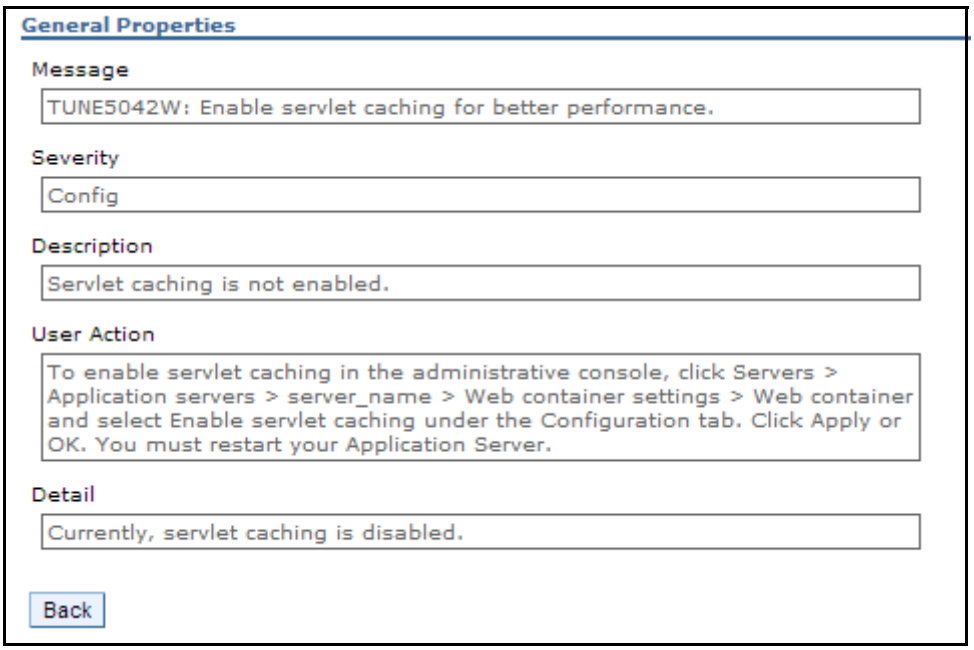
This example shows an **Alert** severity level message. Notice that the **Message** box provides the tuning advice and the **User Action** box provides instructions on how to use the administrative console to tune the server setting. The **Description** box provides a rationale for the tuning advice, and the **Detail** box provides information about the current state of your server.

__ e. Click **Back** to return to the list of advice entries. Examine other entries.



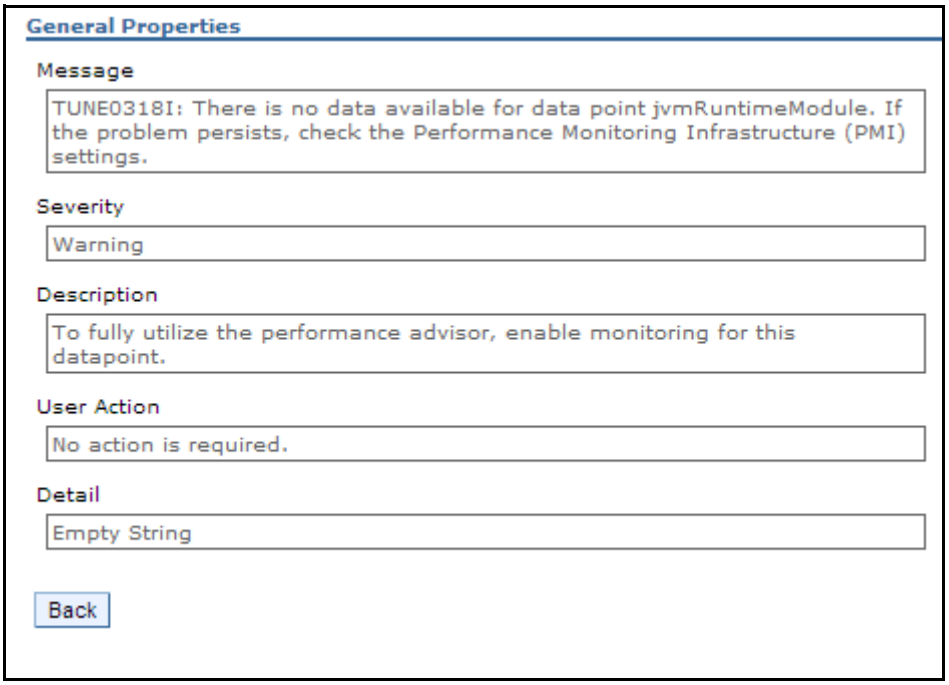
This example shows an alert level message that indicates that memory session size data is not available. This data is gathered by changing the PMI statistics set for the server.

__ f. Here is an example of a common config severity message.



The **User Action** box describes how to enable servlet caching (Dynamic caching) for your server. You are going to enable servlet caching in a future exercise.

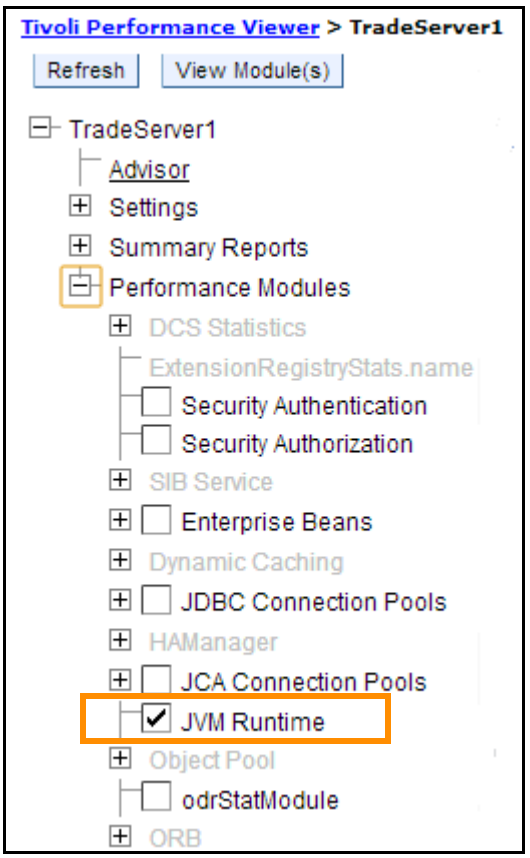
__ g. Here is an example of a warning level message



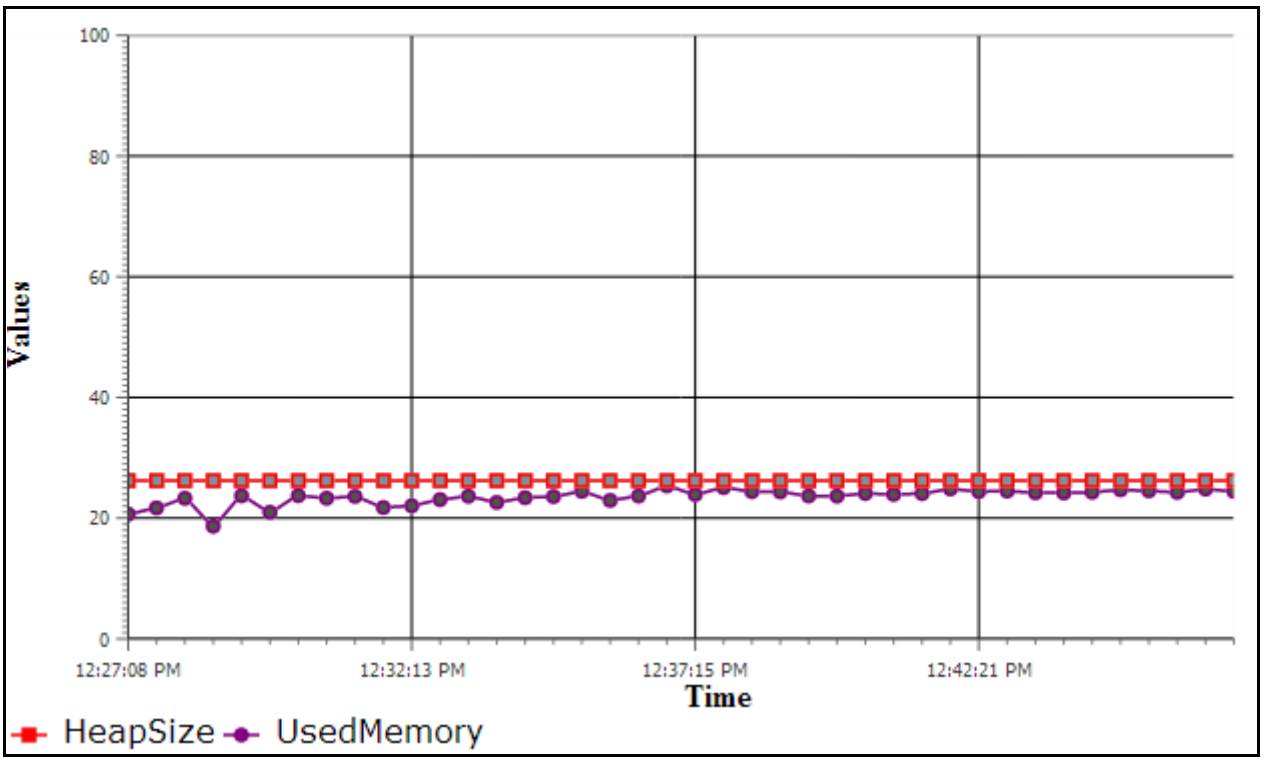
Again, this data is gathered by changing the PMI statistics set for the server.

__ 7. Monitor the JVM run time.

__ a. Expand **Performance Modules**, and select the **JVM Runtime** check box.



__ b. Click **View Modules**.



__ c. The graph shows the Heap size at or near 256 MB, which is the default maximum for the application server. Used heap is very close to the maximum heap. Increasing the Maximum heap size might reduce the frequency of garbage collections, which in turn, might increase throughput.

__ 8. Monitor the jdbc/TradeDataSource.

__ a. Under Performance Modules, expand **JDBC Connection Pools > DB2 Universal JDBC Driver Provider Only (XA).** Select the **jdbc/TradeDataSource** check box.

__ b. Click **View Modules**

__ c. Scroll down to the list of PMI metric for the data source.

Reset To Zero | Clear Buffer | View Table | Show Legend

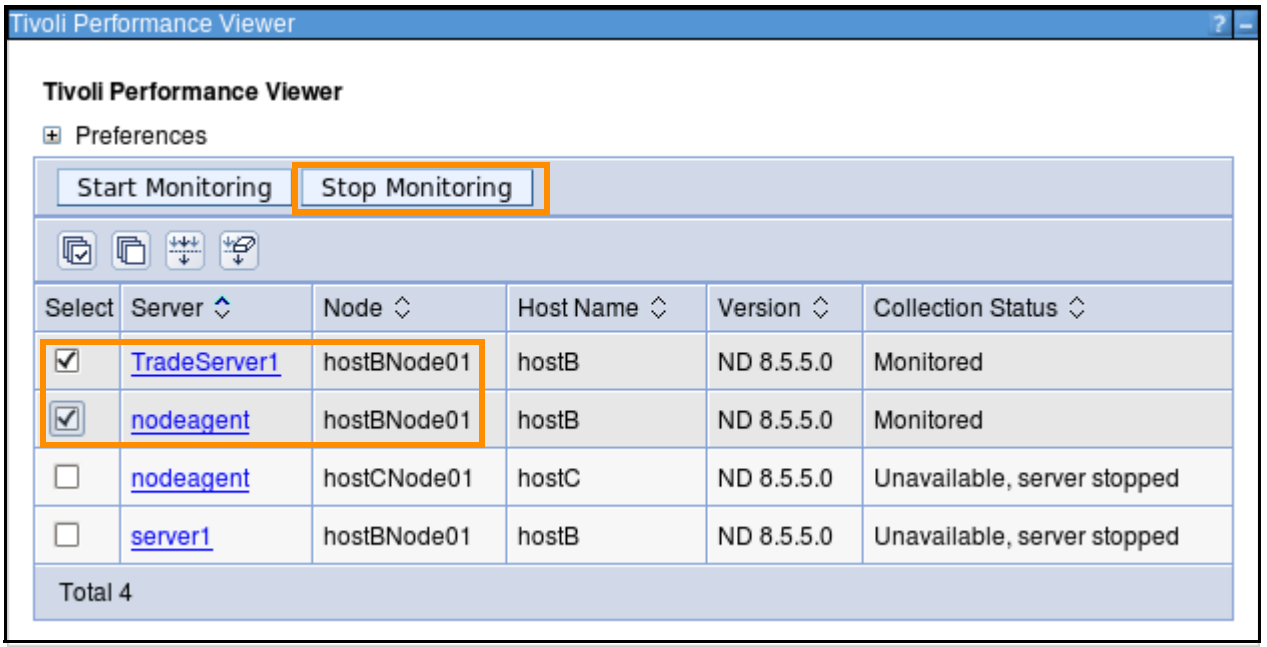
Select	Marker	Name	Value	Scale Update	Scaled Value
jdbc/TradeDataSource					
☑		CreateCount	19.0	1.0	19.0
☑		CloseCount	9.0	1.0	9.0
☑		PoolSize	10.0	1.0	10.0
☐		FreePoolSize	6.0	1.0	6.0
☐		WaitingThreadCount	40.0	1.0	40.0
☐		PercentUsed	20.0	1.0	20.0
☐		UseTime	29.118305	1.0	29.118305
☐		WaitTime	104.04573	0.1	10.404573

__ d. In particular, notice the two metrics **WaitingThreadCount** and **WaitTime**. WaitingThreadCount is average number of threads that are concurrently waiting for a connection. WaitTime is average wait time in milliseconds until a connection is granted. Large values for these two metrics are likely to increase response time.

__ 9. Stop monitoring TradeServer1.

__ a. From the administrative console, click **Monitoring and Tuning > Performance Viewer > Current activity.**

__ b. Select the **TradeServer1** check box and the **nodeagent (hostB)** check box, and click **Stop Monitoring.**



Section 4: Using the Runtime Performance Advisor

The Runtime Performance Advisor provides advice to help tune systems for optimal performance and is configured by using the administrative console. Running in the JVM of the application server, this advisor periodically checks for inefficient settings and issues recommendations as standard product warning messages. These recommendations are displayed both as warnings in the administrative console under WebSphere Runtime Messages in the WebSphere Status pane and as text in the application server's SystemOut.log file. Enabling the Runtime Performance Advisor has minimal system performance impact.

__ 1. Enable the Runtime Performance Advisor and look at the messages it produces.

__ a. Click **Servers > Server Types > WebSphere application servers > TradeServer1 > Performance > Performance and Diagnostic Advisor Configuration.** Click the **Configuration** tab.

__ b. Check **Enable Performance and Diagnostic Advisor Framework**. Verify that the Number of processors is **1**, and change it if needed.

__ c. Set the Calculation Interval to **4** minutes, the default setting.

__ d. The Minimum CPU for Working System and CPU Saturated values are set to 50 and 90 by default. You can keep these values but must make sure that your CPU usage is above 50% when generating load with Apache JMeter during the following exercises.

__ e. Click **OK** and then click **OK** again on the message that informs you that you should simulate a production level workload for the Advisor to provide useful results.

__ f. Click **Save** to save the configuration changes.

__ g. Go back into the Performance and Diagnostic Advisor Configuration, click the **Performance and Diagnostic Advice configuration** link, and then the **Configuration** tab.

__ h. Notice which advice items are currently monitored. Their status is started. Most of these items have low performance impact. Keep the default set of advice items.

Start | Stop

Select	Advice name	Advice applied to component	Advice type	Performance impact	Advice status
You can administer the following resources:					
☐	Thread Max Connections exceeded Diagnostic Alert	J2C Connection Manager	Diagnostic	High	✖
☐	LTC Nesting Threshold Exceeded Alert	J2C Connection Manager	Diagnostic	High	✖
☐	Serial Reuse Violation Diagnostic Alert	J2C Connection Manager	Diagnostic	High	✖
☐	Session Cache Size with Overflow Disabled	Web Container Session Manager	Performance	Low	➔
☐	Session Cache Size with Overflow Enabled	Web Container Session Manager	Performance	Low	➔
☐	Persisted Session Size	Web Container Session Manager	Performance	Low	➔
☐	Persisted Session Time	Web Container Session Manager	Performance	Low	➔
☐	Unbounded Web Container Thread Pool	Web Container	Performance	Low	➔
☐	Bounded Web Container Thread Pool	Web Container	Performance	Low	➔
☐	Unbounded ORB Service Thread Pool	Orb Service	Performance	Low	➔
☐	Bounded ORB Service Thread Pool	Orb Service	Performance	Low	➔
☐	Connection Pool Size	Data Source	Performance	Low	➔
☐	Prepared Statement Cache Size	Data Source	Performance	Low	➔
☐	Memory Leak Rule	Jvm	Performance	Low	➔

__ i. Restart **TradeServer1**.

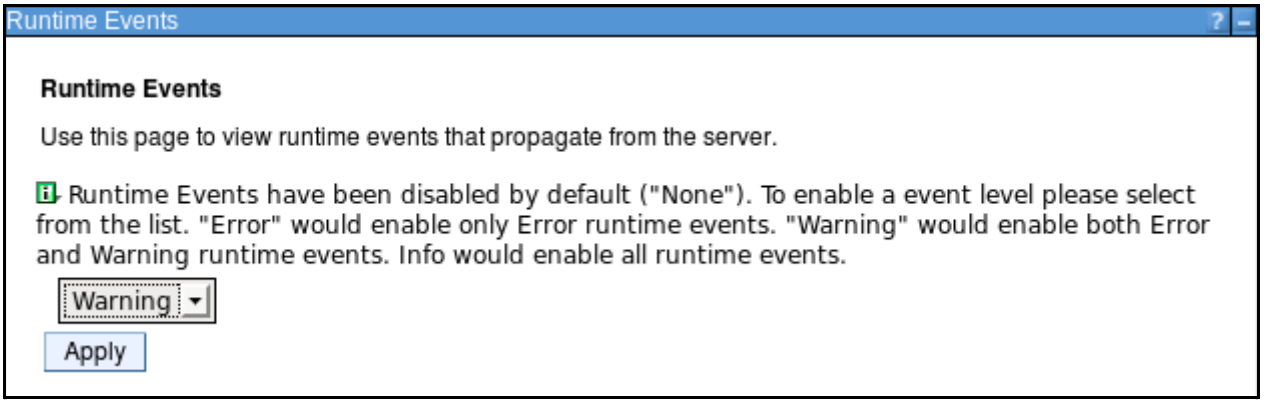
__ 2. Warm-up TradeServer1 by using Apache JMeter to run your baseline test plan.

__ a. For your test plan, set the loop count to **100**.

__ b. Keep the number of users (threads) at your determined value XX.

__ 3. Examine messages from the Runtime performance advisor.

__ a. Click **Troubleshooting > Runtime Messages > Runtime Warning**. The Runtime Advisor messages are displayed as TUNExxxxxx messages. You do not see any messages at the moment. Select **Warning** from the Runtime Events menu list, click **Apply**, and **Save** the change.

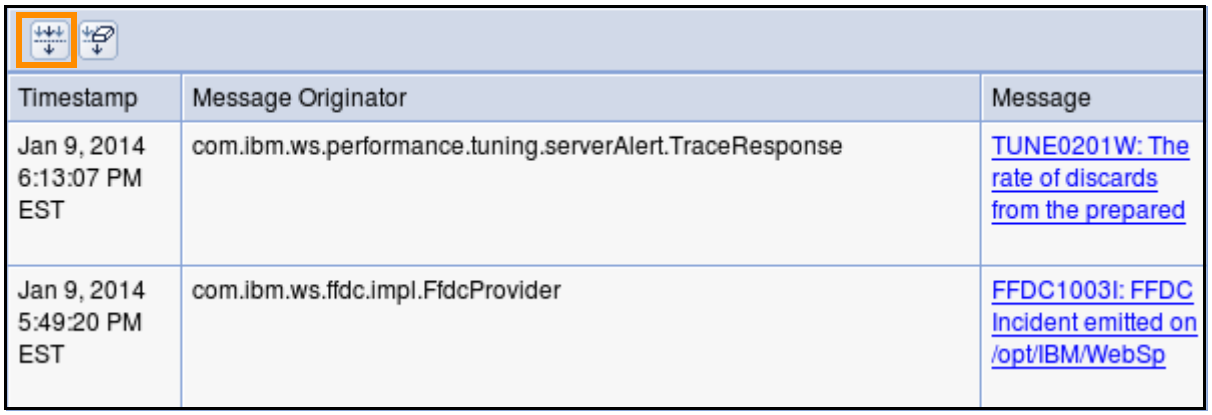


__ 4. From the DayTrader Configuration tab, click **Reset DayTrader**.

__ 5. In JMeter, change the loop count to **Forever** and run the Test Plan.

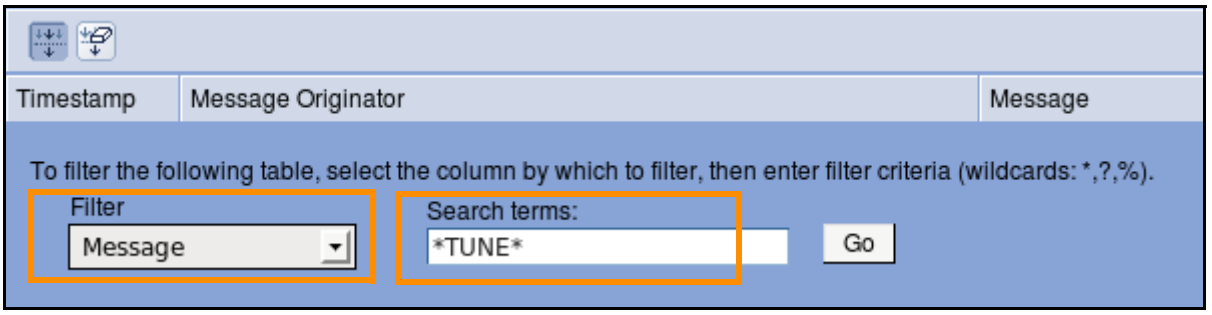
__ 6. Return to the administrative console and click **Runtime Warning** again to refresh the pane.

__ 7. After a few minutes, several TUNE messages appear among other warning level messages. Because there might be many messages, use the filter function to display only the TUNE messages. There are two icons at the top of the table. The left one sets the filter values; the other one clears them. Click the left icon to set a filter.

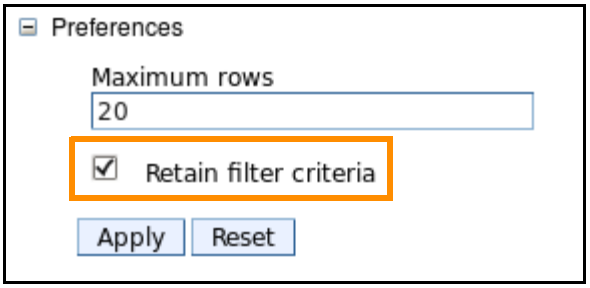


Timestamp	Message Originator	Message
Jan 9, 2014 6:13:07 PM EST	com.ibm.ws.performance.tuning.serverAlert.TraceResponse	TUNE0201W: The rate of discards from the prepared
Jan 9, 2014 5:49:20 PM EST	com.ibm.ws.ffdc.impl.FfdcProvider	FFDC1003I: FFDC Incident emitted on /opt/IBM/WebSp

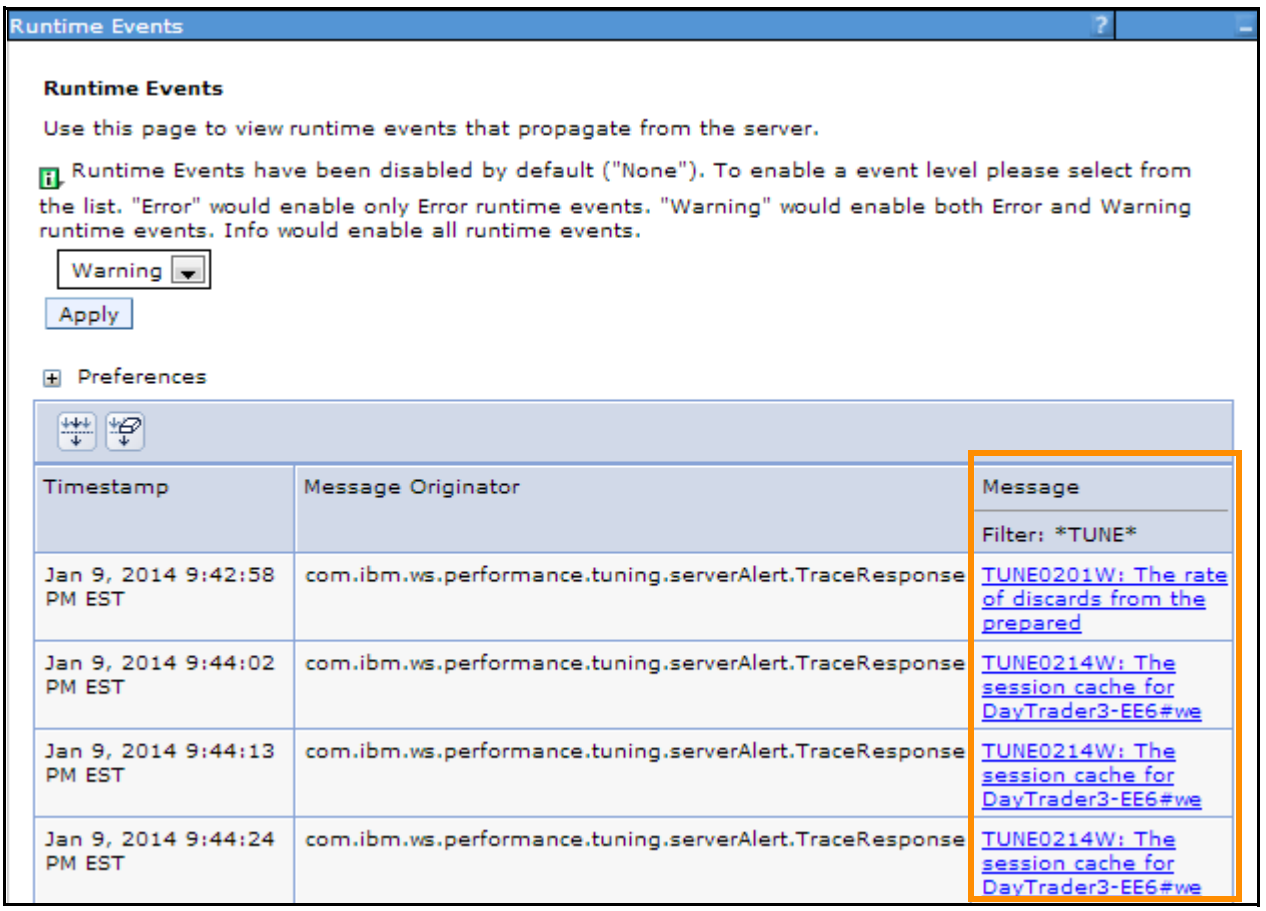
__ 8. A new row is displayed in the table. Select the Filter type **Message** from the menu list. Enter ***TUNE*** into the Search terms field. Click **Go**.



__ 9. Expand **Preferences** above the table. Select the **Retain filter criteria** check box. Click **Apply**. This way the filter is not cleared when you refresh the pane.



__ 10. You should see several **TUNE** messages now.



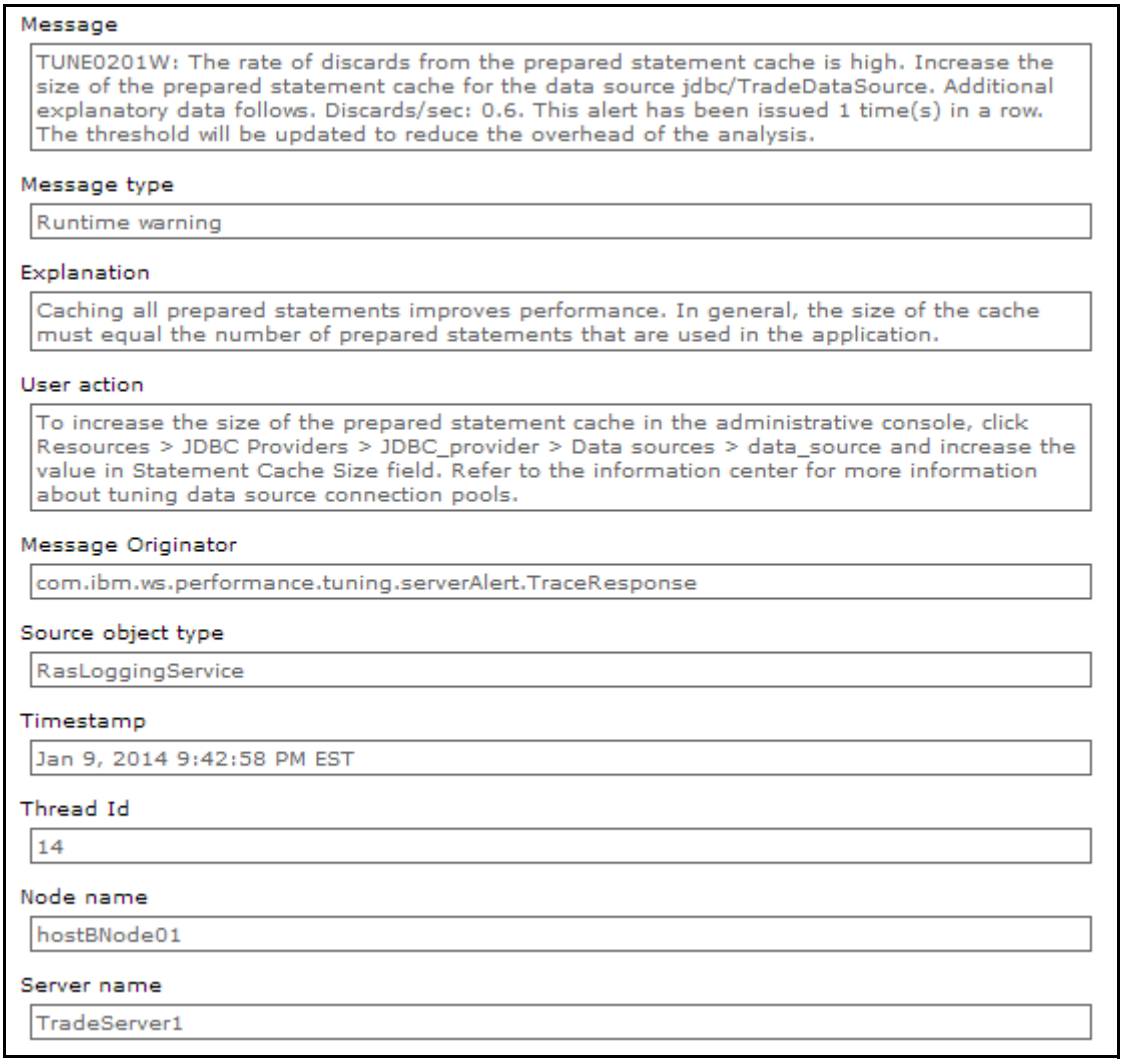
Troubleshooting

No TUNE messages?

The TUNE messages are generated as a result of how you configure the Runtime Performance Advisor. Because you used 50% as the minimum working CPU, your load test must drive the CPU usage above 50%. If you do not see any TUNE messages, increase the number of users (threads) in your test plan to 80 or higher. Increase the loop count to 500. Run the test plan again.

__ 11. Your list of TUNE messages might be different from the ones that are shown in the screen capture. Click the link for each Tune messages to examine the details.

__ 12. One helpful message is **TUNE0201W**. Here are the details of this message.



In another exercise, you learn how to monitor the prepared statement cache by using the Tivoli Performance View. You also tune the cache for the Trade data source.

__ 13. Click **Back**. Look at any other message that you are interested in.

__ 14. Finally, select **None** from the Runtime Events menu list. Click **Apply**. Save the change.

Note

Instead of using the administrative console to look at the TUNE-Messages, you can also go to the SystemOut.log file of the appropriate application server and look at the messages there. However, you have no GUI in this case and also not as many details are available.

Section 5: Monitoring with the Health Center

The IBM Monitoring and Diagnostic Tools for Java - Health Center is a diagnostic tool for monitoring the status of a running Java virtual machine (JVM). The Heath Center uses a small amount of processor time and memory, and can open some log and trace files for analysis.

The tool is provided in two parts:

- The Health Center agent that collects data from a running application. This agent is built in to the Java run time and can be enabled for a server by using the **`-Xhealthcenter`** generic argument.
- An Eclipse-based client that connects to the agent. The client interprets the data and provides recommendations to improve the performance of the monitored application.

The Health Center is also used to monitor Java applications in addition to JVMs.

__ 1. Configure TradeServer1 to have the Health Center tool monitor the JVM.

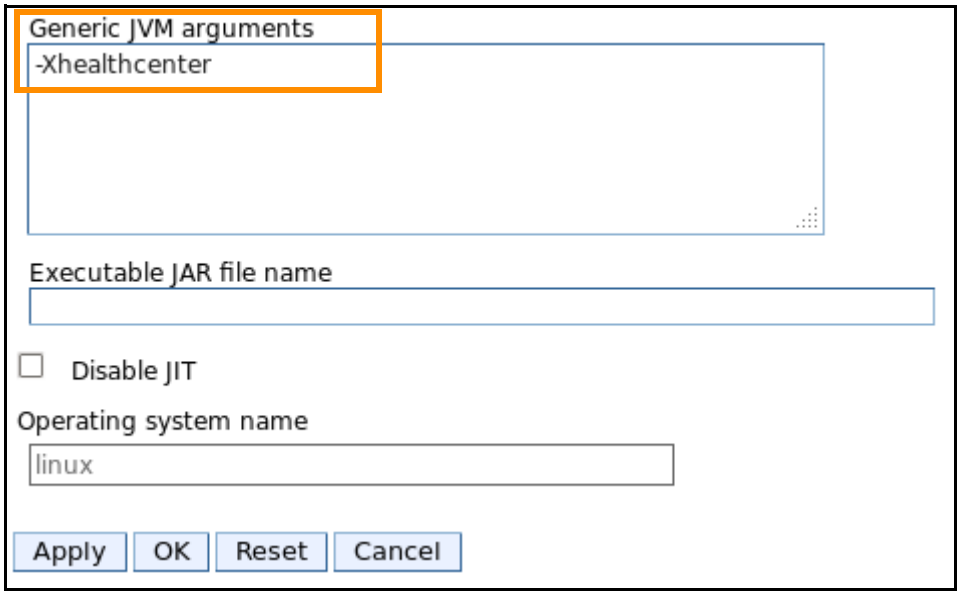
__ a. Log in to the administrative console.

__ b. Click **Server Type > WebSphere application servers > TradeServer1**.

__ c. On the configuration tab for TradeServer1, click **Java and Process Management > Process definition**.

__ d. Under Additional properties, click **Java Virtual Machine**.

__ e. In the Generic JVM arguments box, enter: **`-Xhealthcenter`**



__ f. Click **OK**.

__ g. Click **Save** to save configuration changes.

__ h. When the node synchronizes, click **OK.**

__ i. Restart **TradeServer1** for the new configuration to take effect.

__ 2. Prepare to load test the DayTrader.

__ a. On **hostA**, start a web browser, and go to **`http://hostB:9081/daytrader`**

__ b. Click the **Configuration** tab, and click the **Reset DayTrader** link. Keep this browser open because you are going to reset the DayTrader for each run of the load test.

__ c. Verify that the DayTrader reset completes successfully.

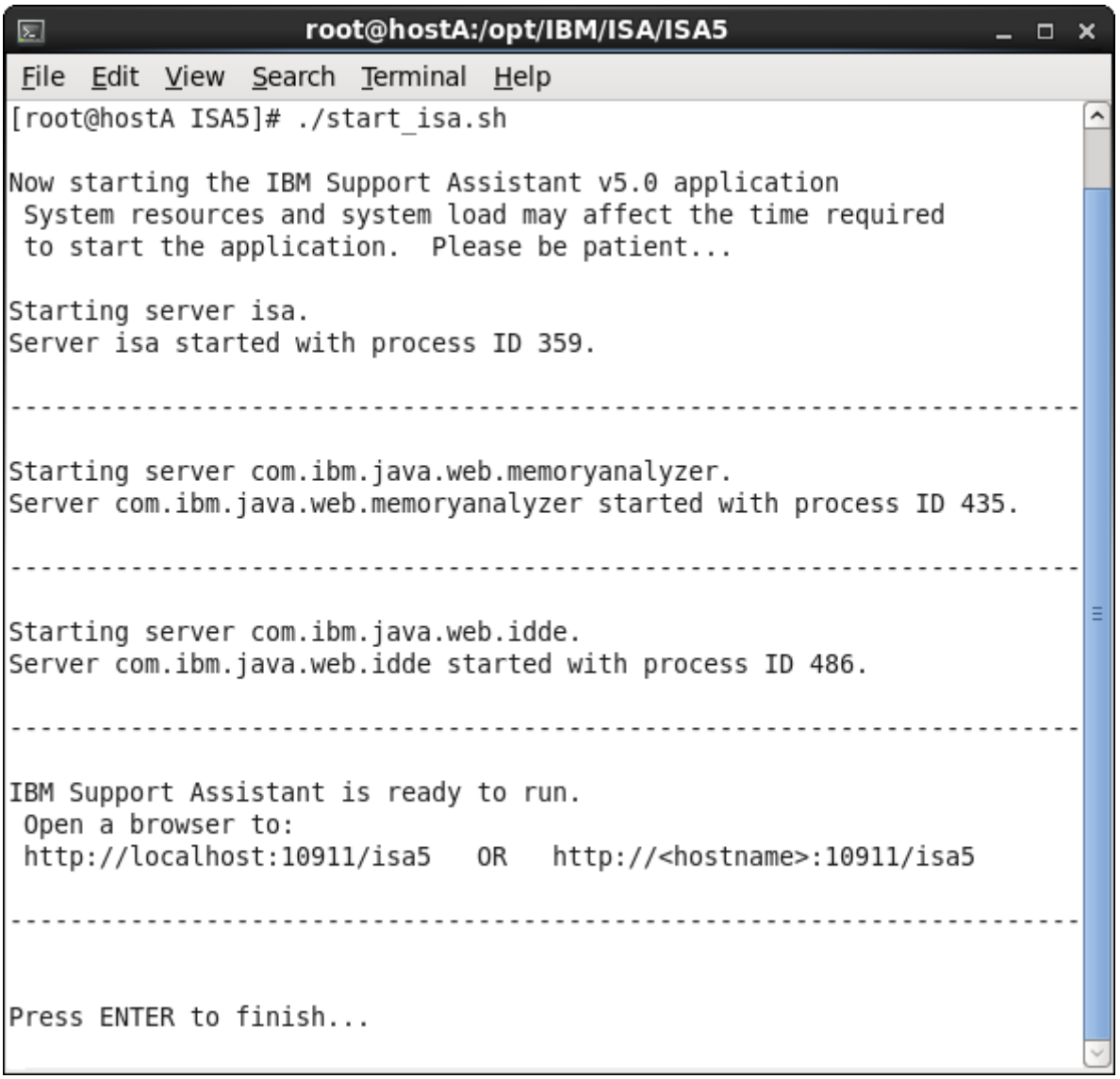
__ 3. Run the DayTrader baseline test plan to warm up TradeServer1.

__ a. From Apache JMeter, click **Summary Report.**

__ b. Click **Run > Start** (or click the green arrow) to start the test.

__ 4. Start the Java Health Center.

__ a. On **hostA**, start ISA from a terminal window by navigating to **/opt/IBM/ISA/ISA5**, and entering: **./start_isa.sh**



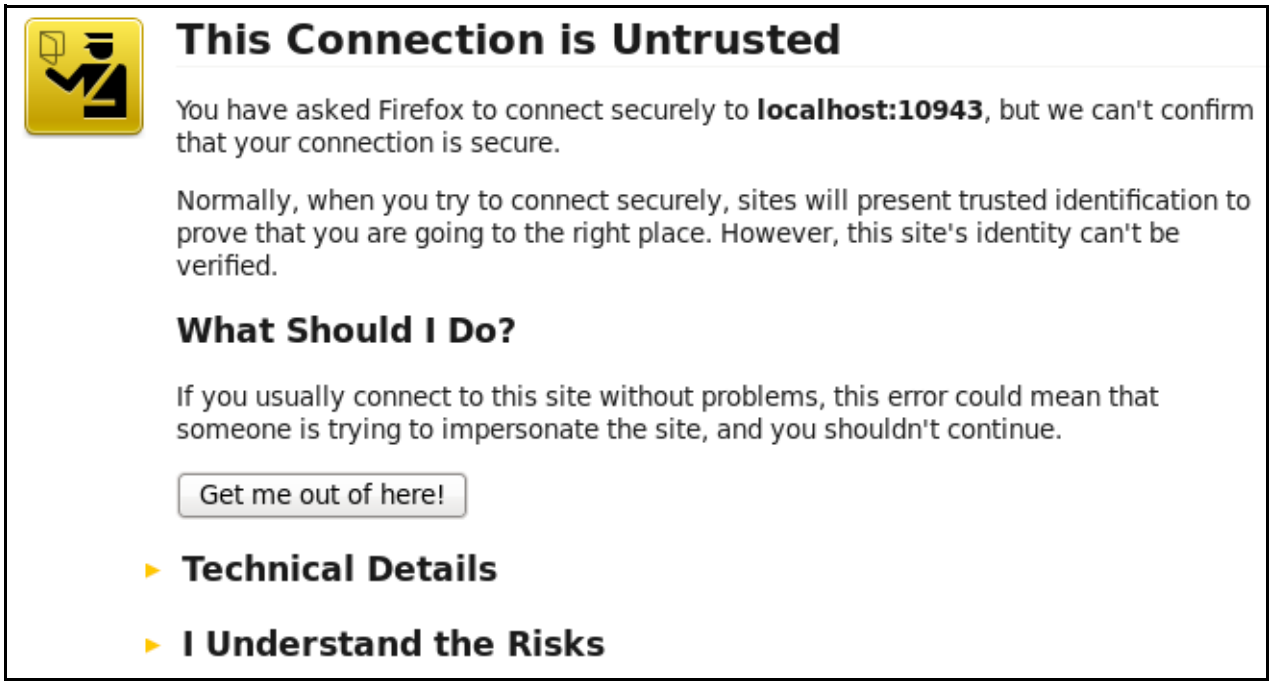
__ b. Press **Enter**.

__ c. Minimize, but to not close the terminal window. You use it again to stop ISA.

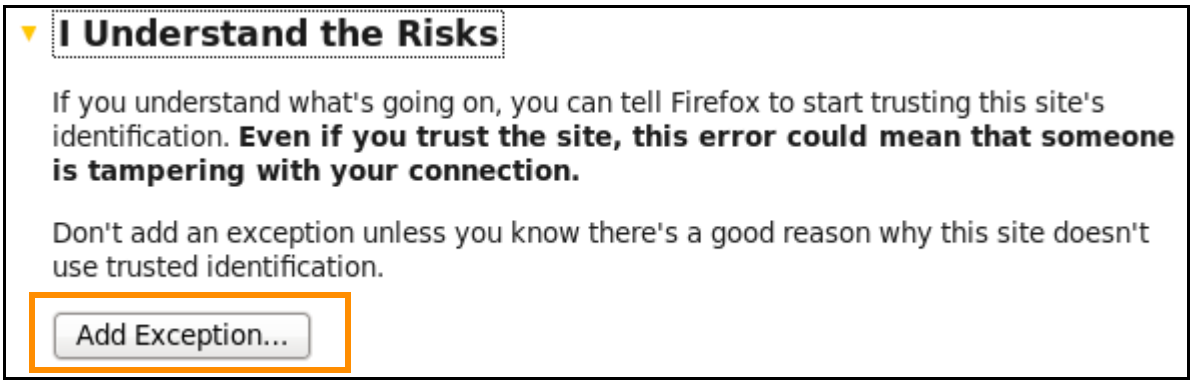
__ 5. Open IBM Support Assistant in a browser.

__ a. Open a Firefox web browser and type the address: **http://localhost:10911/isa5**

__ b. The browser shows a message that the server connection is untrusted. The screen shows how Firefox handles this message: Click **I understand the Risks**.

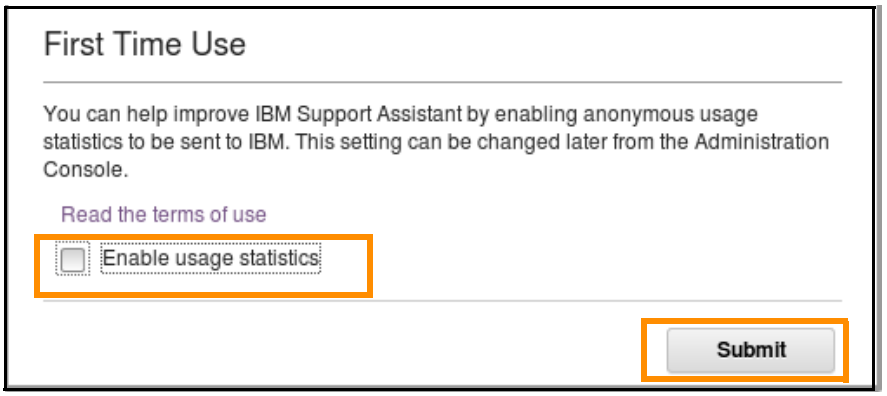


__ c. Firefox shows this screen. Click **Add Exception.**

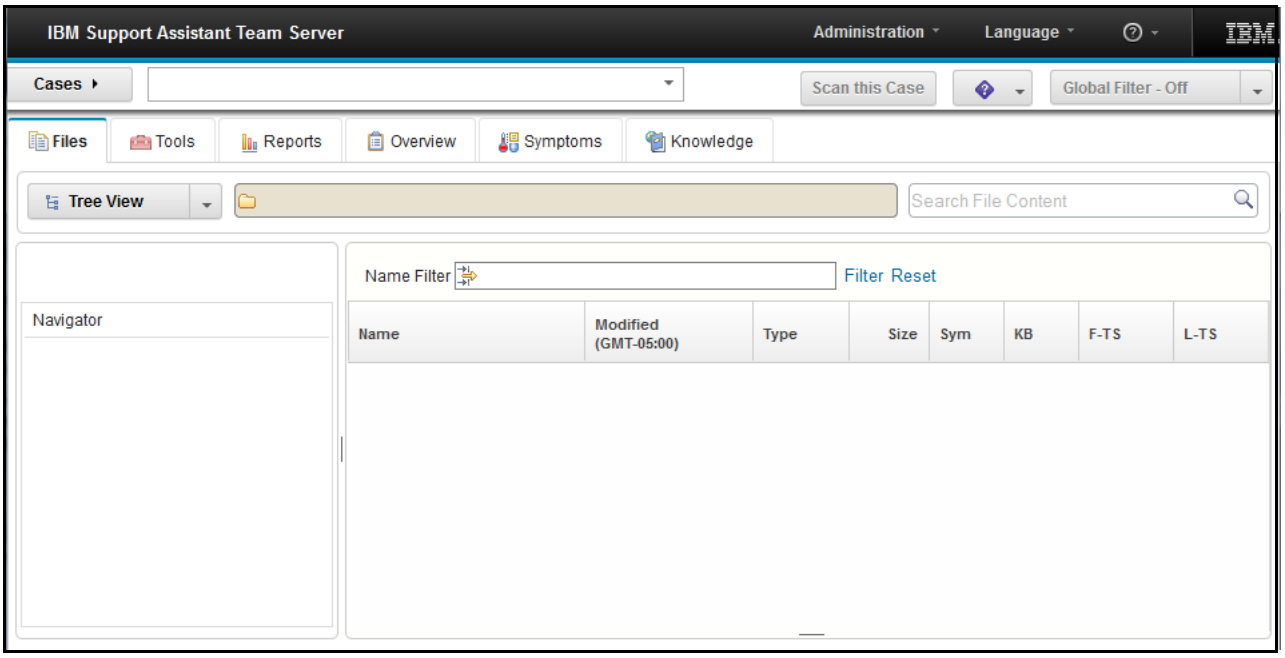


__ d. The next screen provides details about this security exception. Examine this screen; then, click **Confirm Security Exception**.

__ e. On the First Time Use pane, clear **Enable usage statistics**, and click **Submit**.



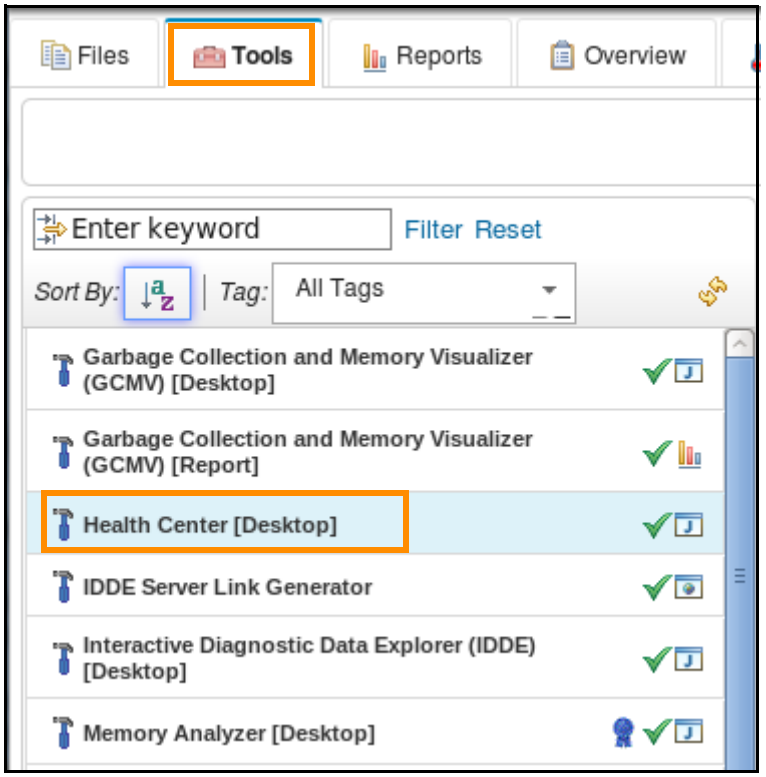
__ f. The IBM Support Assistant home page is displayed in the browser.



__ g. Bookmark IBM Support Assistant in the browser for future use.

__ 6. Start the Health Center tool.

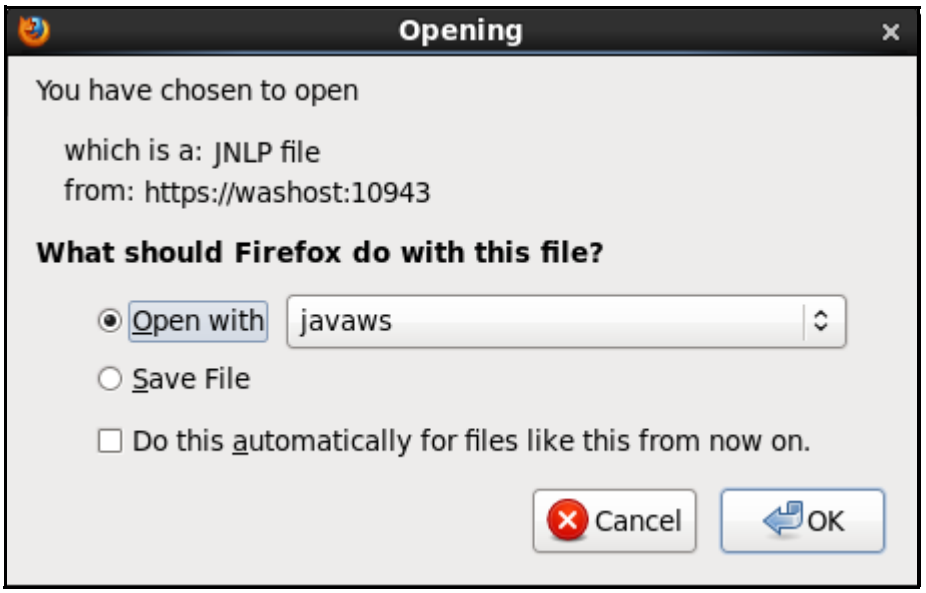
__ a. Click the **Tools** tab and click **Health Center**.



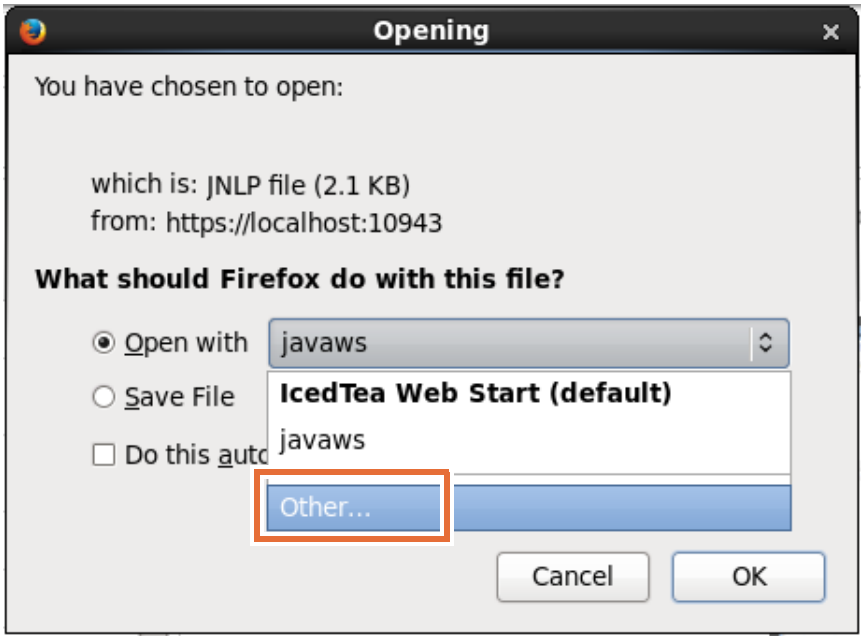
__ b. On the right pane, click **Launch**.

__ c. In the Run Tool window, click **Launch**.

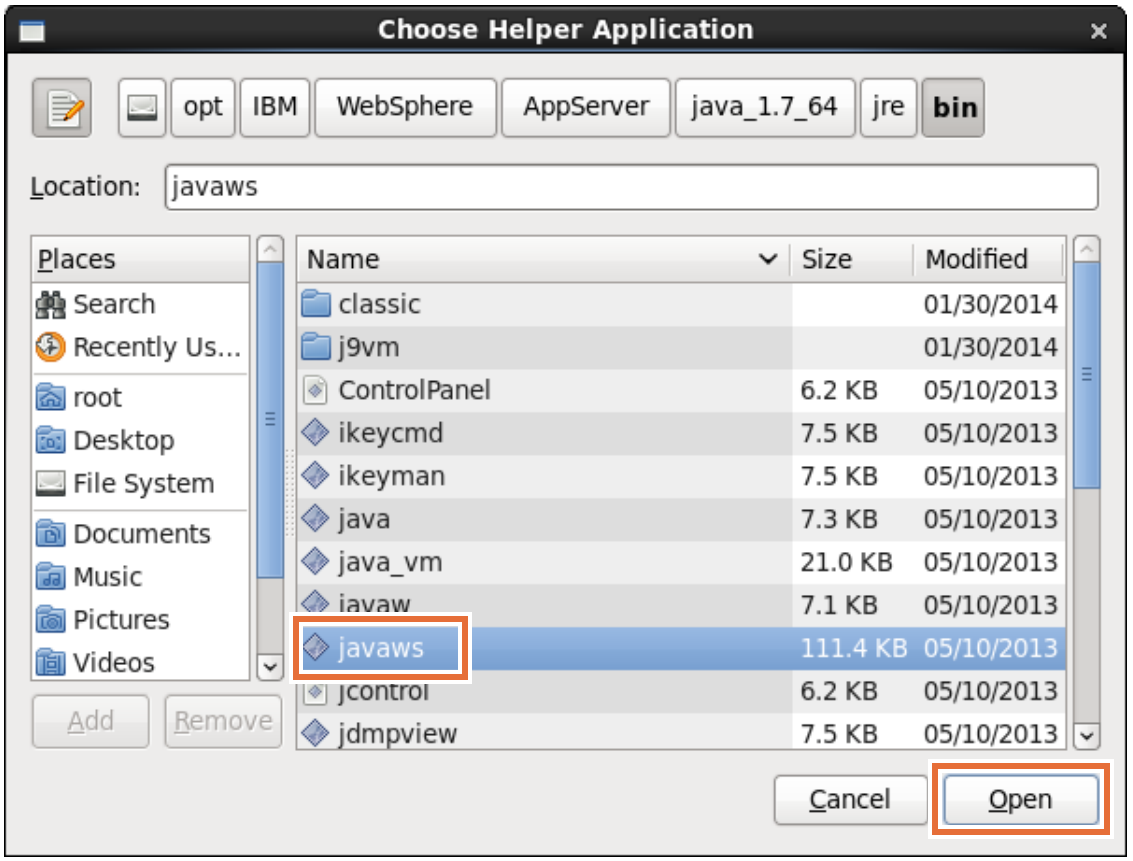
__ d. In the Opening window, select **Open with javaws**.



__ e. Expand the menu for the “Open with” option, and click **Other**.



__ f. Browse to `/opt/IBM/WebSphere/AppServer/java_1.7_64/jre/bin`, select **javaws** (Java Web Start) and click **Open.**

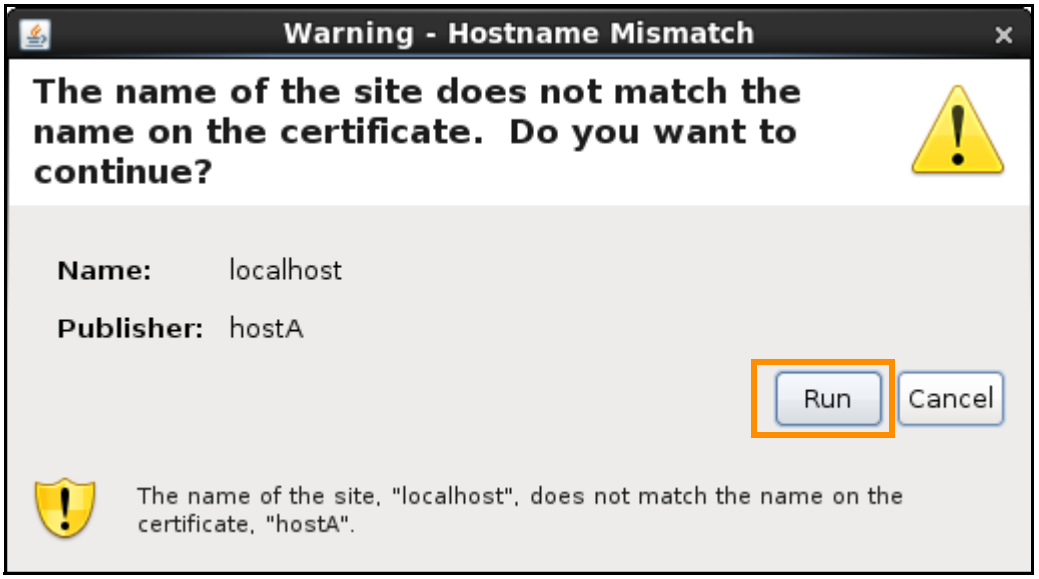


__ g. Back on the Opening pane, click **OK**.

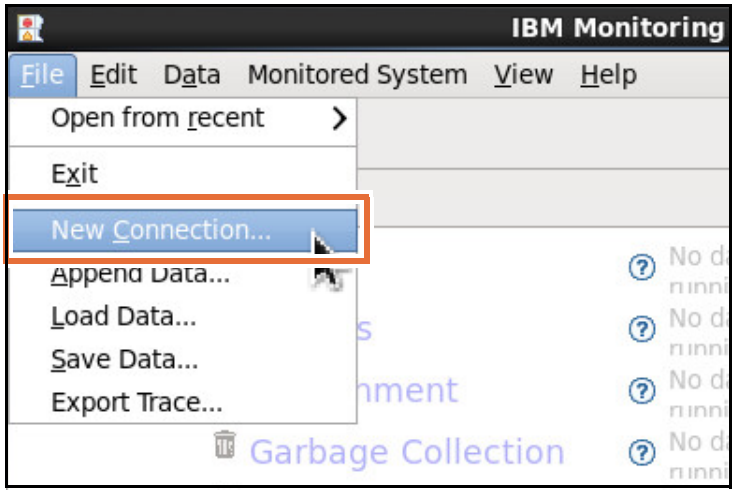
__ h. You might not see any of the following messages. If so, the certificates that are stored on your lab image are valid. In that case, proceed to step k.

__ i. On the Warning-Security pane, select the check box, and click **Yes.**

__ j. Wait for Java Web Start to download the tool, and click **Run**.

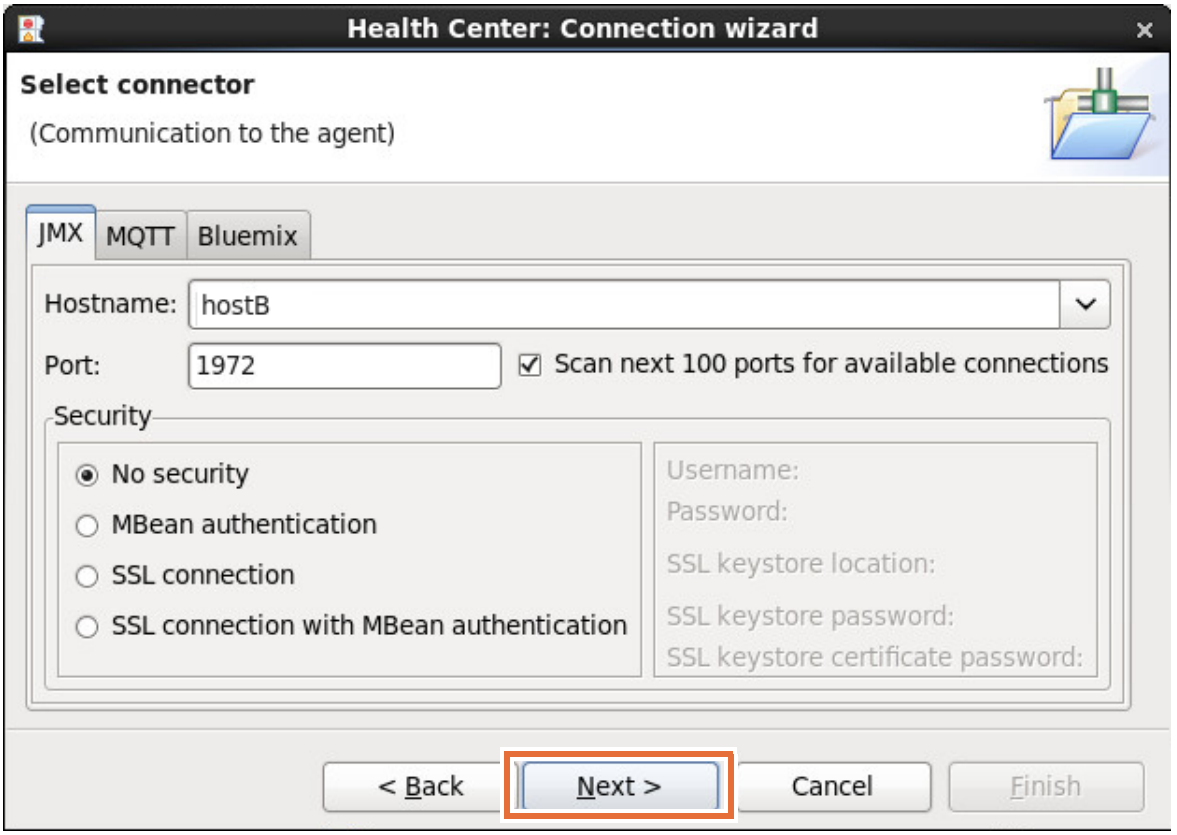


__ k. When the Health Center opens, click **File > New Connection**.

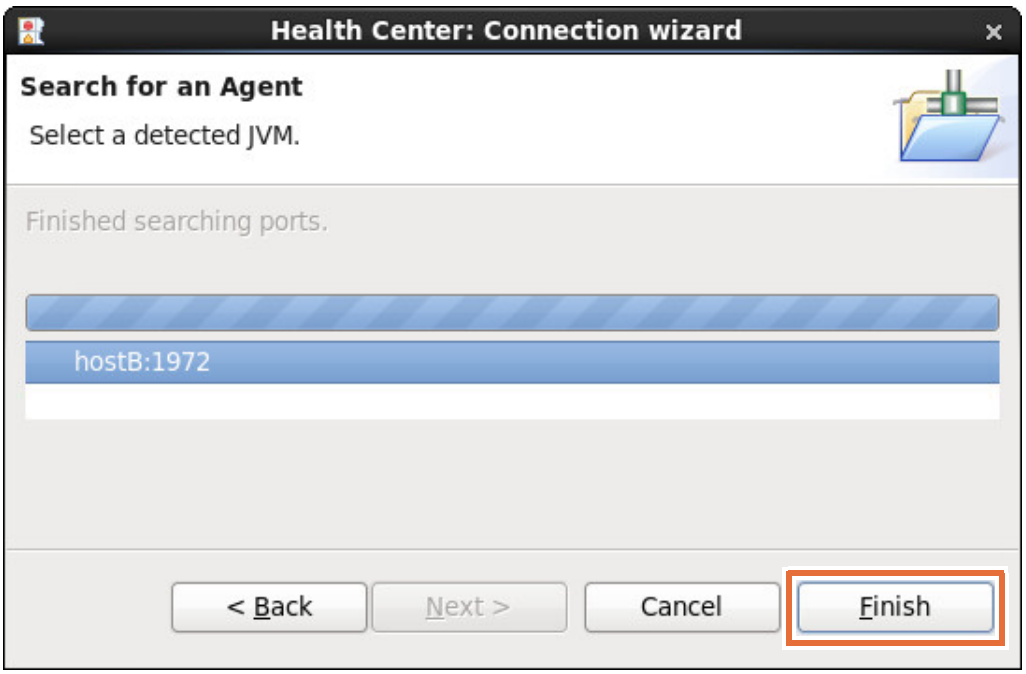


__ l. When the Connection wizard opens, click **Next**.

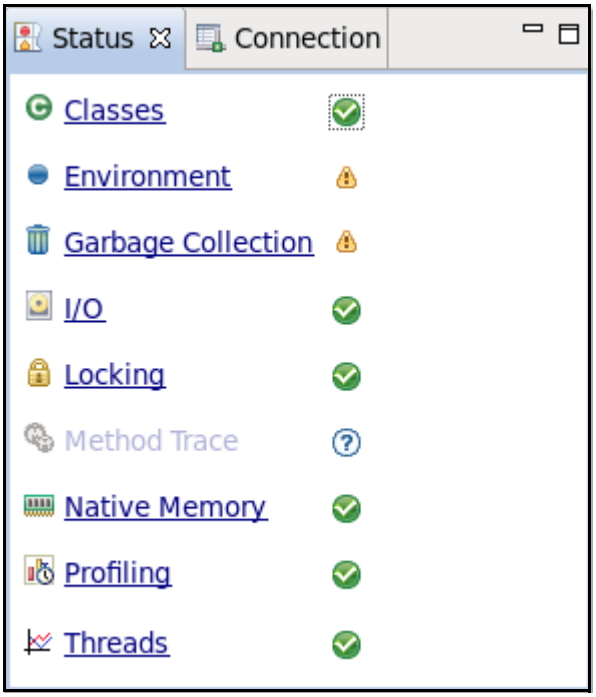
__ m. Make sure that the JMX tab is selected. Enter **hostB** for the hostname, use the default port 1972, and click **Next.**



__ n. Click **Finish** as soon as it detects the agent of TradeServer1 on port 1972. You do not have to wait for the search to complete.



__ o. Wait a few minutes while the Health Center retrieves runtime data from the server. Eventually the Health Center GUI is populated with server data. Examine the **Status** tab.



__ 7. Explore all of the monitored items that are listed by clicking their links.

__ a. Click **Classes** and explore the data that is shown for the Java classes.

__ b. Click **Native Memory** and explore the data that is shown for native memory.

__ c. Pay particular attention to the items whose status shows the yellow warning icon. Click the **Garbage Collection** link and explore the details about garbage collection.

__ 8. Prepare to load test the DayTrader.

__ a. On **hostA**, start a web browser, and go to **`http://hostB:9081/daytrader`**

__ b. Click the **Configuration** tab, and click the **Reset DayTrader** link. Keep this browser open as you are going to reset the DayTrader for each run of the load test.

__ c. Verify that the DayTrader reset completes successfully.

__ 9. Run the DayTrader baseline test plan.

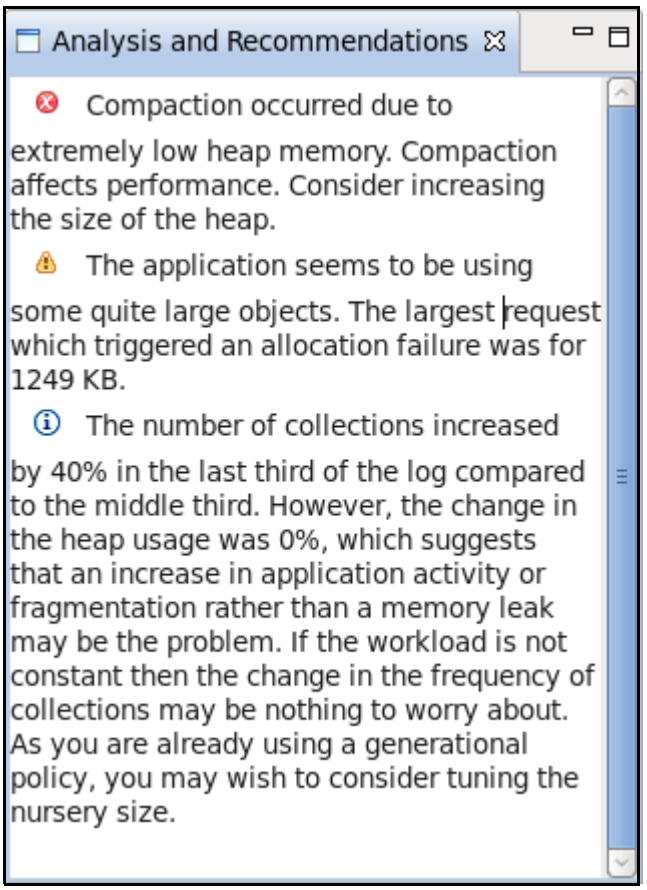
__ a. In Apache JMeter, set the loop count to **Forever.**

__ b. Click **Summary Report.**

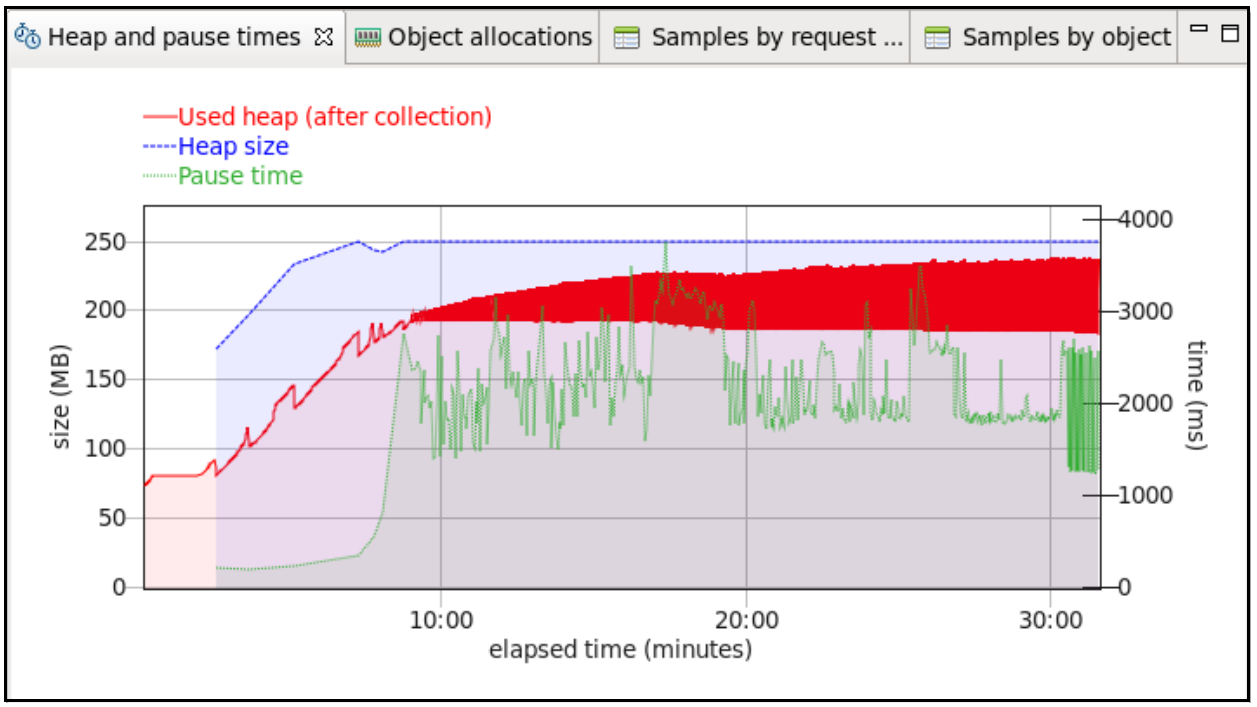
__ c. Click **Run > Start** (or click the green arrow) to start the test.

__ 10. Monitor the Garbage collection for several minutes. Your actual data might be slightly different from the screen captures that are shown here.

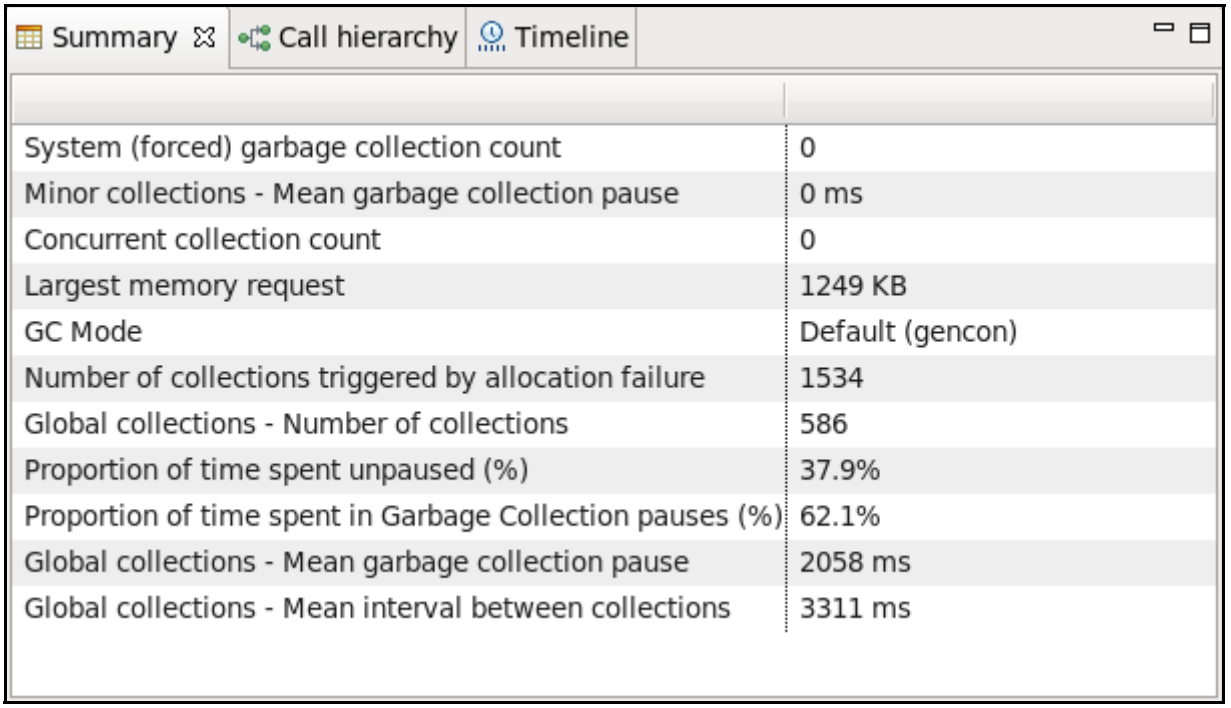
__ a. Examine the messages in the Analysis and Recommendations pane.



__ b. Examine the Heap and pause times tab.



__ c. Examine the Garbage collection summary.



System (forced) garbage collection count	0
Minor collections - Mean garbage collection pause	0 ms
Concurrent collection count	0
Largest memory request	1249 KB
GC Mode	Default (gencon)
Number of collections triggered by allocation failure	1534
Global collections - Number of collections	586
Proportion of time spent unpaused (%)	37.9%
Proportion of time spent in Garbage Collection pauses (%)	62.1%
Global collections - Mean garbage collection pause	2058 ms
Global collections - Mean interval between collections	3311 ms

__ d. Your performance results might be different from the numbers that are shown in the screen capture. In future exercises, you are going to apply some of the tuning recommendations that you saw in this exercise.

__ 11. Shutdown Health Center, ISA, JMeter, and TradeServer1.

__ a. In the Health Center window, click **File > Exit**.

__ b. Stop ISA 5 by returning to terminal window and entering: `./stop_isa.sh`

__ c. Stop the JMeter test plan, and exit JMeter by clicking **File > Exit**.

__ d. In the administrative console, stop **TradeServer1**.

Important

Stop IBM Support Assistant 5 when not in use

The IBM Support Assistant Team Server requires several Java processes to run on its host system. These processes can use much memory and processor resources. So, it is a good idea to stop IBM Support Assistant when it is not in use.

End of exercise

Exercise review and wrap-up

In this exercise, you learned how to use performance monitoring tools and performance advisor facilities that are available in WebSphere Application Server and IBM Support Assistant. You gained some experience monitoring performance with the Tivoli Performance Viewer. You used the performance advisors, including Runtime Performance Advisor and Performance Advisor in Performance Viewer to gather advice about improving performance. Finally, you used the Health Center tool in the IBM Support Assistant to monitor a running JVM.

Exercise 6. Exploring GC policies and monitoring JVM performance

What this exercise is about

The purpose of this exercise is to explore different garbage collection policies and monitor their performance during load testing.

What you should be able to do

At the end of this exercise, you should be able to:

- Enable verbose GC for an application server
- Manually view verbose GC logs
- Configure the optthruput GC policy
- Configure the optavgpause GC policy
- Use Apache JMeter to load test an application server
- Use IBM Garbage Collection and Memory Visualizer (GCMV) to analyze and compare GC data

Introduction

When the JVM cannot allocate an object from the heap because of lack of contiguous space, a memory allocation fault occurs, and the garbage collector is called. The first task of the garbage collector is to collect all garbage that is in the heap. This process starts when any thread calls the garbage collector either indirectly as a result of allocation failure, or directly by a specific call to **`System.gc()`**.

In this exercise, you configure and explore two different GC modes or policies. The default GC policy, gencon, attempts to provide both optimized throughput and response times. In addition, the gencon GC policy virtually eliminates heap fragmentation by using a partitioned heap. The former default policy, opttruput, attempts to optimize throughput for your applications. Another GC policy, optavgpause, attempts to minimize pause times to improve response times for your application.

Tuning the JVM properly is a process that takes time and must be tailored to your application. The JVM must be tuned in two iterative steps over a testing cycle.

- Step 1: Select and tune the GC policy
- Step 2: Tuning the heap size

Tuning the JVM is an iterative process. Change one parameter at a time. Measure the results of each tuning action that is taken.

In this exercise, you begin the first step: Select and tune the GC policy. In the next exercise, you do the second step: Tuning the heap size.

Requirements

Exercise 3: Installing DayTrader and **Exercise 4: Use Apache JMeter to load test DayTrader** must be successfully completed before you can complete this exercise. At the end of Exercise 4, you created an Apache JMeter test plan, **`/usr/labfiles/Test_Plans/DayTrader_baselineXX_test.jmx,`** which you are going to use in this exercise.

Exercise instructions

Preface

- Most steps in this exercise are done on **hostA** unless otherwise specified.

Note

<*profiles_root*> = /opt/IBM/WebSphere/AppServer/profiles

Section 1: Start the servers and verify that verbose GC is enabled

You use the administrative console to enable verbose GC for TradeServer1.

__ 1. On **hostA**, ensure that the deployment manager is started.

__ a. In a terminal window, change directory to:

<*profiles_root*>/Dmgr/bin

__ b. Enter the command:
./serverStatus.sh dmgr -user wasadmin -password web1sphere

__ c. If dmgr is stopped, start it with the command:
./startManager.sh

__ 2. On **hostB**, ensure that the node agent is started and TradeServer1 is stopped.

__ a. In a terminal window, change directory to

<*profiles_root*>/profile1/bin

__ b. Enter the command:
./serverStatus.sh -all -user wasadmin -password web1sphere

__ c. If the node agent is stopped, start it with command:
./startNode.sh

__ d. If TradeServer1 is started, stop it with command:
./stopServer.sh TradeServer1 -user wasadmin -password web1sphere

__ e. If server1 is started, stop it with command:
./stopServer.sh server1 -user wasadmin -password web1sphere

__ 3. Use the administrative console to enable verbose GC for TradeServer1.

__ a. Log in to the administrative console as user ID **wasadmin** and password: **web1sphere**

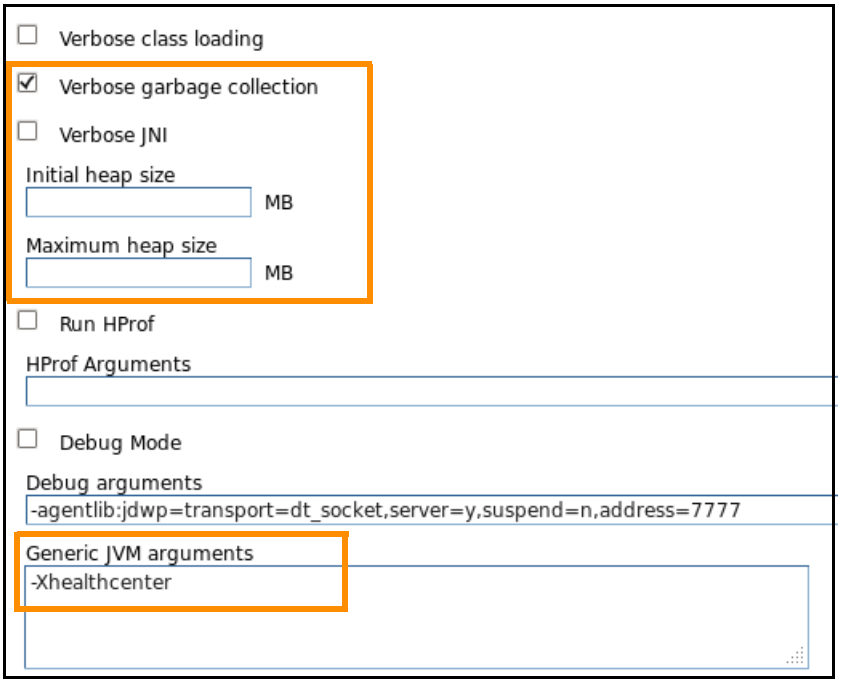
__ b. Click **Servers > Server Types > WebSphere application servers > TradeServer1**.

__ c. On the configuration tab for TradeServer1, under Server Infrastructure, click **Java and process management**.

__ d. Click **Process definition > Java Virtual Machine**.

__ e. On the JVM configuration tab for TradeServer1, select the **Verbose garbage collection** check box.

__ f. If necessary, set the initial heap and maximum heap sizes back to the default by clearing the fields for each.



__ g. Verify that **`-Xhealthcenter`** is listed as a generic JVM argument. Add it if necessary.

__ h. Click **OK**.

__ i. Click the link to **Save** directly to the master configuration.

__ j. Wait for the node to synchronize and click **OK**.

__ 4. On **hostC**, restore the tradedb database.

__ a. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**

__ b. Enter the command:
```
su - db2inst1
```

__ c. Run the following script to restore the tradedb database.
```
/usr/labfiles/db2scripts/db2restore.sh
```

__ d. Run the following script to unquiesce the database.
```
/usr/labfiles/db2scripts/db2unquiesce.sh
```

__ 5. Examine the verbose GC data for TradeServer1.

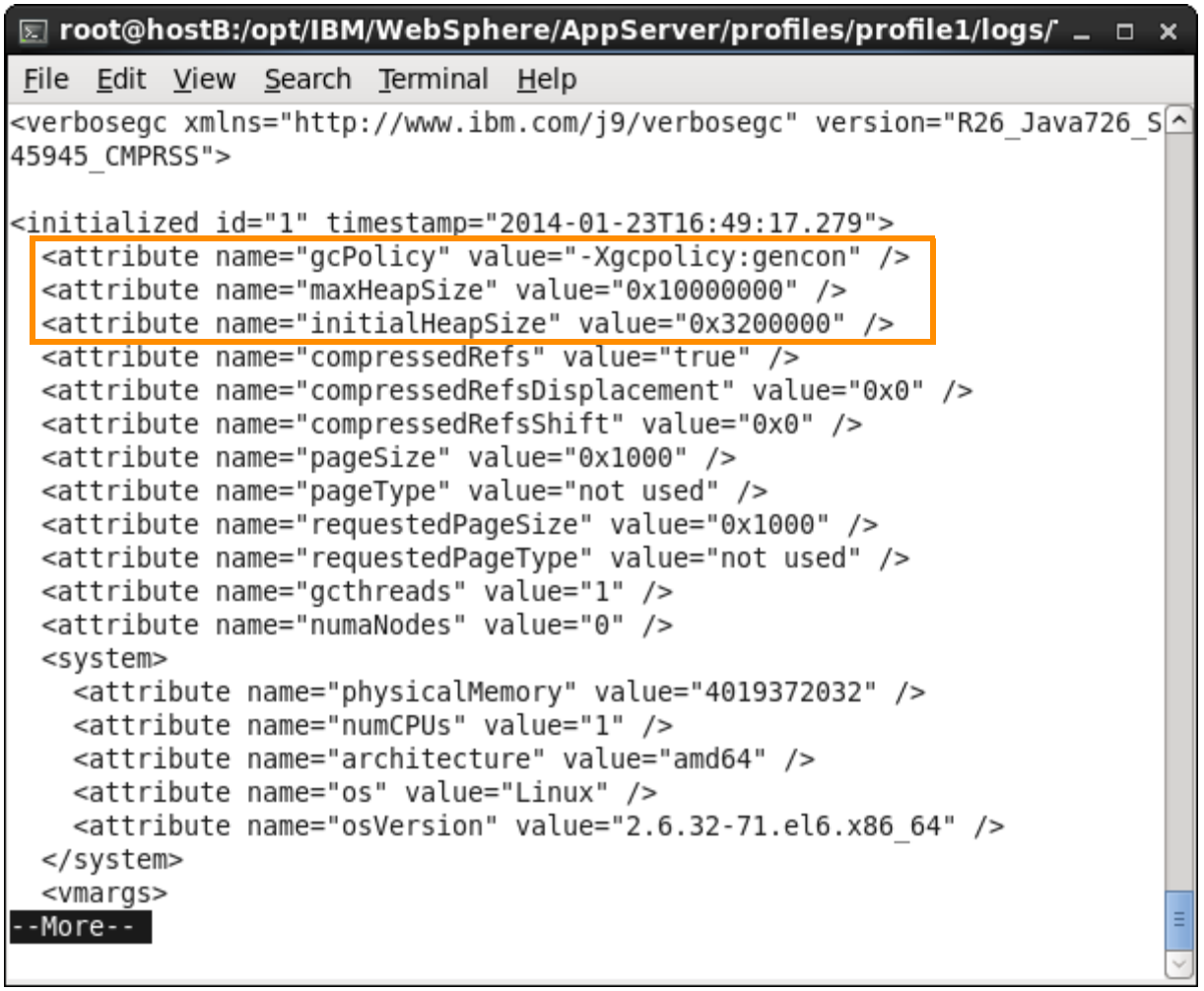
__ a. Open an ssh terminal to **hostB**, and navigate to
`<profile_root>/profile1/logs/TradeServer1`

__ b. Remove all the files in the directory: **`rm *`**

__ c. Start **TradeServer1** from the administrative console and wait for it to start successfully.

__ d. Back in the ssh terminal window, enter the command:
```
cat native_stderr.log | more
```

__ e. At the top of the log, you should see something like the following.



__ f. Notice that the GC policy is generational concurrent (gencon), which is the default policy. Notice that maximum heap size is 256 MB (0x10000000 bytes) and initial heap size is 50 MB (0x3200000 bytes). These values are the default heap sizes for Linux OS.

__ g. Scroll down to view 1 or 2 allocation failures and garbage collection cycles. You are going to use the GCMV tool in a future section to look at the garbage collection data in more detail.

__ h. Close the ssh terminal window.

Section 2: Load test TradeServer1 with the gencon policy

In this section and the following section, you are going to attempt to drive the same load against the TradeServer1 and gather comparable verbose GC log data for the configured GC policy.

__ 1. Start Apache JMeter.

__ a. On **hostA**, open a terminal window and enter the command:
`cd /opt/apache-jmeter/bin`

__ b. Enter the command:
`./jmeter.sh &`

__ 2. Run the JMeter baseline test plan.

__ a. In Apache JMeter, click **File > Open**, and navigate to **`/usr/labfiles/Test_Plans/DayTrader_baselineXX_test.jmx`**

__ b. Click **Open**.

__ c. In the JMeter tree, click **Thread Group** and change the Loop Count to **100**.

__ d. In JMeter, click **Run >Start** (or click the green arrow) to start the test.

__ 3. The first run is a warm-up run. Reset DayTrader. In the JMeter tree, click **Thread Group** and change the Loop Count to **500**, and run the test load.

__ 4. After the test run, stop TradeServer1. Record your performance results (average response time and throughput) in the table.

Table 7: Test run results with -Xmx=512m -Xgcpolicy=gencon

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

__ 5. Gather the verbose GC log for these load tests.

__ a. Open an ssh terminal window for **hostB**, and navigate to **`<profile_root>/profile1/logs/TradeServer1`**

__ b. Rename the **`native_stderr.log`** file **`genconGC.log`** and FTP it from hostB to **`/usr/labfiles/case1`** on hostA

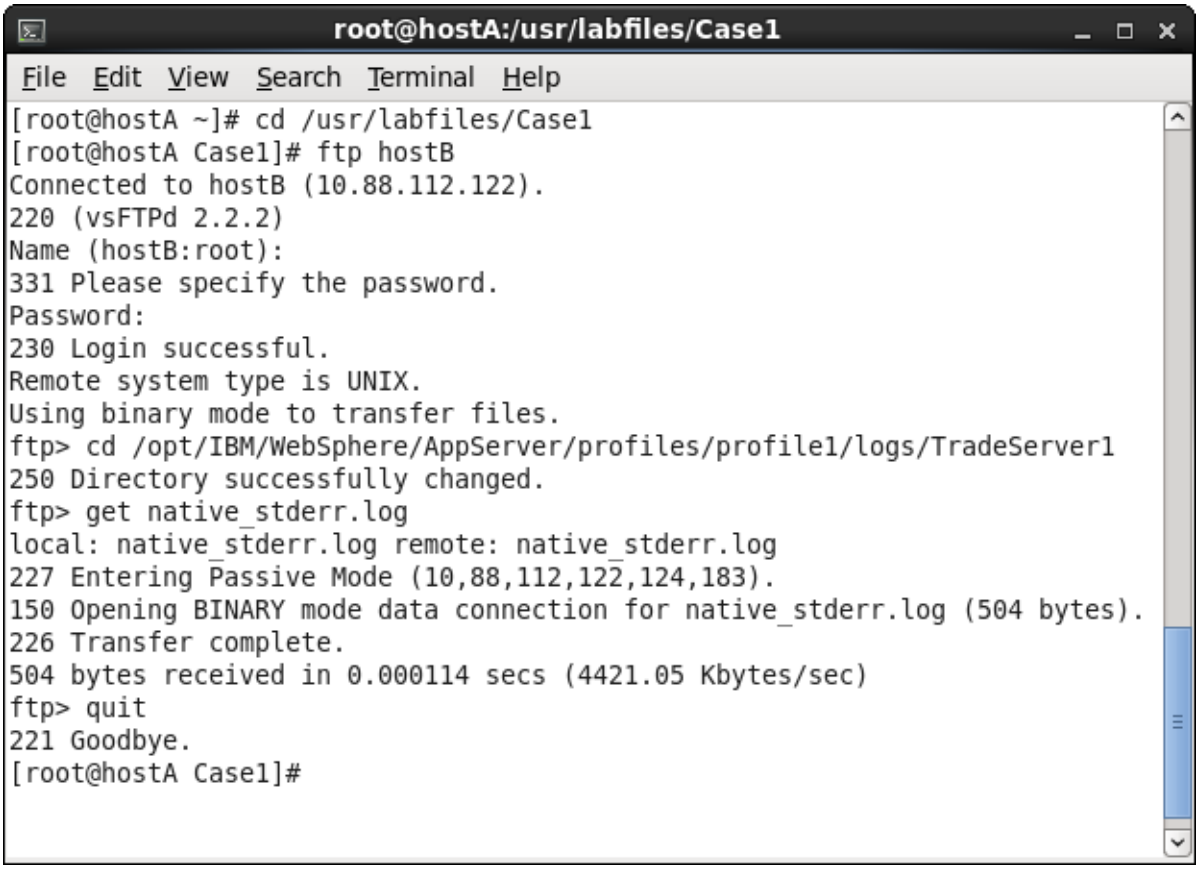
Information

Using FTP

If you are not familiar with using FTP, follow these steps.

Open a terminal window on **hostA**, and type the following commands:

1. `cd /usr/labfiles/Case1`
2. `ftp hostB`
3. When prompted for name, enter: **`root`**
4. When prompted for password, enter: **`web1sphere`**
5. `cd /opt/IBM/WebSphere/AppServer/profiles/profile1/logs/TradeServer1`
6. `get genconGC.log`
7. `quit`



Section 3: Load test TradeServer1 with the opttruput policy

In this section and the previous section, you are going to attempt to drive the same load against the TradeServer1 and gather comparable verbose GC log data for the configured GC policy.

__ 1. Change the GC policy for TradeServer1 to optimize for throughput (opttruput).

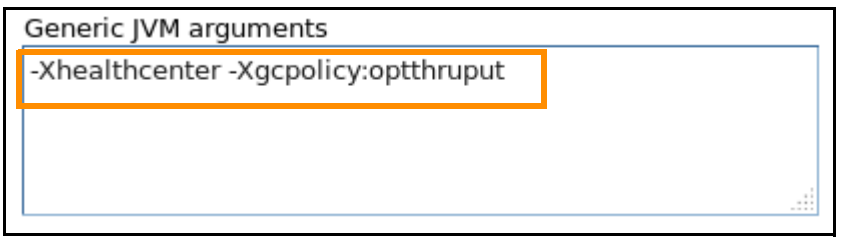
__ a. Log in to the administrative console as user ID **`wasadmin`** and password: **`web1sphere`**

__ b. Click **Servers > Server Types > WebSphere application servers > TradeServer1**.

__ c. On the configuration tab for TradeServer1, click **Java and process management**.

__ d. Click **Process definition > Java Virtual Machine**.

__ e. In the Generic JVM argument section, add **`-Xgcpolicy:optthruput`**.



__ f. Click **OK**.

__ g. Click the link to **Save** directly to the master configuration.

__ h. Wait for the node to synchronize and click **OK**.

__ 2. On **hostC**, restore the tradedb database.

__ a. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ b. Enter the command: **`su - db2inst1`**

__ c. Run the following script to restore the tradedb database.

`/usr/labfiles/db2scripts/db2restore.sh`

__ d. Run the following script to unquiesce the database.

`/usr/labfiles/db2scripts/db2unquiesce.sh`

__ 3. Gather a clean log of verbose GC data for TradeServer1.

__ a. Open an ssh terminal to **hostB**, and navigate to **`<profile_root>/profile1/logs/TradeServer1`**

__ b. Remove all the files in the directory: **`rm *`**

__ c. Start **TradeServer1** from the administrative console and wait for it to start successfully.

__ 4. Run the test plan.

__ a. In the JMeter tree, click **Thread Group** and change the Loop Count to **100**.

__ b. In JMeter, click **Run > Start** (or click the green arrow) to start the test.

__ 5. The first run is a warm-up. Reset DayTrader. In the JMeter tree, click **Thread Group** and change the Loop Count to **500**, and run the test load.

__ 6. After the test run, stop TradeServer1. Record your performance results (average response time and throughput) in the table.

Table 8: Test run results with -Xmx=512m -Xgcpolicy=optthruput

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

__ 7. Gather the verbose GC log for these load tests.

__ a. Open an ssh terminal window for hostB, and navigate to **`<profile_root>/profile1/logs/TradeServer1`**

__ b. Rename the **native_stderr.log** file **optthruputGC.log** and FTP it from **hostB** to **`/usr/labfiles/case1`** on **hostA**.

Section 4: Load test TradeServer1 with the optavgpause policy

In this section and the previous section, you are going to attempt to drive the same load against the TradeServer1 and gather comparable verbose GC log data for the configured GC policy.

__ 1. Change the GC policy for TradeServer1 to optimize for throughput (opttruput).

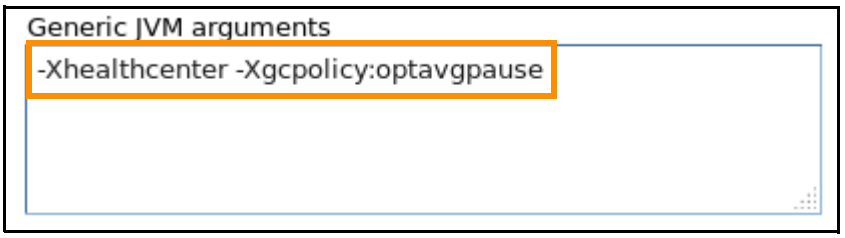
__ a. Log in to the administrative console with the user ID **wasadmin** and password **web1sphere**

__ b. Click **Servers > Server Types > WebSphere application servers > TradeServer1**.

__ c. On the configuration tab for TradeServer1, click **Java and process management**.

__ d. Click **Process definition > Java Virtual Machine**.

__ e. In the Generic JVM argument section, add a different **-Xgcpolicy:optavgpause**.



__ f. Click **OK**.

__ g. Click the link to **Save** directly to the master configuration.

__ h. Wait for the node to synchronize and click **OK**.

__ 2. On **hostC**, restore the tradedb database.

__ 3. Gather a clean log of verbose GC data for TradeServer1.

__ a. Open an ssh terminal to **hostB**, and navigate to **<*profile_root*>/profile1/logs/TradeServer1**

__ b. Remove all the files in the directory: **rm ***

__ c. Start **TradeServer1** from the administrative console and wait for it to start successfully.

__ 4. Run the test plan.

__ a. In the JMeter tree, click **Thread Group** and change the Loop Count to **100**.

__ b. In JMeter, click **Run > Start** (or click the green arrow) to start the test.

__ 5. The first run is a warm-up. Reset DayTrader. In the JMeter tree, click **Thread Group** and change the Loop Count to **500**, and run the test load.

__ 6. After the test run, stop TradeServer1. Record your performance results (average response time and throughput) in the table.

Table 9: Test run results with -Xmx=512m -Xgcpolicy=optavgpause

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

__ 7. Gather the verbose GC log for these load tests.

__ a. Open an ssh terminal window for hostB, and navigate to **`<profile_root>/profile1/logs/TradeServer1`**

__ b. Rename the **`native_stderr.log`** file **optavgpauseGC.log** and FTP it from hostB to **`/usr/labfiles/case1`** on hostA.

Section 5: Use GCMV to compare the different GC policies

This section serves as an introduction to using the GCMV desktop tool to examine charts and tables of information that it generates by analyzing raw verbose GC log data.

__ 1. Start the IBM Support Assistant.

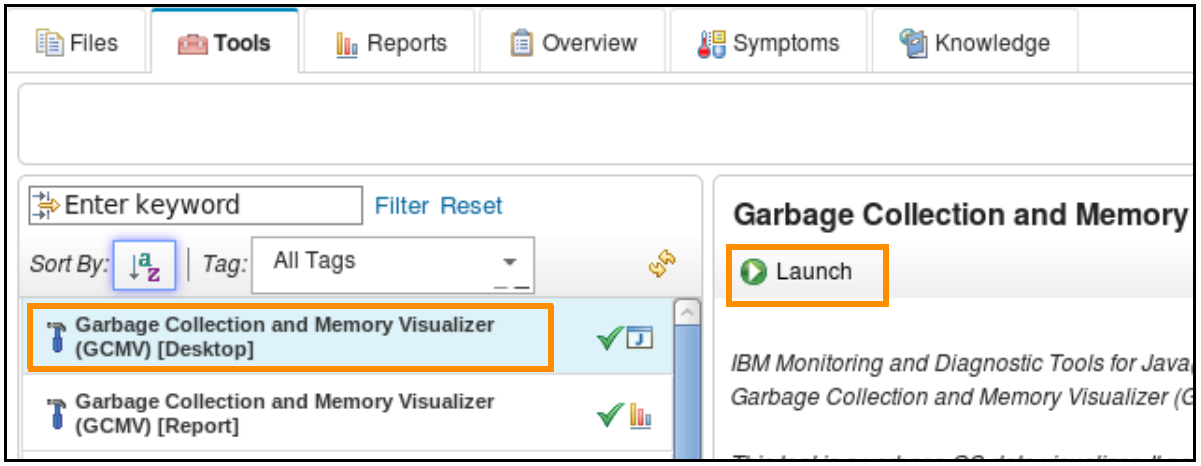
__ a. On **hostA**, start ISA from a terminal window by navigating to **`/opt/ibm/ISA5`**, and entering: **`./start_isa.sh`**

__ b. Minimize, but do not close the terminal window.

__ c. Open a Firefox web browser and use the ISA bookmark, or type the address: **`http://localhost:10911/isa5`**

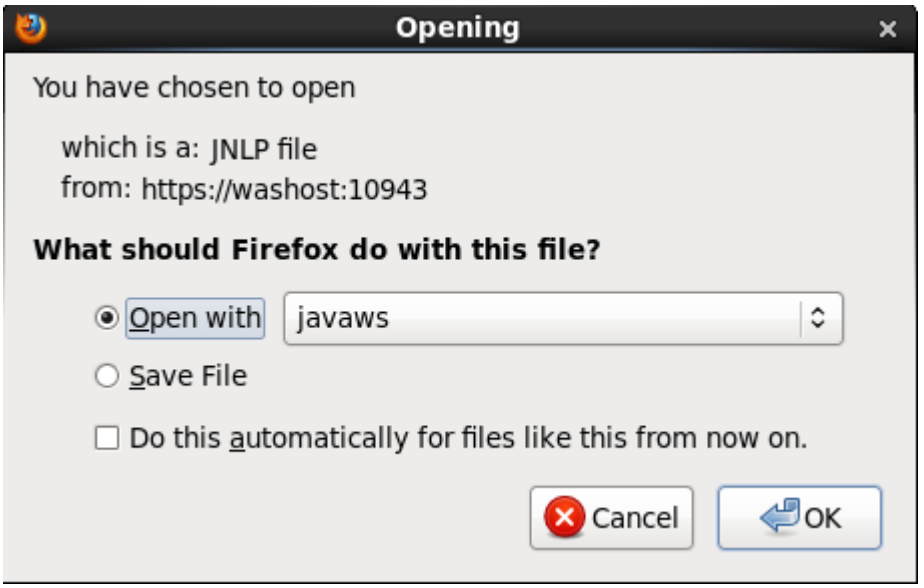
__ 2. Run the GCMC desktop tool.

__ a. In the **Tools** tab of IBM Support Assistant, select **GCMV [desktop]** and click **Launch**.



__ b. In the Run Tool window, click **Submit**.

__ c. In the Opening window, select **Open with javaws**.



__ d. For the “Open with” option, select menu item **Other**. Browse to **`/opt/IBM/WebSphere/Appserver/java_1.7_64/jre/bin`** and select **javaws**, and click **Open**.

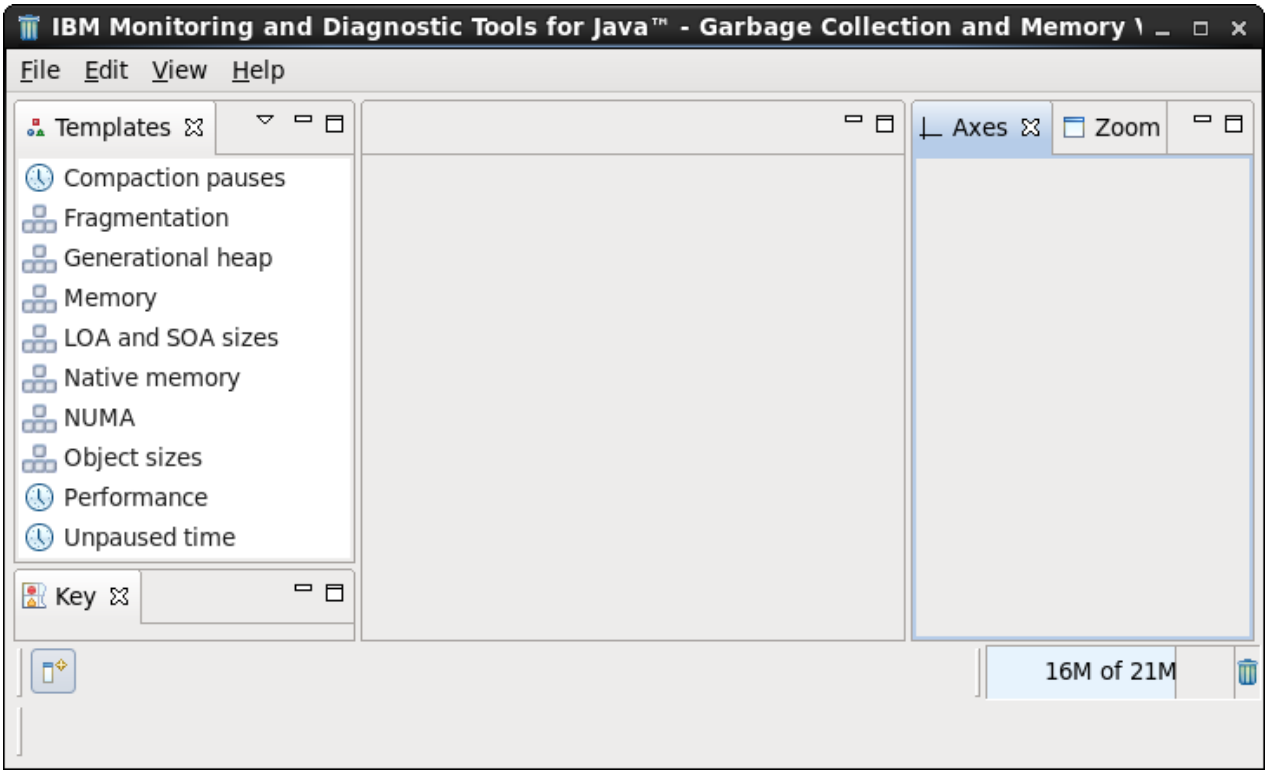
__ e. Click **OK**.

Note

Using the Java 7 version of Java Web Start

For a few browsers and versions, some of the ISA desktop tools must use the Java 7 version of Java Web Start. So, for the GCMV desktop tool, make sure that you use Java 7 if you are using a Firefox browser.

__ f. Wait for Java Web Start to download the desktop tool. You should see the tool in its own browser as show here.



__ 3. Load the genconGC.log file into the GCMV tool.

__ a. Click **File > Load File** and browse to **`/usr/labfiles/Case1`**

__ b. Select **genconGC.log**, and click **OK**.

__ c. Wait for the tool to process the file.

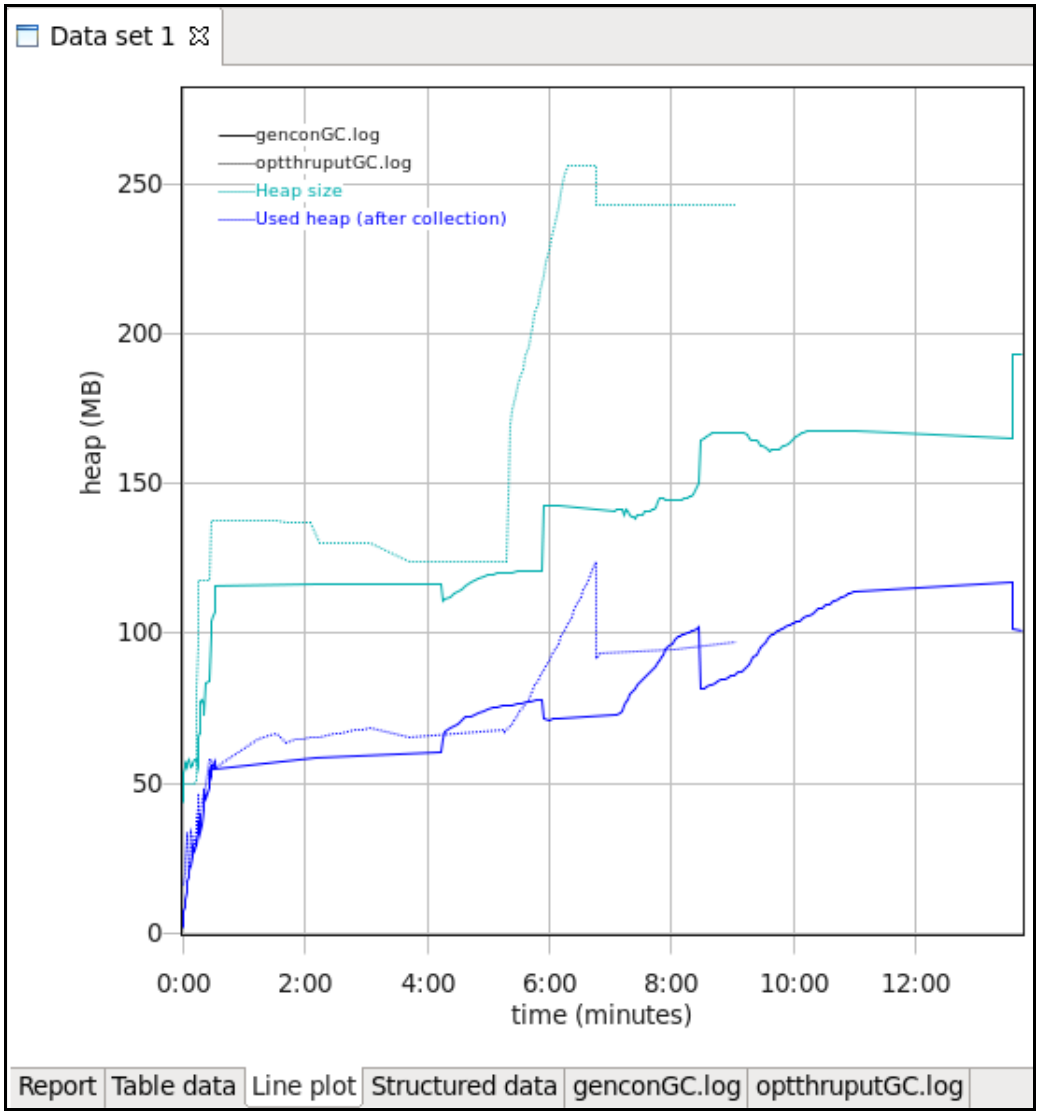
__ 4. Load the optthruputGC.log file into the GCMV tool.

__ a. Click **File > Compare File** and browse to **`/usr/labfiles/Case1`**

__ b. Select **optthruputGC.log**, and click **OK**.

__ c. Wait for the tool to process the file.

__ d. Notice that the GCMV desktop tool does a comparison of the two files in all of the views and templates. Click the different tabs such as **Report**, **Table data**, and **Line plot**.



__ e. Click the **genconGC.log** and **optthruput.log** tabs to view the raw data in the native_stderr.logs for each GC policy that you tested.

Information

Interpreting the verbose GC data for the gencon policy

Notice that the type is **"nursery",** which signifies that the gencon GC policy is in use. The **intervalms** provides the amount of time in milliseconds (ms) between allocation failures.**minimum requested_bytes** is the size of the object in bytes that caused the allocation failure.

Even though this data shows the three **<tenured .../tenured>** sections the same as the optthruput data did, you want to pay attention to the three **<nursery ...\>** sections because gencon allocates from the nursery, not the tenured part of the heap. At the time of the allocation failure in this example, 0% of the nursery was available. Garbage collection recovered 55% of the nursery.

Within the **<gc...\gc>** section you see data about the garbage collection process. **type="scavenger"** means that the nursery was collected by using a special copy algorithm rather than mark and sweep. **flipped objectcount** refers to the number of objects that were copied from the nursery's allocate subspace to its survivor subspace. The term, **tenured objectcount,** refers to the number of objects that were promoted (moved) from the nursery to the tenured space. The term, **scavenger tiltratio,** refers to what portion of the nursery is used for the allocate subspace. (50 means that the allocate and survivor subspaces are equal in size. However, this ratio is dynamically adjusted upward if mostly short-lived objects are allocated.) The term **tenureage** refers to the number of "flips" an object must survive before it is promoted to the tenured space.

Finally, time **totalms** gives the total time in milliseconds required to process the allocation failure.

Information

Interpreting the verbose GC data for the optthruput policy

Notice that the type is **"tenured"** which signifies that the optthruput GC policy is in use. The **intervalms** provides the amount of time in milliseconds (ms) between allocation failures. The term **minimum requested_bytes** is the size of the object in bytes that caused the allocation failure.

Notice that there are three **<tenured freebytes...</tenured>** sections. There is one before the garbage collection cycle, one after garbage collection, and one after the requested heap space is allocated. The term **soa** stands for small object area, a part of the heap on which small objects are allocated. The term **loa** stands for large object area. In this example, at the time of the allocation failure 0% of the **soa** was available. Garbage collection recovered 64% of the **soa**.

Within the **<gc .../gc>** section you see data about the garbage collection process. **Global** means that the entire heap was collected. Time in milliseconds is given during the mark, sweep, and compact phases (**compact="0.000"** means that no compaction occurred during this garbage collection).

Finally, time **totalms** gives the total time in milliseconds required to process the allocation failure.

__ f. The Report tab and the Structured data tab provide helpful data for comparing the performance of the two GC policies. Click the **Report** tab, and examine the tuning recommendations. Scroll down and examine the **Summary** table that compares GC metrics in side by side columns for each policy.

Summary

Variant	genconGC.log	optthruputGC.log
Concurrent collection count	4	N/A
Forced collection count	3	3
GC Mode	gencon	optthruput
Global collections - Mean garbage collection pause (ms)	301	N/A
Global collections - Mean interval between collections (ms)	102021	N/A
Global collections - Number of collections	8	N/A
Global collections - Total amount tenured (MB)	457	N/A
Largest memory request (bytes)	33800	2097160

__ g. Because you are comparing two different GC policies, many metrics (variants) in one do not have equivalent metrics in the other. Scroll down to the end of the table to see several metrics that are common to both polices.

Nursery collections - Total amount tenured (MB)	117	N/A
Proportion of time spent in garbage collection pauses (%)	1.82	6.13
Proportion of time spent unpaused (%)	98.18	93.87
Rate of garbage collection (MB/minutes)	499	716

__ h. One of the key metrics for comparing the performance of two different GC policies is the proportion of time spent unpaused. In this table, you see that the gencon policy appears to show slightly better performance than the optthruput policy. Your actual results might be different, but they are likely to also show better performance for the gencon policy.

__ 5. Load the optavgpauseGC.log file into the GCMV tool.

__ a. Click **File > Compare File** and browse to **`/usr/labfiles/Case1`**

__ b. Select **optavgpauseGC.log**, and click **OK**.

__ c. Wait for the tool to process the file.

__ 6. Compare all three GC policies.

__ a. Click the **Reports** tab to see what tuning recommendations are provided for the optavgpause GC policy.

__ b. You might see that optthruput and optavgpause have similar tuning recommendations.

optavgpauseGC.log	⚠ At one point 8900 objects were queued for finalization. Using finalizers is not recommended as it can slow garbage collection and cause wasted space in the heap. Consider reviewing your application for occurrences of the finalize() method. You can use the ISA Tool Add-on, IBM Monitoring and Diagnostic Tools for Java - Memory Analyzer to list objects that are only retained through finalizers. ⚠ Your application is allocating many large objects, which affects performance.Consider increasing -Xloaminimum and -Xloainitial.

Information

Large object area (LOA) GC tuning parameters

The LOA boundary is calculated when the heap is initialized, and recalculated after every garbage collection. The size of the LOA can be controlled by using command line options: **`-Xloainitial`** and **`-Xloamaximum`**.

`-Xloainitial` specifies the initial percentage (between 0 and 0.95) of the current tenure space that is allocated to the large object area (LOA). The default value is 0.05.

`-Xloaminimum` specifies the minimum percentage (between 0 and 0.95) of the current tenure space that is allocated to the large object area (LOA). The LOA does not shrink to less than this value. The default value is 0.

__ c. Scroll down to the **Summary** table and compare the metrics for each GC policy.

Summary

Variant	genconGC.log	optavgpauseGC.log	optthruputGC.log
Concurrent collection count	4	3	N/A
Forced collection count	3	3	3
GC Mode	gencon	optavgpause	optthruput

__ d. Again, scroll down to the end of the table to see several metrics that are common to all three polices.

Proportion of time spent in garbage collection pauses (%)	1.82	8.44	6.13
Proportion of time spent unpaused (%)	98.18	91.56	93.87
Rate of garbage collection (MB/minutes)	499	698	716

__ e. In this table, you see that the gencon policy (column 2) appears to show slightly better performance than the optavgpause policy (column 4). But comparing opttruput (column 3) and optavgpause (column 4), you might see a higher percentage of unpaused time for the optavgpause policy. Your actual results might be different, but they are likely to also show better performance for the gencon policy.

__ 7. Exit the GCMV tool.

__ a. In the GCMV browser, click **File > Exit**.

__ 8. Change the GC policy for TradeServer1 back to the default of gencon.

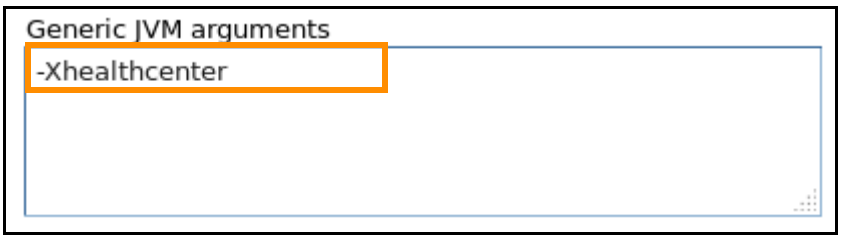
__ a. Log in to the administrative console as user ID **`wasadmin`** and password: **`web1sphere`**

__ b. Click **Servers > Server Types > WebSphere application servers > TradeServer1**.

__ c. On the configuration tab for TradeServer1, click **Java and process management**.

__ d. Click **Process definition > Java Virtual Machine**.

__ e. In the Generic JVM argument section, delete **`-Xgcpolicy:optavgpause`**.



__ f. Click **OK**.

__ g. Click the link to **Save** directly to the master configuration.

__ h. Wait for the node to synchronize and click **OK**.

Section 6: Cleanup

If this exercise is the last exercise you do for several hours, you should stop the WebSphere servers.

__ 1. On **hostA**, stop the Deployment Manager.

__ a. Open a terminal window and enter the command:
`cd <profiles_root>/Dmgr/bin`

__ b. Enter the command:
`./stopManager.sh -username wasadmin -password web1sphere`

__ 2. On **hostB**, stop the node agent and TradeServer1.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
`cd <profiles_root>/profile1/bin`

__ c. Enter the command:
`./stopNode.sh -username wasadmin -password web1sphere`

__ d. Enter the command:
`./stopServer.sh TradeServer1 -username wasadmin -password web1sphere`

Exercise review and wrap-up

In this exercise, you learned how to enable verbose GC for an application server and manually view verbose GC logs. You learned how to configure the different GC policies such as optthruput, optavgpause, and gencon. You used the IBM Garbage Collection and Memory Visualizer (GCMV) desktop tool to analyze and compare GC data.

Exercise 7. Tuning the JVM

What this exercise is about

This exercise covers tuning the application server JVM. You configure and load test different heap sizes for TradeServer1. You use monitoring and analysis tools such as Health Center and Garbage Collection and Memory Visualizer (GCMV) to gather tuning recommendations for the JVM.

What you should be able to do

At the end of this exercise, you should be able to:

- Configure the initial and maximum JVM heap size
- Use Apache JMeter to load test various heap sizes
- Use Heath Center to monitor garbage collection and heap usage
- Use GCMV to gather tuning recommendations for the JVM

Introduction

Tuning the JVM properly is a process that takes time and must be tailored to your application. You can typically get 80% of the maximum performance with 20% of the work by ensuring that you are making good choices on a few key settings. To get maximum performance from your application, you must know your application's memory allocation and runtime needs. The JVM must be tuned in two iterative steps over a testing cycle.

- Step1: Select and tune the GC policy
- Step 2: Tuning the heap size

Tuning the JVM is an iterative process. Change one parameter at a time. Measure the results of each tuning action that is taken.

In this exercise, you do the second step: Tuning the heap size. In the previous exercise, you did the first step: Select and tune the GC policy.

Requirements

Exercise 3: Installing DayTrader and **Exercise 4: Use Apache JMeter to load test DayTrader** must be successfully completed before you can do this exercise. At the end of Exercise 4, you created an Apache JMeter test plan, **`/usr/labfiles/Test_Plans/DayTrader_baselineXX_test.jmx,`** which you are going to use in this exercise.

Exercise instructions

Preface

- Most steps in this exercise are done on **hostA** unless otherwise specified.
- In this exercise, you get a chance to run several tuning-testing-monitoring cycles on TradeServer1 with different heap size settings, and a different Java SDK.
- To observe meaningful results, it is necessary to monitor the server performance for a fairly long period. In the following exercises, you are asked to monitor for 8 to 10 minutes or longer. Make sure that you allow enough time to complete all the tests.

Note

<*profiles_root*> = /opt/IBM/WebSphere/AppServer/profiles

Section 1: Start the WebSphere servers

If the Deployment Manager, node agent, and TradeServer1 are already running, stop and restart them now.

__ 1. On **hostA**, start the Deployment Manager.

__ a. Open a terminal window and enter the command:
cd <*profiles_root*>/Dmgr/bin

__ b. Enter the command:
./startManager.sh

__ 2. On **hostB**, start the node agent.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
cd <*profiles_root*>/profile1/bin

__ c. Enter the command:
./startNode.sh

__ 3. Verify the minimum and maximum heap size of TradeServer1 and then start it.

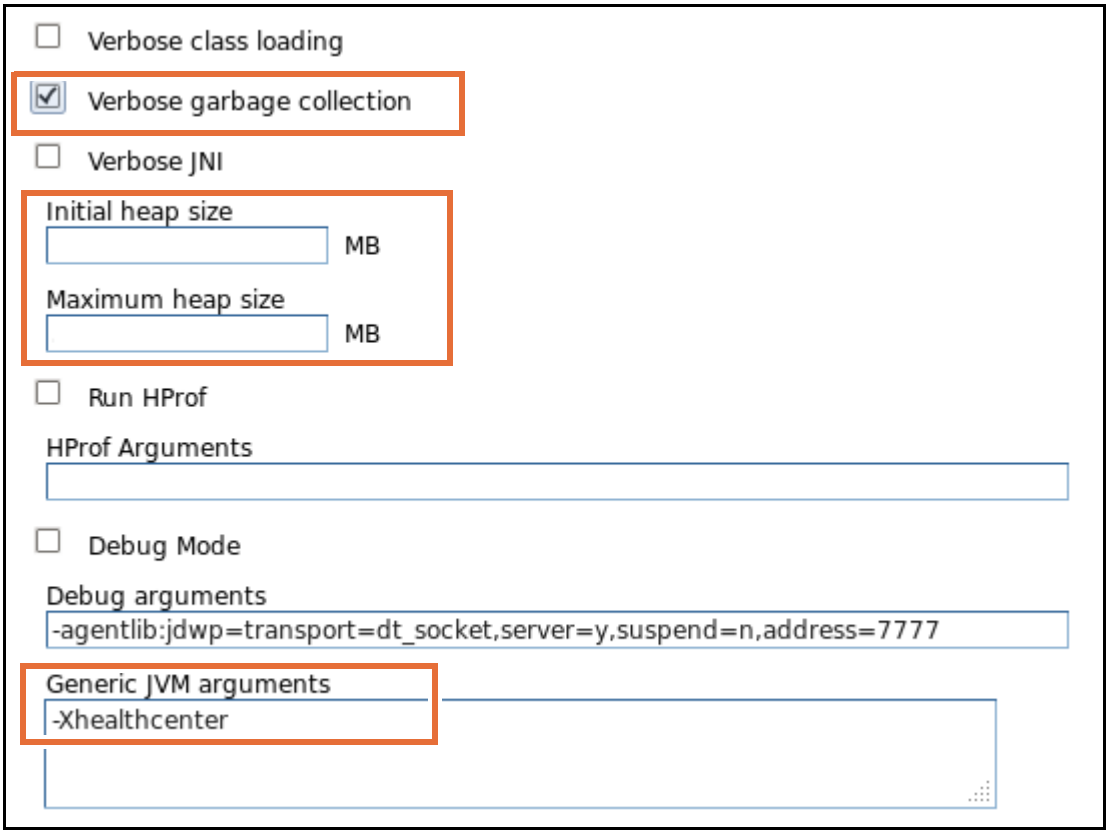
__ a. Log in to the administrative console.

__ b. Click **Server Types > WebSphere application servers > TradeServer1**.

__ c. On the Configuration tab for TradeServer1, click **Java and Process Management > Process definition**.

__ d. Under **Additional properties**, click **Java Virtual Machine**.

__ e. Click the **Configuration** tab, and verify that the minimum and maximum heap sizes are blank. If not, clear the entries for both heap sizes. Blank values cause the server to use the default minimum heap size of 50 MB and maximum heap size of 256 MB. Make sure that **Verbose garbage collection** is selected, and that **`-Xhealthcenter`** is listed as a Generic JVM argument.



__ f. Click **OK**.

__ g. Click **Save** to save the configuration.

__ h. Wait for the node to synchronize, and click **OK**.

__ 4. Stop **TradeServer1**.

__ 5. Delete all of the log files for TradeServer1.

__ a. Open an ssh terminal to hostB, and enter the command:
***`cd <profiles_root>`*`/profile1/logs/TradeServer1`**

__ b. Enter the command: **`rm *`**

__ 6. On **hostC**, restore the DayTrader database.

__ a. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ b. Enter the command: **`su - db2inst1`**

__ c. Run the following script to restore the DayTrader database.

`/usr/labfiles/db2scripts/db2restore.sh`

__ d. Run the following script to unquiesce the database.

```
/usr/labfiles/db2scripts/db2unquiesce.sh
```

__ 7. Start **TradeServer1**.

Section 2: Run the DayTrader baseline test to warm up TradeServer1

In this section, you warm up TradeServer1 and determined a new baseline for response time and throughput.

__ 1. If it is not already started, start Apache JMeter.

__ a. On **hostA**, open a terminal window and enter the command:
`cd /opt/apache-jmeter/bin`

__ b. Enter the command:
`./jmeter.sh &`

__ 2. Open the **`DayTrader_baselineXX_test.jmx`** test plan.

__ a. Click **File > Open**, and navigate to
`/usr/labfiles/Test_Plans/DayTrader_baselineXX_test.jmx`

__ b. In the JMeter tree, click **Thread Group** and change the Loop Count to **100**.

__ 3. Prepare to load test the DayTrader.

__ a. Start a web browser, and go to **`http://hostb:9081/daytrader`**

__ b. Click the **Configuration** tab, and click the **Reset DayTrader** link. Keep this browser open as you are going to reset the DayTrader for each run of the load test.

__ c. Verify that the reset completes successfully.

DayTrader Scenario Runtime Statistics

Trade Reset completed successfully — Modify runtime configuration

Benchmark runtime configuration summary	Value
Run-Time Mode	Full EJB3
Order-Processing Mode	Synchronous
Scenario Workload Mix	Standard
Web Interface	JSP
Active Traders / Trade User population	15000 / 15000
Active Stocks / Trade Stock population	10000 / 10000
Benchmark scenario verification	

__ 4. Run the JMeter test plan.

__ a. In the JMeter tree, click **Summary Report**.

__ b. Click **Run > Start** (or click the green arrow) to start the test.

__ c. Run the test three more times and record the performance results (average response time and throughput). Remember to reset DayTrader and clear the JMeter Summary Report before each test run.

Table 10: Verify baseline test.

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

__ 5. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

Section 3: Monitor garbage collection and heap usage for TradeServer1

In this section, you use two tools from the IBM Support Assistant to monitor heap usage and garbage collection for TradeServer1 as you load test the DayTrader application by using Apache JMeter.

__ 1. Start the IBM Support Assistant and run the Health Center as you did in the previous exercise.

__ a. On **hostA**, start the IBM Support Assistant from a terminal window by navigating to **`/opt/ibm/ISA5`**, and entering the command: **`./start_isa.sh`**

__ b. Minimize, but do not close the terminal window. You use this window to stop IBM Support Assistant at the end of this exercise.

__ c. Open a Firefox web browser and use the IBM Support Assistant bookmark, or type the address: **`http://localhost:10911/isa5`**

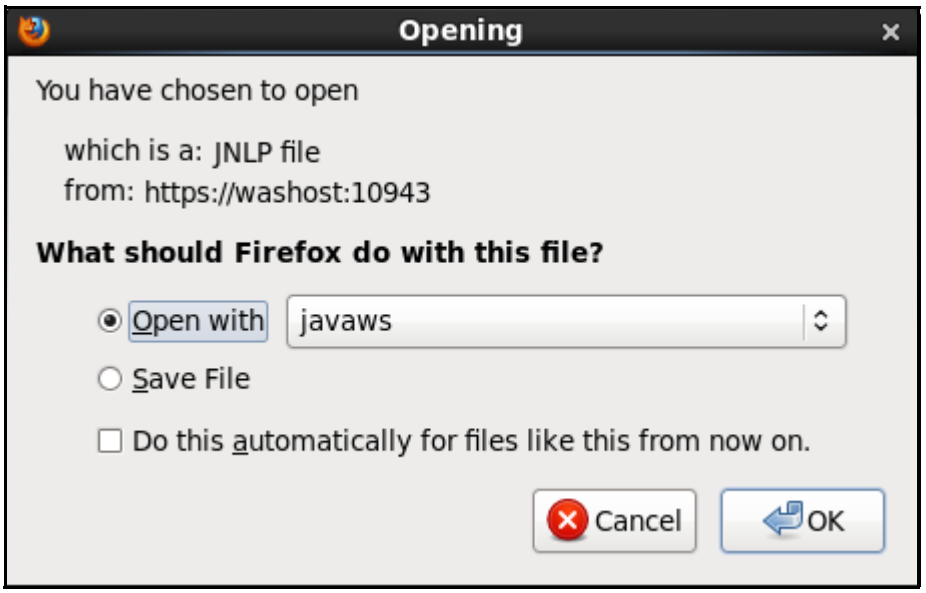
__ 2. Start Health Center.

__ a. On the IBM Support Assistant web page, click **Tools** and click **Health Center [Desktop]**.

__ b. On the right pane, click **Launch**.

__ c. In the Run Tool window, click **Submit**.

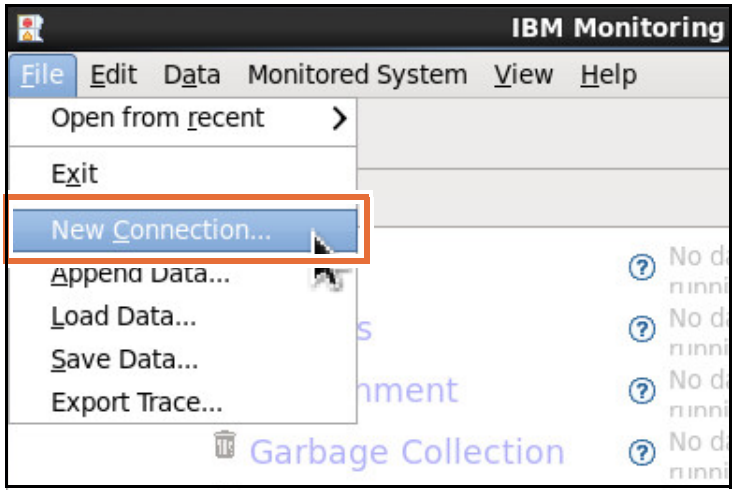
__ d. In the Opening window, select **Open with javaws**.



__ e. Click the menu for the “Open with” option, browse to `/opt/IBM/WebSphere/AppServer/java_1.7_64/jre/bin` and select **javaws**.

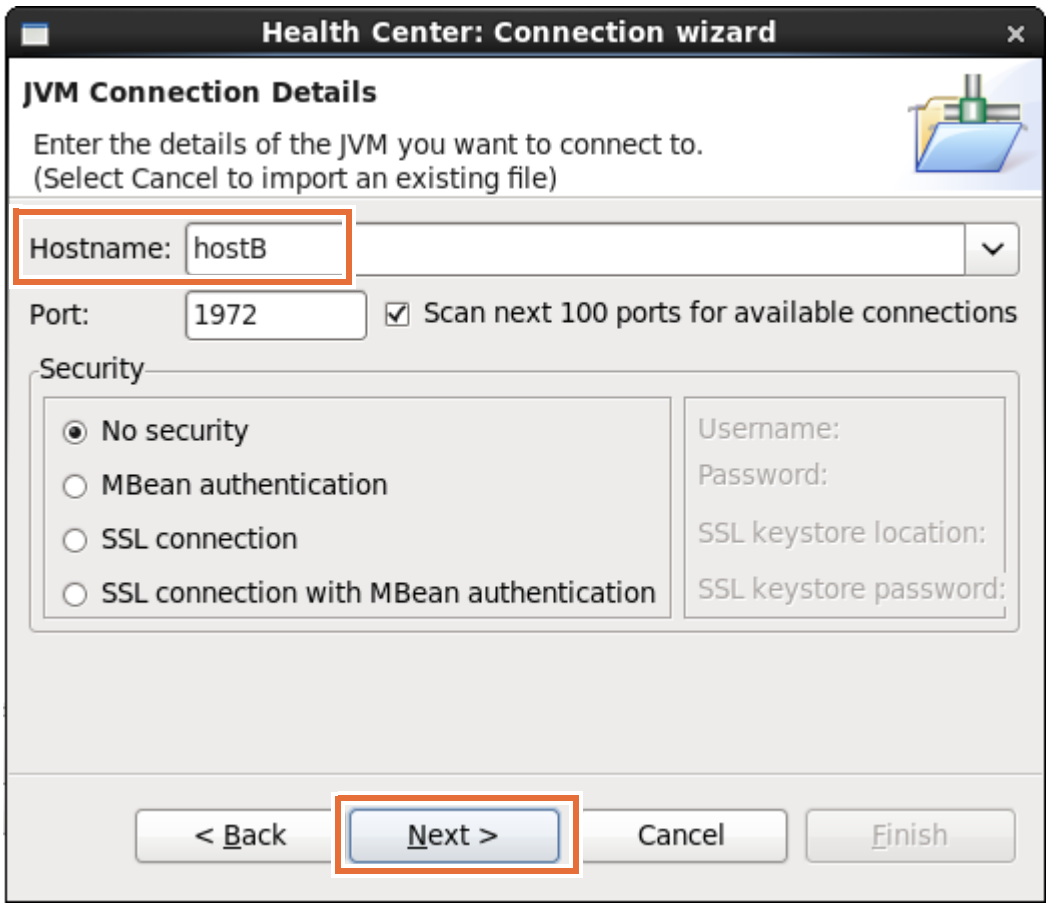
__ f. Click **OK**.

__ g. When the Health Center opens, click **File > New Connection**.



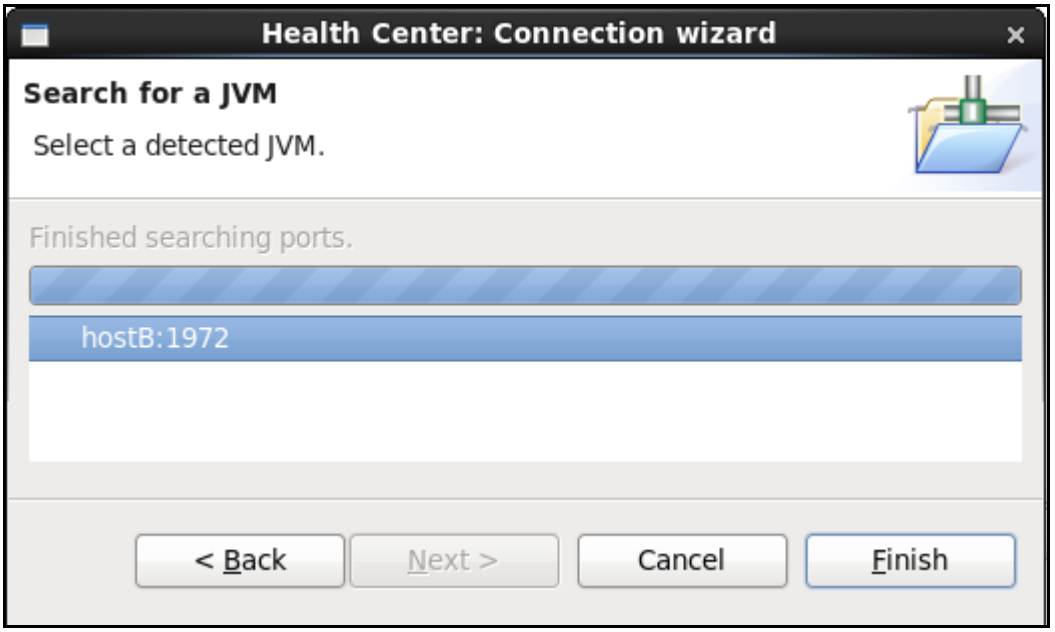
__ h. Click **Next** on the Health Center connection wizard.

__ i. Enter **hostB** for the host name and use the default port.



__ j. Click **Next**.

__ k. When the wizard detects the server that listens on port **1972**, click **Finish**.



__ l. The Health Center now pulls information from TradeServer1 and populates the GUI. This process takes a few minutes. Proceed to the next step.

__ 3. Reset DayTrader.

__ a. Open a Firefox web browser and use the DayTrader bookmark, or type the address: **`http://hostb:9081/daytrader`**

__ b. Click the **Configuration** tab, and click the **Reset DayTrader** link.

__ c. Leave this browser open because you are going to reset DayTrader several times in this lab exercise.

__ 4. If not already started, start Apache JMeter.

__ a. On **hostA**, open a terminal window and enter the command:
`cd /opt/apache-jmeter/bin`

__ b. Enter the command:
`./jmeter.sh &`

__ 5. Run the JMeter baseline test plan.

__ a. In Apache JMeter, click **File > Open**, and navigate to
`/usr/labfiles/Test_Plans/DayTrader_baselineXX_test.jmx`

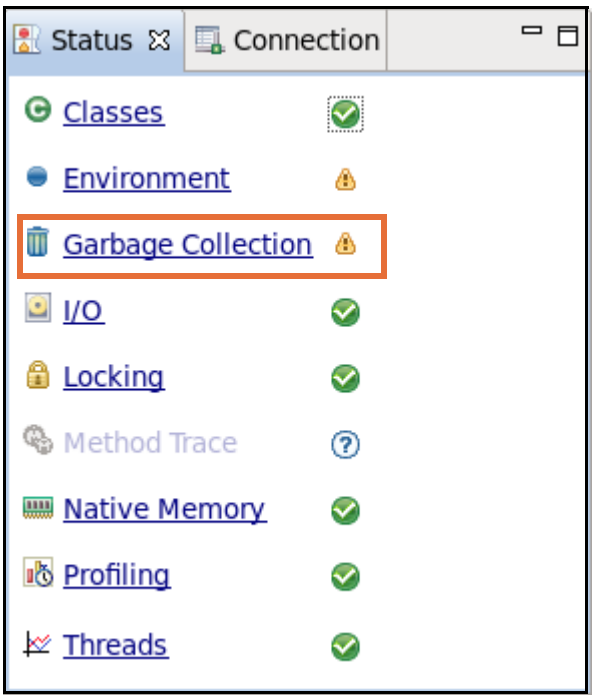
__ b. Click **Open**.

__ c. In the JMeter tree, click **Thread Group** and change the Loop Count to **Forever**.

__ d. In JMeter, click **Run > Start** (or click the green arrow) to start the test.

__ 6. Return to the Health Center, and monitor the Garbage Collection for several minutes.

__ a. In the **Status** pane, click **Garbage Collection**.

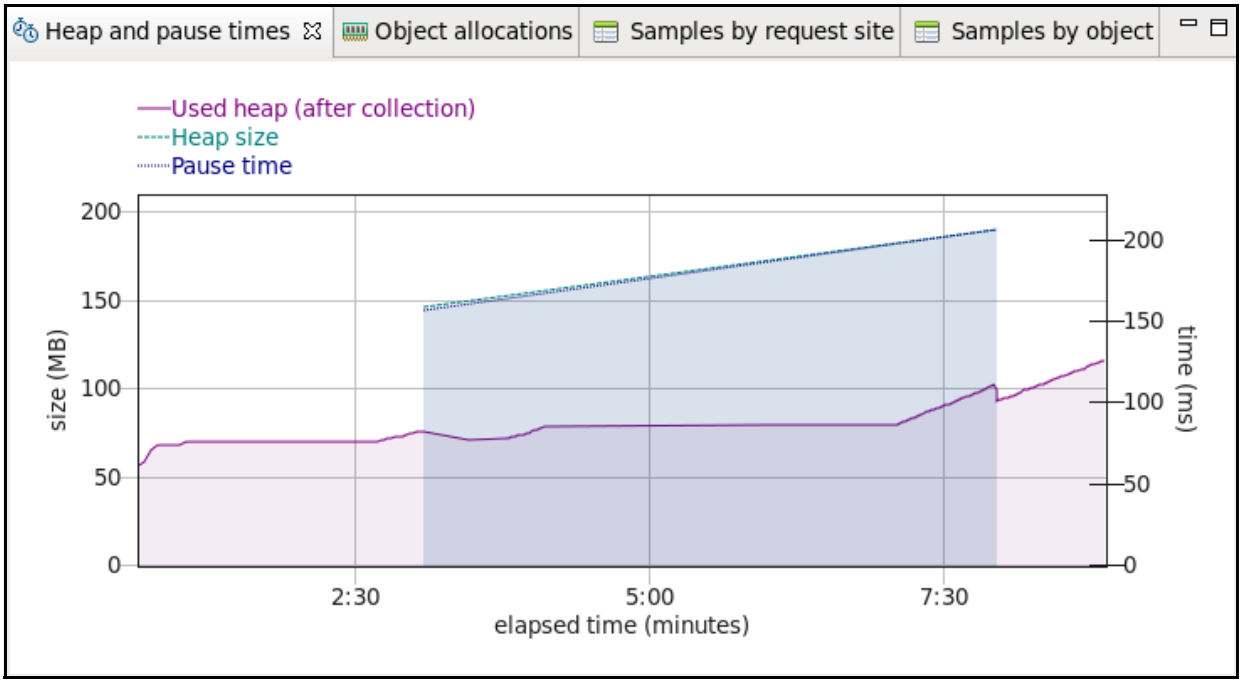


Troubleshooting

No Garbage Collection data in the Health Center?

1. Stop JMeter.
2. Reset DayTrader.
3. In the Health Center window, click **File > New Connection**.
4. Click **Next** twice.
5. Click **Yes** to close the current connect, and start a new connection on hostB port 1972 again.
6. Run the JMeter test run.

__ b. Look at the graph of the **Heap and pause times**. Pay attention to the **Heap size** and the **Used heap (after collection)**.



__ 7. After several (8 - 10) minutes of monitoring, you might see that the **Used heap (after collection)** is approaching the maximum heap size that is expanded to nearly 256 MB. Check the **Analysis and Recommendations** pane. You might see something like:

```
Compaction occurred due to extremely low heap memory. Compaction
affects performance. Consider increasing the size of the heap.
```

Or you might see some other messages about the heap size. Record your messages here.

_____

_____

_____

_____

_____

__ 8. Stop the Health Center.

__ a. In the Health Center window, click **File > Exit**.

__ 9. Stop the JMeter test.

__ a. Click **Run > Stop**.

__ 10. Stop **TradeServer1**.

__ 11. Transfer the **native_sdterr.log** file from **hostB** to **hostA**.

__ a. Use FTP to transfer the file **<*profiles_root*>/profile1/logs/TradeServer1/native_stderr.log** from **hostB** to **/usr/labfiles/Case1** on **hostA.**

__ b. In the directory **/usr/labfiles/Case1,** rename **native_stderr.log** to **native_stderr.log_256**.

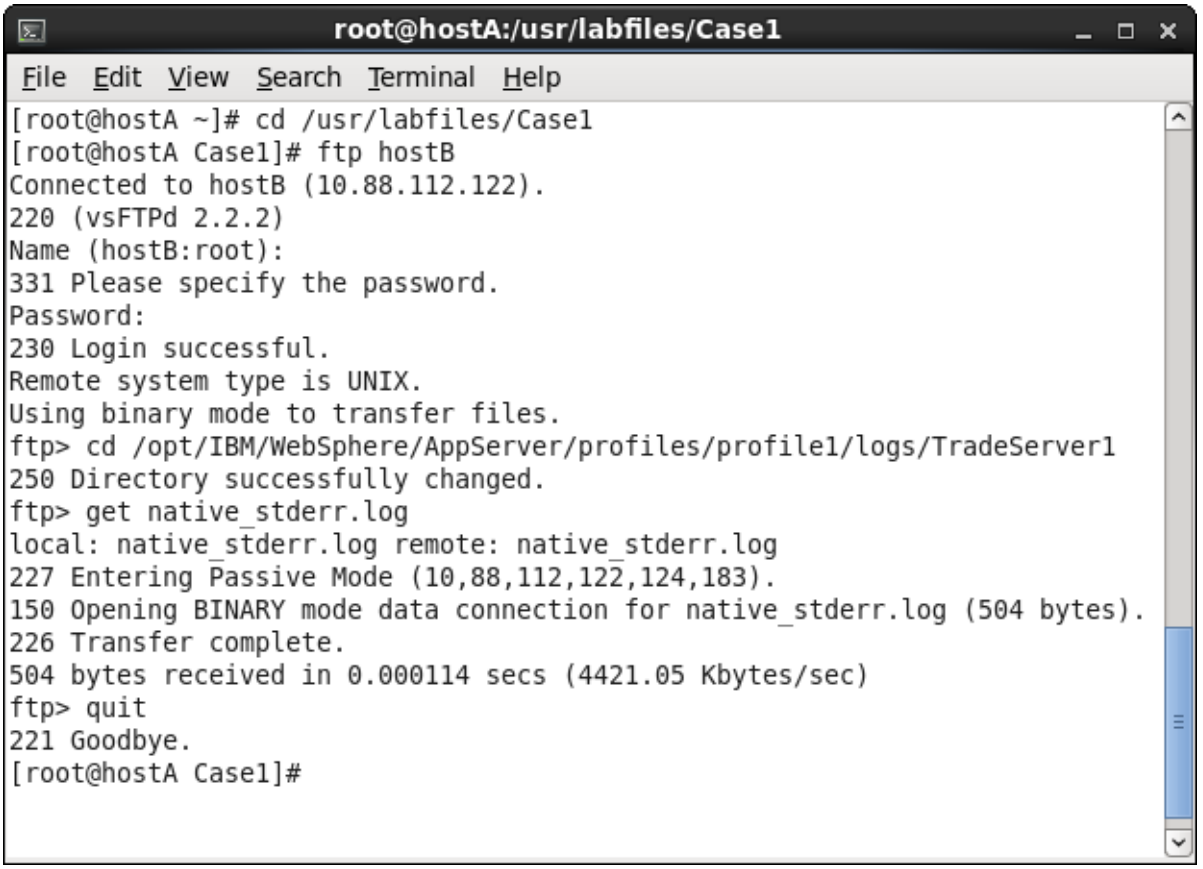
Information

Using FTP

If you are not familiar with using FTP, follow these steps.

Open a terminal window on **hostA**, and type the following commands:

1. `cd /usr/labfiles/Case1`
2. `ftp hostB`
3. `When prompted for name, enter: root`
4. `When prompted for password, enter: web1sphere`
5. `cd /opt/IBM/WebSphere/AppServer/profiles/profile1/logs/TradeServer1`
6. `get native_stderr.log`
7. `quit`



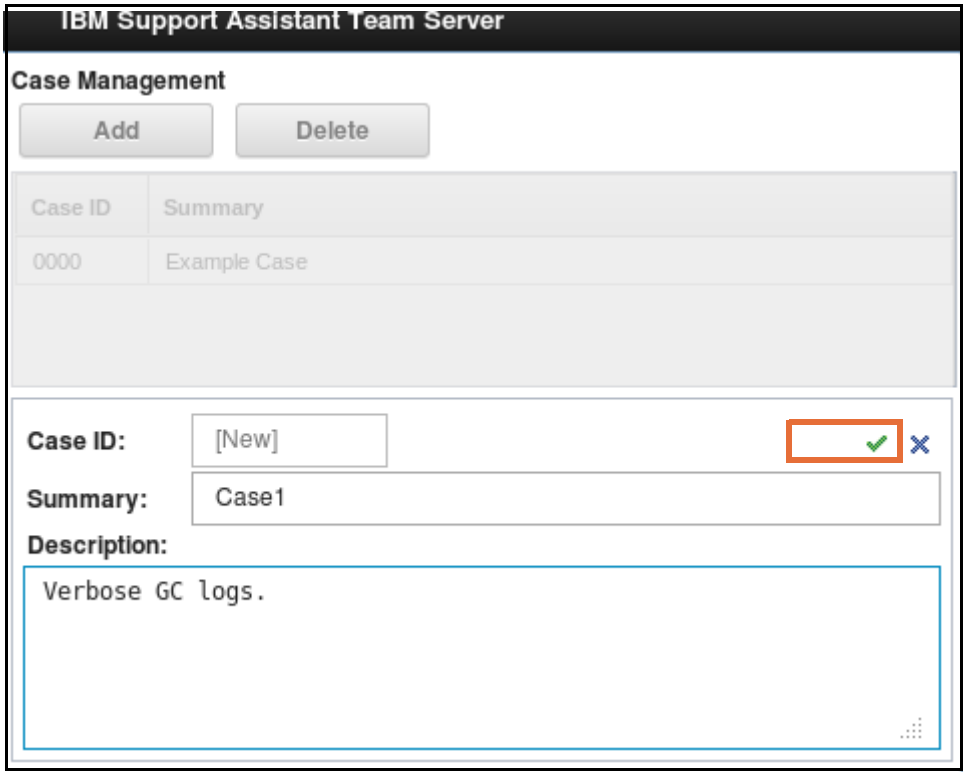
__ 8. Create a case in IBM Support Assistant to hold the native_sdterr.log files. Recall that when verbose garbage collection is enabled for a server, the verbose data is logged to native_sdterr.log of the server.

__ a. Click **Cases** to start the Case Management tool.

__ b. Click **Add**.

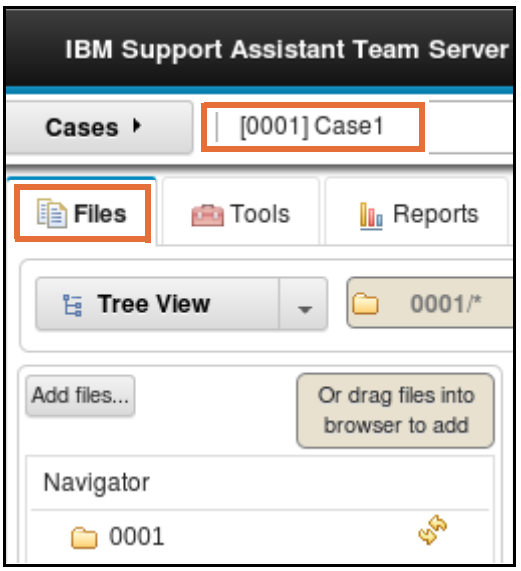
__ c. Type **Case1** as the Summary and provide a description.

__ d. Click the check mark to save the case.



__ e. Click anywhere outside the Case Management tool to close it.

__ f. Make sure that **Case1** shows in the case window, and click the **Files** tab.



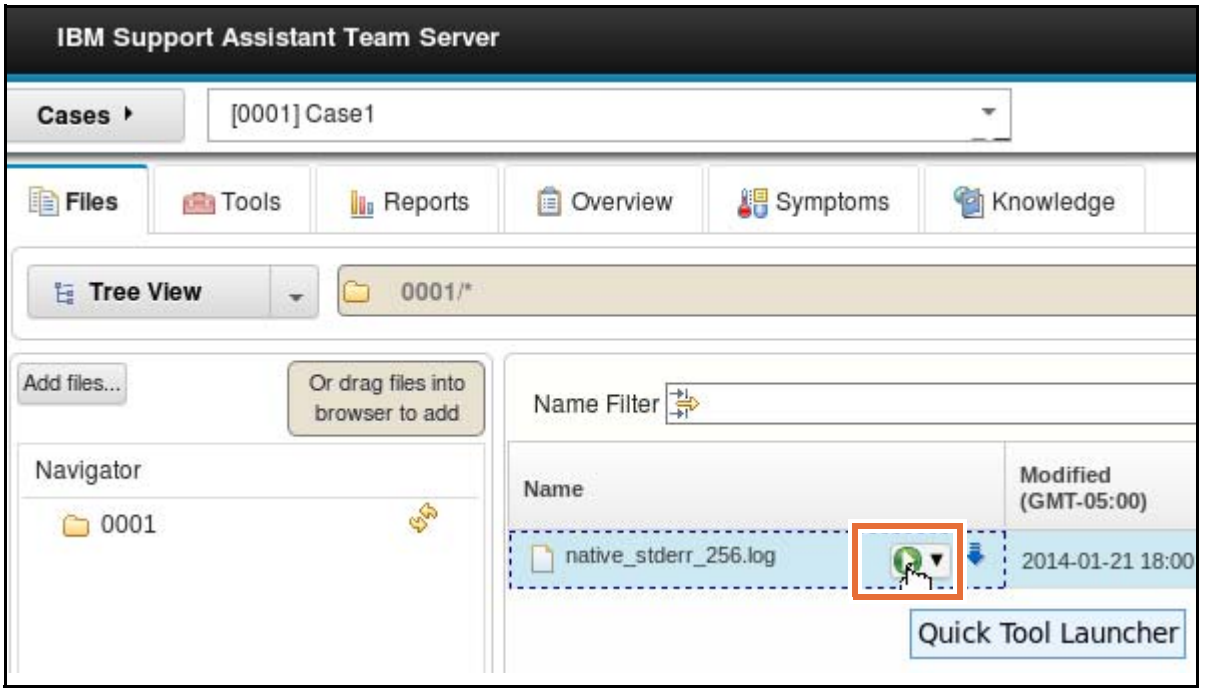
__ g. Click **Add files** and browse for **`/usr/labfiles/Case1`**.

__ h. Select **native_stderr.log_256**.

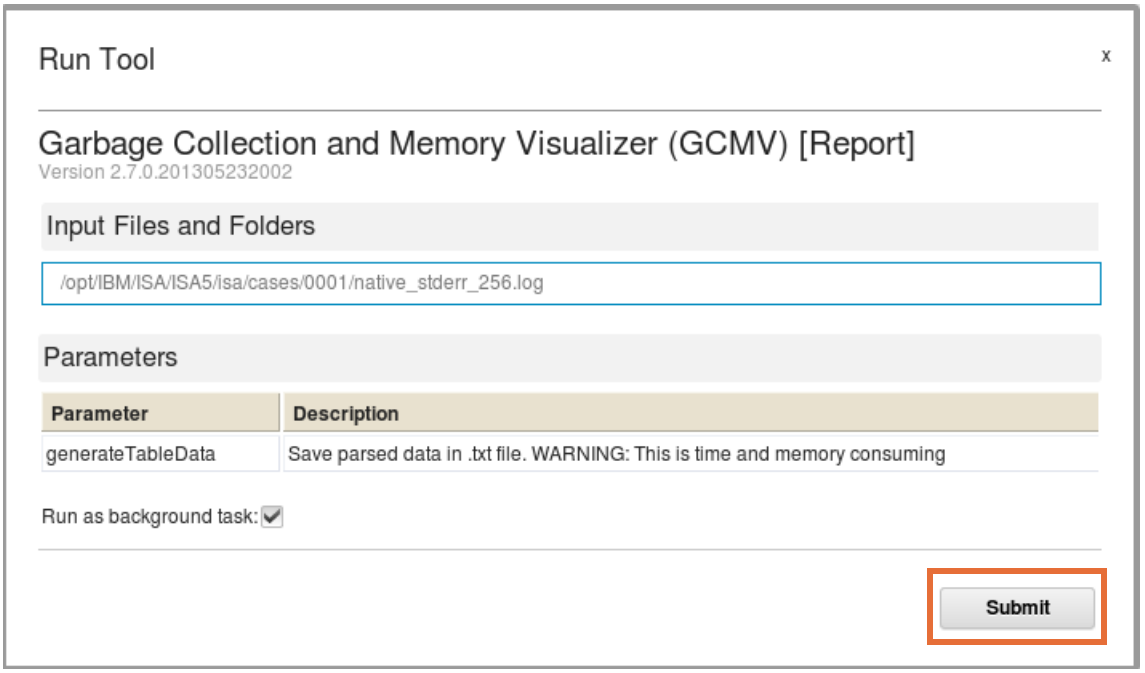
__ i. Click **Open**.

__ 9. View the verbose GC data for TradeServer1 by using the GCMV tool.

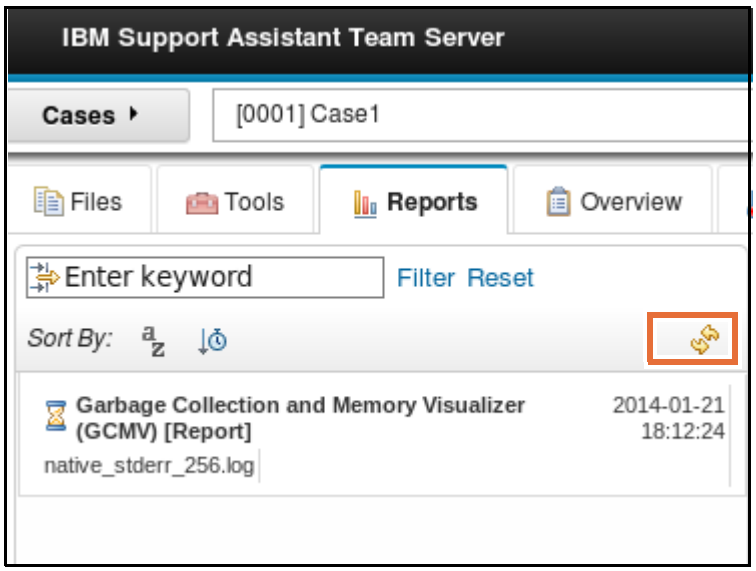
__ a. Start the GCMV Report tool from ISA 5 by clicking the **Quick Tool Launcher**. Click **GCMV Report** from the menu**.**



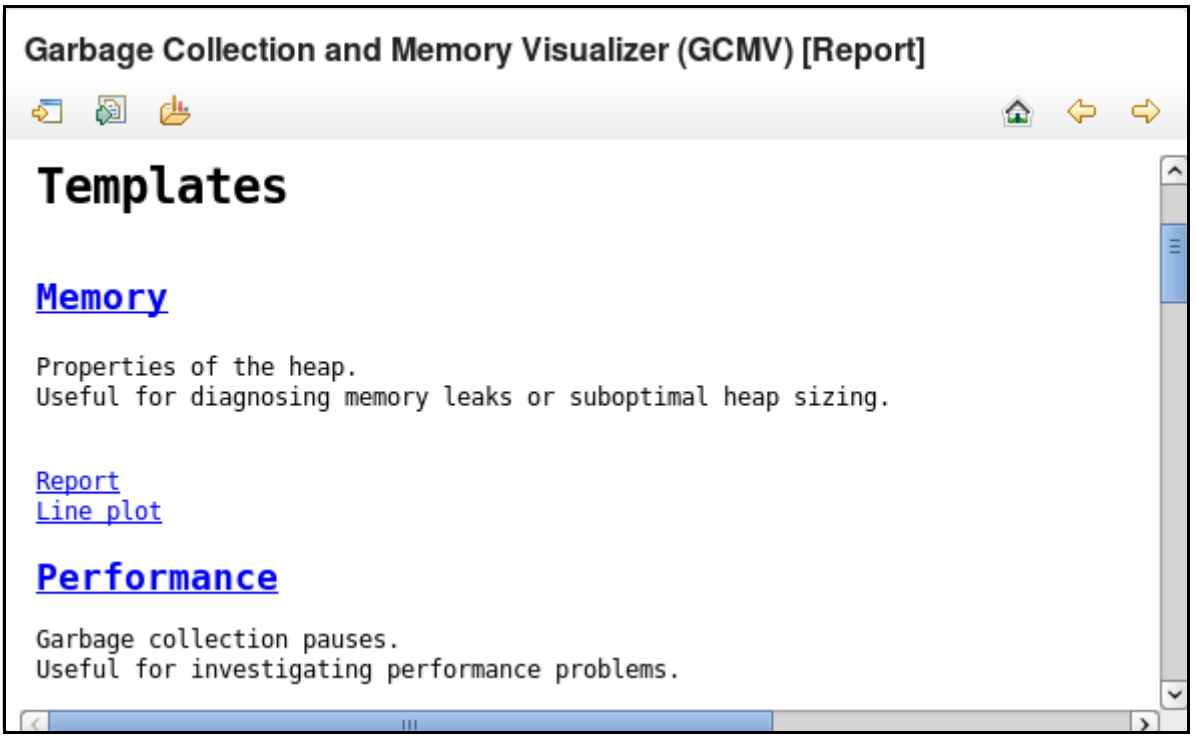
__ b. Click **Submit** on the Run Tool pane.



__ c. Click the **Reports** tab, and click the refresh icon.



__ d. Wait a few seconds and click the refresh icon until the HourGlass icon changes to a green check mark. The check mark indicates that the analysis is complete. Select the report name in the Reports list to see the report in the details pane.



__ e. Under the **Memory** template, click the **Report** link. Pay particular attention to the **Tuning recommendation** section of the report.

Tuning recommendation

Compaction occurred due to extremely low heap memory. Compaction affects performance. Consider increasing the size of the heap.

Excessive time (33.99%) is being spent in GC. Consider increasing the size of the heap.

Your application is allocating many large objects, which affects performance.Consider increasing the size of the heap.

The average memory occupancy of the tenured heap is 99% which is high. You may improve application performance by increasing the heap size.

33% of nursery collects have percolated to become global collects. This is affecting performance and increasing pause times. Consider using the Balanced GC policy for applications deployed on a 64-bit platform with a heap size greater than 4GB.

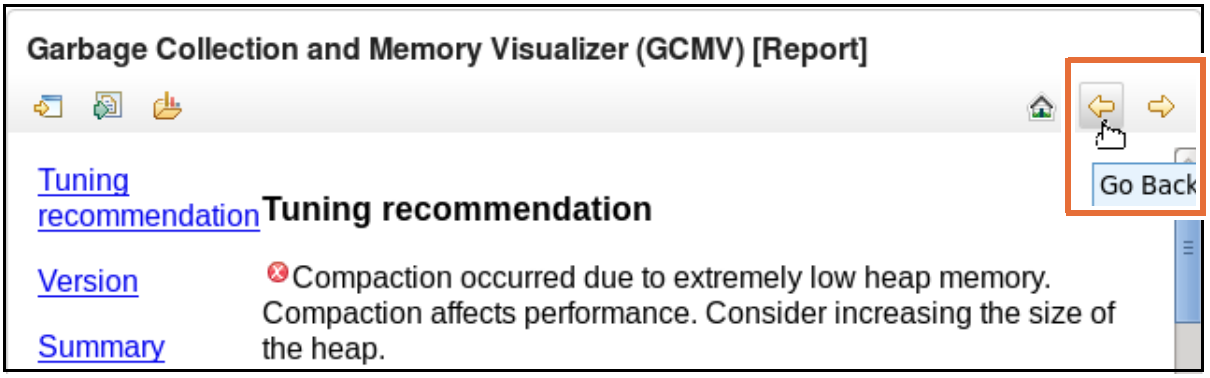
434 global garbage collects took on average 1,379% longer than the average nursery collect. If you believe this is abnormally high and unacceptable, consider using the Balanced GC policy for applications deployed on a 64-bit platform with a heap size greater than 4GB.

A high proportion of the nursery is tenured during each collection, possibly leading to longer nursery collections and more frequent collections in the tenured area. Consider increasing the nursery size, to reduce this proportion (the average is 2%).

Your application appears to be relying on class unloading during global collects. Consider using the Balanced GC policy for applications deployed on a 64-bit platform with a heap size greater than 4GB, which performs class unloading incrementally.

__ f. Your tuning recommendations might be different, but you can see many settings that you can tune. Start by increasing the maximum heap size.

__ g. Take time to explore some of the other information in the report. Click the **Go Back** icon to return to the report screen.



__ h. When you are done exploring the report, minimize the ISA browser. You are going to use ISA again in this exercise.

Section 4: Tune the maximum heap size for TradeServer1

In this section, you implement the recommendations from the Health Center and GCMV tools to increase the heap size for TradeServer1.

__ 1. Increase the maximum heap size for TradeServer1.

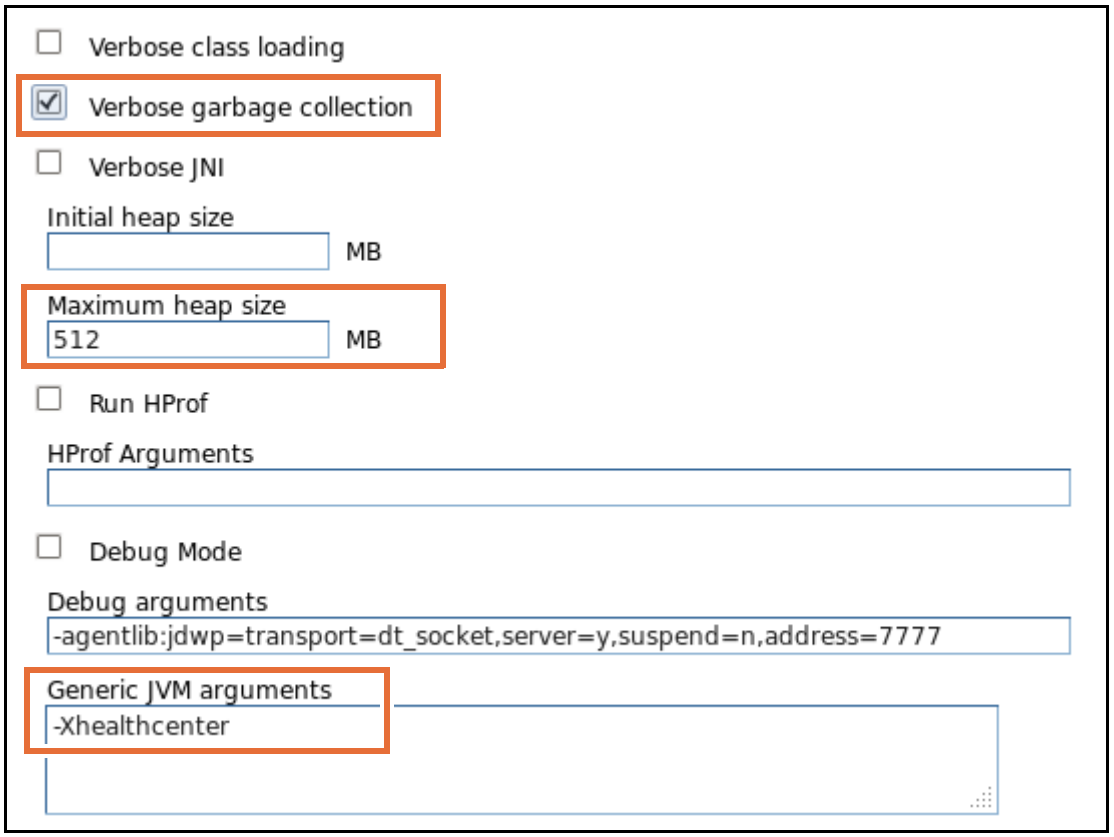
__ a. Log in to the administrative console.

__ b. Click **Servers > Server Type > WebSphere application servers > TradeServer1**.

__ c. On the configuration tab for TradeServer1, click **Java and Process Management > Process definition**.

__ d. Under **Additional properties**, click **Java Virtual Machine**.

__ e. On the configuration tab, set the maximum heap to **512 MB**, select the **Verbose garbage collection** check box, and make sure `-Xhealthcenter` is listed as a Generic JVM argument.



__ f. Click **OK**.

__ g. Click **Save** to save the configuration.

__ h. Wait for the node to synchronize, and click **OK**.

__ 2. Delete all of the log files for TradeServer1.

__ a. Open an ssh terminal to **hostB**, and enter the command:
`cd <profiles_root>/profile1/logs/TradeServer1`

__ b. Enter the command: **`rm *`**

__ 3. Restore the DayTrader database.

__ a. Run the following script to restore the DayTrader database.

```
/usr/labfiles/db2scripts/db2restore.sh
```

__ b. Run the following script to unquiesce the database.

```
/usr/labfiles/db2scripts/db2unquiesce.sh
```

__ 4. Start **TraderServer1**.

__ 5. Run the JMeter test plan.

__ a. In the JMeter tree, click **Thread Group** and change the loop count to **100**.

__ b. In the JMeter tree, click **Summary Report.**

__ c. Click **Run > Start** (or click the green arrow) to start the test.

__ d. Run the test three more times and record the performance results (average response time and throughput). Remember to reset DayTrader before each test run.

Table 11: Test run results with -Xmx=512

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

__ 6. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

__ 7. Use Health Center to monitor Garbage collection and heap usage.

__ a. Start Health Center as you did in **Section 3, Step 2 a-i**.

__ 8. Reset DayTrader.

__ 9. In the JMeter tree, click **Thread Group** and change the loop count to **Forever**, and run the test load.

__ 10. Monitor the Garbage Collection as you did **Section 3, Step 6 a-b**.

__ a. Look at the graph of the **Heap and pause times**. Pay attention to the Heap size and the Used heap (after collection).

__ b. Check the **Analysis and Recommendations** pane. Compare these messages to the ones you saw in the previous section. You might see a message similar to this one:

```
Heap usage seems to be growing over time. It increased by x% in
the last third of the log compared to the middle of the log.
```

However, the number of collections decreased by y%. This indicates that the rate at which your application is producing garbage seems to be slowing down. This may mean that your application will reach a steady-state at which the heap usage will no longer be increasing.

__ c. Summarize your messages here.

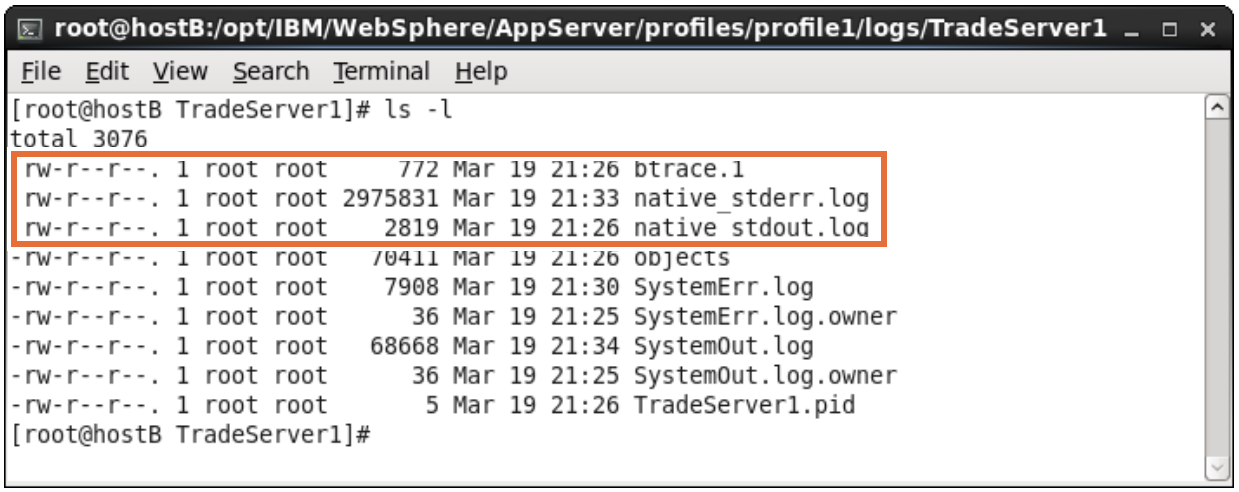
_____

_____

_____

_____

__ 11. Monitor heap usage in the Health Center for 8 to 10 minutes. Make sure that the **native_stderr.log** file for TradeServer1 is at least 1 MB in size. If not, the GCMV tool might not provide any tuning recommendations.



__ 12. Stop the Health Center.

__ a. In the Health Center window, click **File > Exit**.

__ 13. Stop the JMeter test.

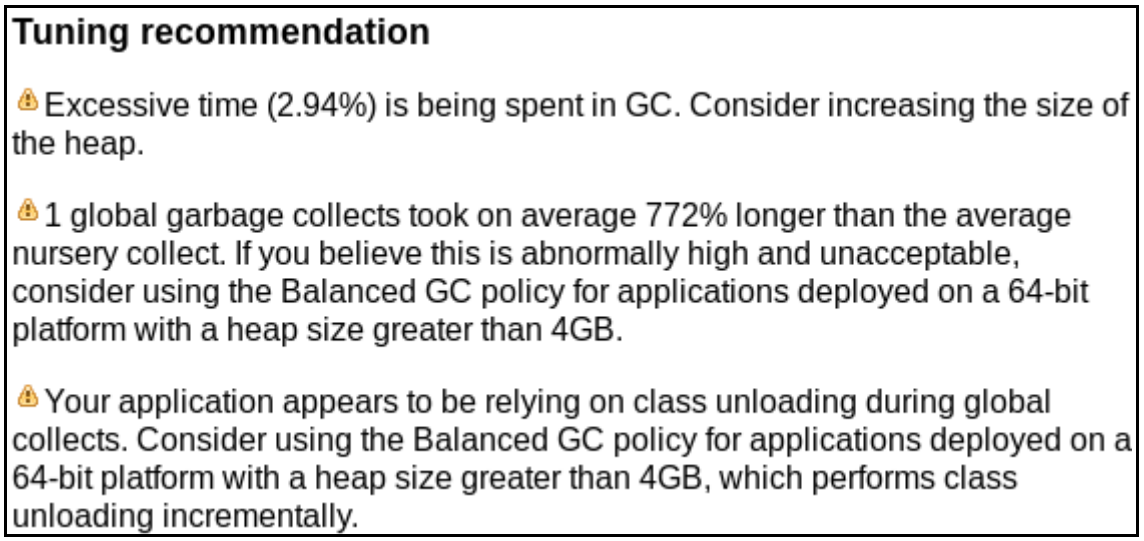
__ a. Click **Run > Stop**.

__ 14. Stop **TradeServer1**.

__ 15. Rename the **native_stderr.log** of TradeServer1 **native_stderr_512.log** and FTP it from hostB to **/usr/labfiles/Case1** on hostA

__ 16. Run the GCMV Report tool against **native_stderr_512.log**.

__ 17. View the results in the Reports tab. Click the **Report** link and look for tuning recommendations.



__ 18. Restore the DayTrader database and try increasing the maximum heap size to 768 MB.

__ a. Run the DB2 scripts to restore the DayTrader database.

__ b. Set the maximum heap size for TradeServer1 to **768 MB**.

__ c. Save the configuration.

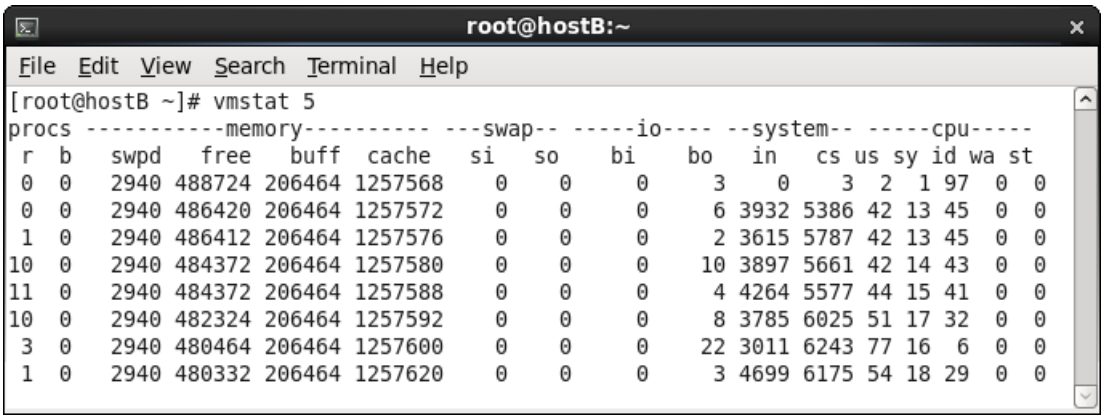
__ d. Delete all the logs on **hostB** for TradeServer1 in **`<profiles_root>/profile1/logs/TraderServer1`**

__ e. Start **TradeServer1**.

Note

Monitor server page swapping

When you increase the Java heap size, you must be careful that there is enough physical memory on your host server to prevent swapping and paging. When you run the load test, open a terminal window on hostB, and run the command: **`vmstat 5`**. Make sure that the **`si`** (swapped in) and **`so`** (swapped out) metrics are near zero.



__ 19. Run the JMeter test plan.

__ a. In the JMeter tree, click **Thread Group** and change the loop count to **100**.

__ b. In the JMeter tree, click **Summary Report**.

__ c. Click **Run > Start** (or click the green arrow) to start the test.

__ d. The first run is a warm-up, run the test three more times and record the performance results. Remember to reset DayTrader before each test run. Record your results (average response time and throughput) in the table.

Table 12: Test run results with -Xmx=768

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

__ 20. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

__ 21. Repeat **Steps 6-10**, but rename the native_stderr.log file **native_stderr_768.log.**

__ 22. Run the GCMV Report tool against **native_stderr_768.log**.

__ 23. View the results in the Reports tab. Click the **Report** link and look for tuning recommendations. Record the recommendations here.

_____

_____

_____

_____

_____

__ 24. Compare the performance of the TradeServer1 for the heap sizes you tested. Did the largest maximum heap size provide the best response time and throughput?

Section 5: Test fixed heap by setting initial heap to maximum heap size

In this section, you explore the performance of a server that uses a fixed heap size.

__ 1. Restore the DayTrader database and set both initial heap and maximum heap size to 768 MB.

__ a. Run the DB2 scripts to restore the DayTrader database.

__ b. Set both the **initial heap size** and **maximum heap size** for TradeServer1 to **768 MB**.

__ c. Save the configuration.

__ d. Delete all the logs on **hostB** for TradeServer1 in **`<profiles_root>/profile1/logs/TraderServer1`**

__ e. Start **TradeServer1**.

__ 2. Run the test plan.

__ a. In the JMeter tree, click **Thread Group** and change the Loop Count to **100**.

__ b. In JMeter, click **Run >Start** (or click the green arrow) to start the test.

__ c. The first run is a warm-up, run the test three more times and record the performance results. Remember to reset DayTrader before each test run. Record your results (average response time and throughput) in the table.

Table 13: Test run results with -Xmx=-Xms

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

__ 3. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

__ 4. Repeat **Steps 6-12** from **Section 4**, but rename the `native_stderr.log` file `native_stderr_fixed.log`. Let this test run a little longer, possibly 15 minutes, you might see an Analysis and Recommendations message similar to the following statement.

Heap usage seems to be growing over time. It increased by x% in the last third of the log compared to the middle of the log. However, the number of collections that are decreased by y%. This indicates that the rate at which your application is producing garbage seems to be slowing down. This may mean that your application will reach a steady-state at which the heap usage will no longer be increasing.

__ 5. Looking back over the data that you collected in the previous tests, enter your results here.

- Shortest average response time (ms)_____-Xmx_____-Xms_____
- Highest average throughput (req/sec)_____-Xmx_____-Xms_____

Section 6: Test TradeServer1 performance with IBM Java 7

In WebSphere Application Server V8.5 and V8.5.5, IBM Java 6 is the default Java run time, but you have the option of installing and switching to IBM Java 7. Java 7 is installed on your lab images, so in this section, you switch to Java 7 for TradeServer1 and test performance. Performance testing with DayTrader3 in EJB3 Mode, shows that Java 7 provides a significant throughput improvement over Java 6, but has a longer startup time and larger memory footprint.

__ 1. Run the DB2 scripts to restore the DayTrader database.

__ a. Run the following script to restore the DayTrader database.

```
/usr/labfiles/db2scripts/db2restore.sh
```

__ b. Run the following script to unquiesce the database.

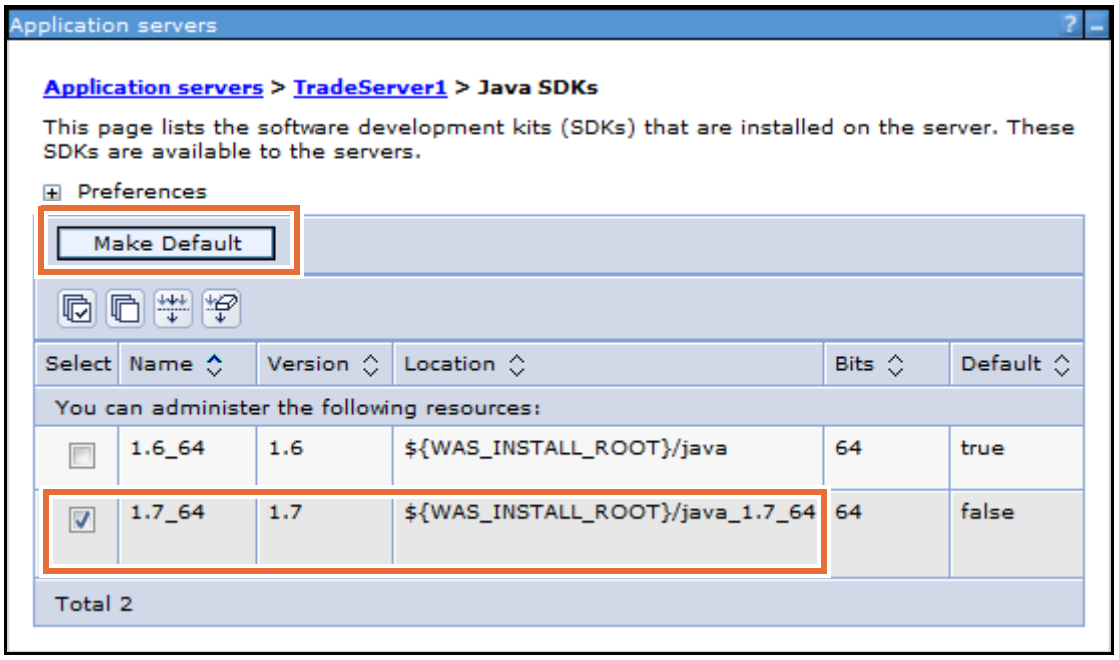
```
/usr/labfiles/db2scripts/db2unquiesce.sh
```

__ 2. Change the JDK for TradeServer1 to IBM Java 7

__ a. From the administrative console, click **Servers > Server Type > WebSphere application server > TradeServer1**.

__ b. Under **Server Infrastructure**, click **Java SDKs**.

__ c. Select Java SDK version 1.7.



__ d. Click **Make Default**.

__ e. Click **Save** to save the configuration changes.

__ f. Wait for the node to synchronize, and click **OK**.

__ g. Change the heap size back to whatever values gave the best throughput performance.

__ h. Delete all the logs on **hostB** for TradeServer1 in **`<profiles_root>/profile1/logs/TraderServer1`**

__ i. Start the **TradeServer1**.

__ 3. Run the test plan.

__ a. In the JMeter tree, click **Thread Group** and change the Loop Count to **100**.

__ b. In JMeter, click **Run > Start** (or click the green arrow) to start the test.

__ c. The first run is a warm-up, run the test three more times and record the performance results. Remember to reset DayTrader before each test run. Record your results (average response time and throughput) in the table.

Table 14: Test run results with Java 7 and -Xmx=_____-Xms=_____

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

__ 4. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

__ d. Repeat **Steps 6-12** from **Section 4**, but rename the `native_stderr.log` file **`native_stderr_Java7.log`**.

__ 5. Run the GCMV Report tool against **`native_stderr_Java7.log`**.

__ 6. View the results in the Reports tab. Click the **Report** link and look for tuning recommendations. Record the recommendations here.

_____

_____

_____

_____

_____

Important

Change the default Java SDK back to version 1.6

For the purposes of the remaining exercises, use the steps that are described in the beginning of this **Section 6** to set the default **Java SDK** back to **version 1.6**.

Section 7: Cleanup

If this exercise is the last exercise you do for several hours, you should stop the WebSphere servers.

__ 1. On **hostA**, stop the Deployment Manager.

__ a. Open a terminal window and enter the command:
`cd <profiles_root>/Dmgr/bin`

__ b. Enter the command:
`./stopManager.sh -username wasadmin -password web1sphere`

__ 2. On **hostB**, stop the node agent and TradeServer1.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
`cd <profiles_root>/profile1/bin`

__ c. Enter the command:
`./stopNode.sh -username wasadmin -password web1sphere`

__ d. Enter the command:
`./stopServer.sh TradeServer1 -username wasadmin -password web1sphere`

End of exercise

Exercise review and wrap-up

In this exercise, you configure the initial and maximum JVM heap size for the application server. You used Apache JMeter to load test different heap sizes, and you used Heath Center to monitor garbage collection and heap usage. Finally, you used the Garbage Collection and Memory Visualizer (GCMV) tool to analyze garbage collection log data and gather tuning recommendations for the JVM.

Exercise 8. Troubleshooting JVM problems

What this exercise is about

The purpose of this exercise is to show how to detect and troubleshoot common JVM problems. Problems that cause the server to hang or crash can cause poor performance.

What you should be able to do

At the end of this exercise, you should be able to:

- Use the IBM Support Assistant Health Center tool to monitor Java heap size and usage
- Detect and troubleshoot a server that is experiencing out-of-memory exceptions
- Use the IBM Support Assistant Memory Analyzer tool to analyze heap dumps
- Detect and troubleshoot a server with hung threads
- Use the IBM Support Assistant Thread Monitor Dump Analyzer tool to analyze thread dumps

Introduction

When operating a WebSphere Application Server environment, you might see problems and failures of running application servers. Typically two scenarios cause these problems. For example:

- Servers failing due to java.lang.OutOfMemory errors
- Servers slow down or stop working due to hung threads

In the first scenario, you learn how to use the Memory Analyzer for analyzing a Java heap dump that results from a java.lang.OutOfMemory error due to a faulty servlet. In the second scenario, you analyze a javacore file of a hanging application server and try to determine the cause of the hang by using the IBM Thread and Monitor Dump Analyzer.

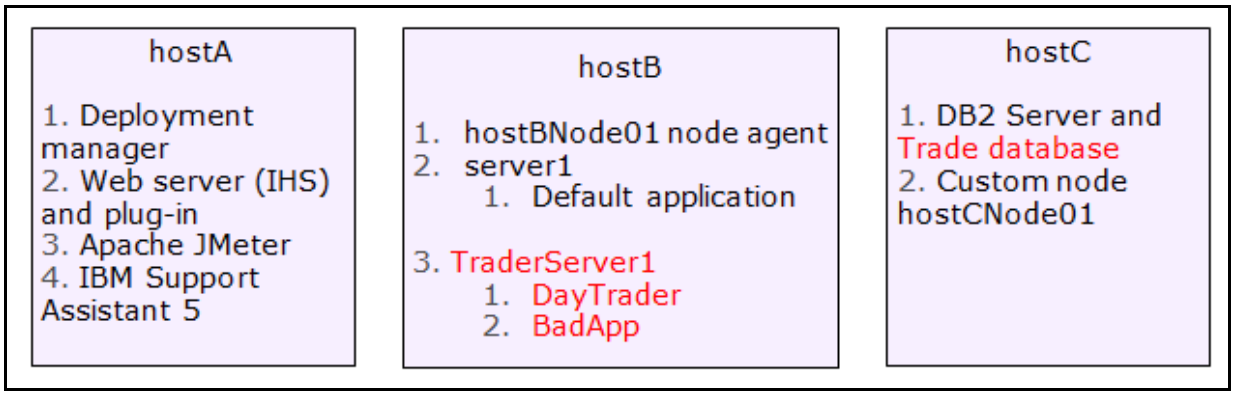
Requirements

This exercise does not depend on any previous exercise. To complete the activities in this exercise, you need the **BadApp.ear** file and the **IBM Support Assistant**. The required items are provided on the lab image that is used for this course.

Exercise instructions

Preface

- Most steps in this exercise are done on **hostA** unless otherwise specified.
- After completing this exercise, the BadServer and BadApp application are on hostB.



Note

<*profiles_root*> = /opt/IBM/WebSphere/AppServer/profiles

Section 1: Start the WebSphere servers

If the Deployment Manager and node agent are already running, stop and restart them now.

__ 1. On **hostA**, start the Deployment Manager.

__ a. Open a terminal window and enter the command:
cd <*profiles_root*>/Dmgr/bin

__ b. Enter the command:
./startManager.sh

__ 2. On **hostB**, start the node agent.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
cd <*profiles_root*>/profile1/bin

__ c. Enter the command:
./startNode.sh

__ 3. If TradeServer1 or server1 is running, stop them.

__ a. On **hostB**, enter the command:
cd <*profiles_root*>/profile1/bin

__ b. Enter the command:
`./stopServer.sh server1`

__ c. Enter the command:
`./stopServer.sh TradeServer1`

__ 4. Create an application server called BadServer.

__ a. Start a web browser and enter the web address:
`http://localhost:9060/ibm/console`

__ b. Log in to the administrative console with the user ID **wasadmin** and password **web1sphere**

__ c. Click **Servers > Server Types > WebSphere application servers**.

__ d. Click **New** to create an application server.

__ e. In **Step 1** of the wizard, select node **hostBNode01**.

__ f. For the Server name, enter: **`BadServer`**

__ g. Click **Next**.

__ h. In **Step 2**, accept the default template and click **Next**.

__ i. In **Step 3**, select **Generate Unique Ports** and **Next**.

__ j. In **Step 4**, read the Summary and click **Finish**.

__ k. Save the changes.

__ l. Wait for the node to synchronize and click **OK.**

Section 2: Install the BadApp application

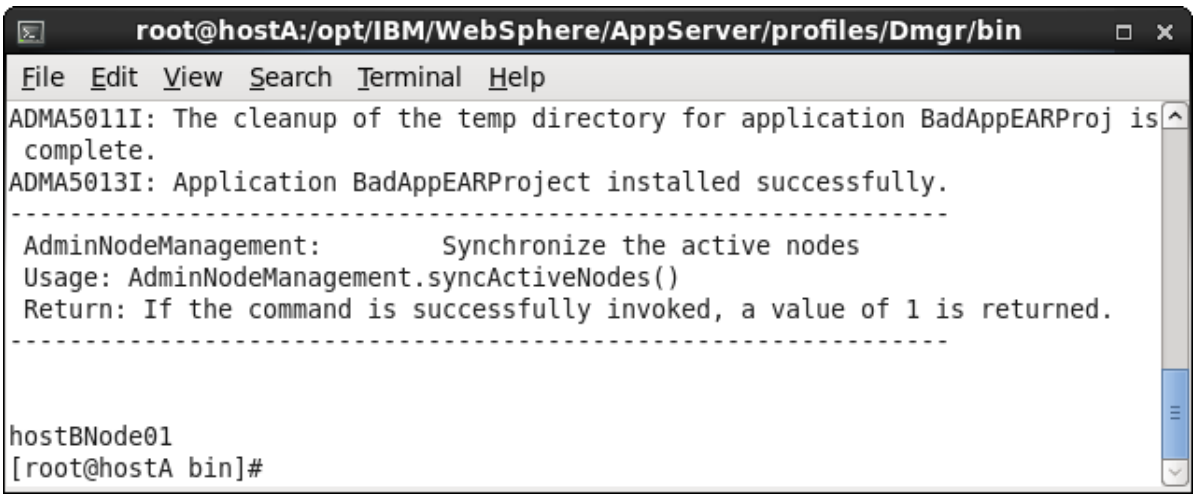
This application was specially developed to exhibit common behaviors that reduce performance such as memory exhaustion and hung threads.

__ 1. On **hostA**, open a terminal window and enter the command:
`cd <profiles_root>/Dmgr/bin`

__ 2. Run the following command:

```
./wsadmin.sh -f /usr/labfiles/Badapp/install_badapp.py -user
wasadmin -password web1sphere
```

__ 3. When the installation process completes, verify that you see the following message.



__ 4. Map BadServer's web container port 9082 to the virtual host default_host.

__ a. From the administrative console, click **Environment > Virtual hosts > default_host**.

__ b. Click **Host Aliases**.

__ c. Click **New**.

__ d. Enter Port **9082** and click **OK**.

__ e. Save the changes.

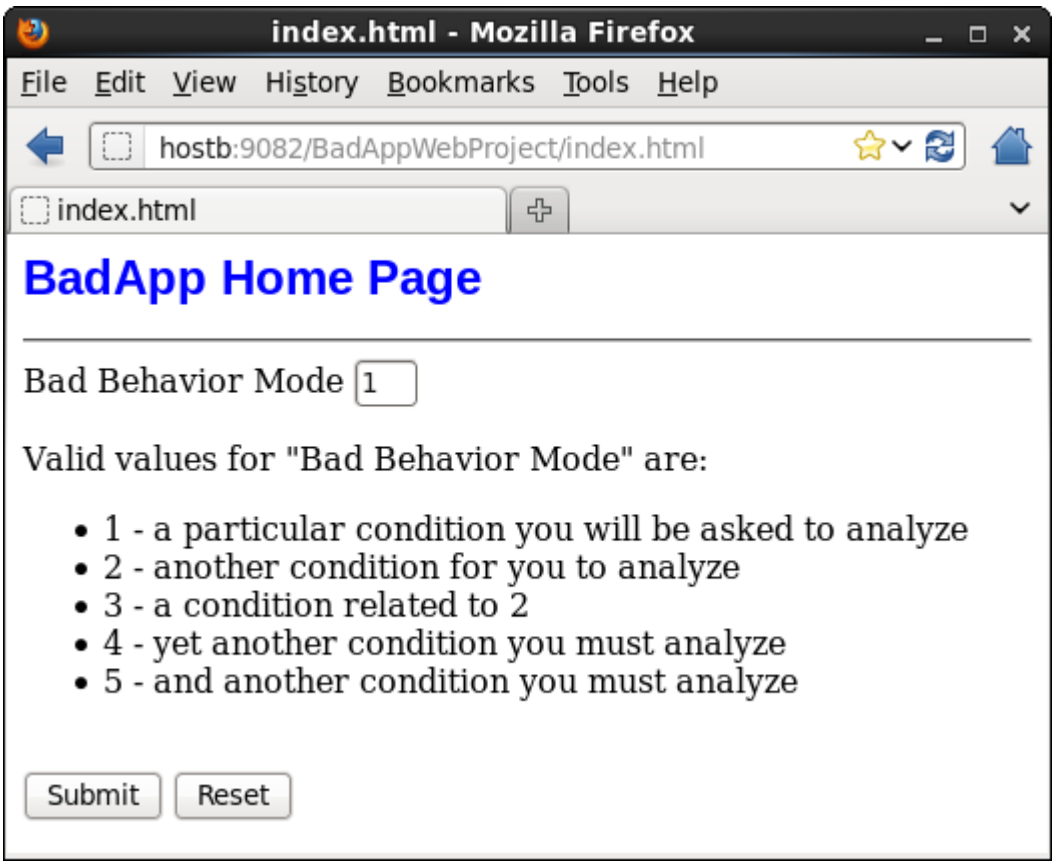
__ f. Wait for the node to synchronize and click **OK**.

__ 5. Start BadServer from the administrative console.

__ 6. Access the BadApp application in a web browser.

__ a. Start a new instance of the Firefox web browser.

__ b. Enter the web address: **`http://hostb:9082/BadAppWebProject/index.html`**



__ c. Bookmark the BadApp home page in your browser.

__ d. Close the browser.

Section 3: Trigger a memory leak with the BadApp application

You decrease maximum heap size for BadServer to 128 MB in this section to demonstrate an OutOfMemory exception.

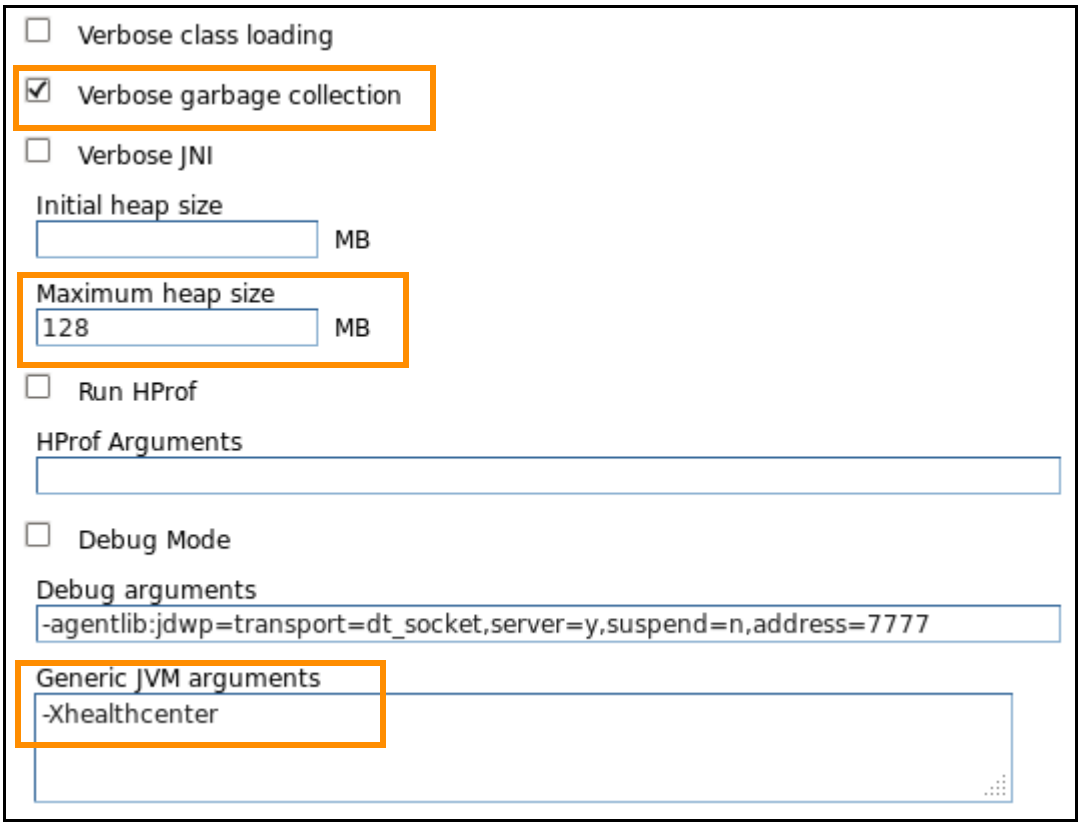
__ 1. Modify the JVM heap size for BadServer.

__ a. From the administrative console, click **Servers > Server Types > WebSphere application servers > BadServer**.

__ b. On the configuration tab for BadServer, click **Java and process management**.

__ c. Click **Process definition > Java Virtual Machine**.

__ d. On the JVM configuration tab for BadServer, select the **Verbose garbage collection** check box; set Maximum heap size to **128**, and add **`-Xhealthcenter`** to the Generic JVM arguments.



__ e. Click **OK**.

__ f. Click the link to **Save** directly to the master configuration.

__ g. Wait for the node to synchronize and click **OK**.

__ 2. Stop **BadServer**, delete its log files, and then restart **BadServer**.

__ a. Stop **BadServer** from the administrative console.

__ a. Open an ssh terminal to **hostB**, and navigate to **`<profile_root>/profile1/logs/BadServer`**

__ b. Remove all the files in the directory: **`rm *`**

__ c. Start **BadServer** from the administrative console and wait for it to start successfully.

__ d. Log out of the administrative console.

__ 3. Start IBM Support Assistant.

__ a. On **hostA**, start ISA from a terminal window by navigating to **`/opt/ibm/ISA5`**, and entering the command:
`./start_isa.sh`

__ b. Minimize, but do not close the terminal window.

__ c. Open a Firefox web browser and use the ISA bookmark, or type the address:
`http://localhost:10911/isa5`

__ d. The IBM Support Assistant home page is displayed in the browser.

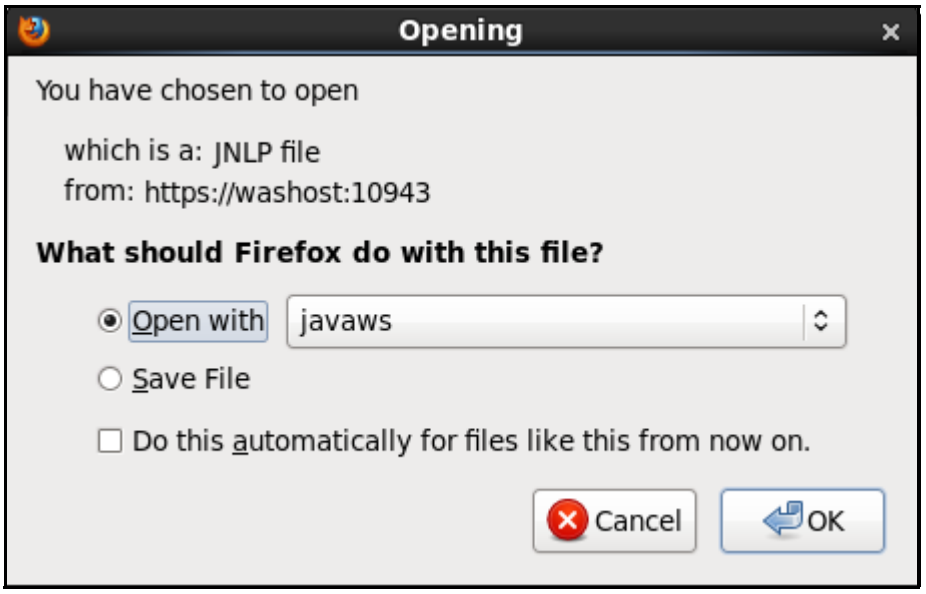
__ 4. Start the Health Center [Desktop] tool.

__ a. In the IBM Support Assistant, click the **Tools** tab and click **Health Center**.

__ b. On the right pane, click **Launch.**

__ c. In the Run Tool window, click **Submit**.

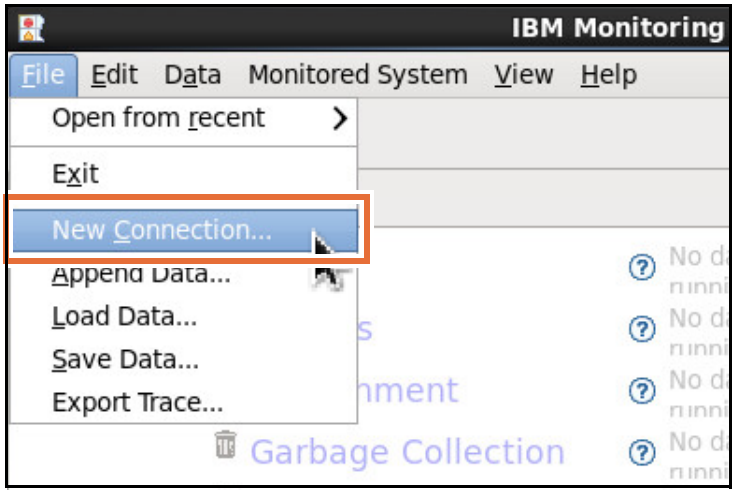
__ d. In the Opening window, select **Open with javaws**.



__ e. Click the menu for the “Open with” option, browse to `/opt/IBM/WebSphere/AppServer/java_1.7_64/jre/bin` and select **javaws**.

__ f. Click **OK**.

__ g. When the Health Center opens, click **File > New Connection**.

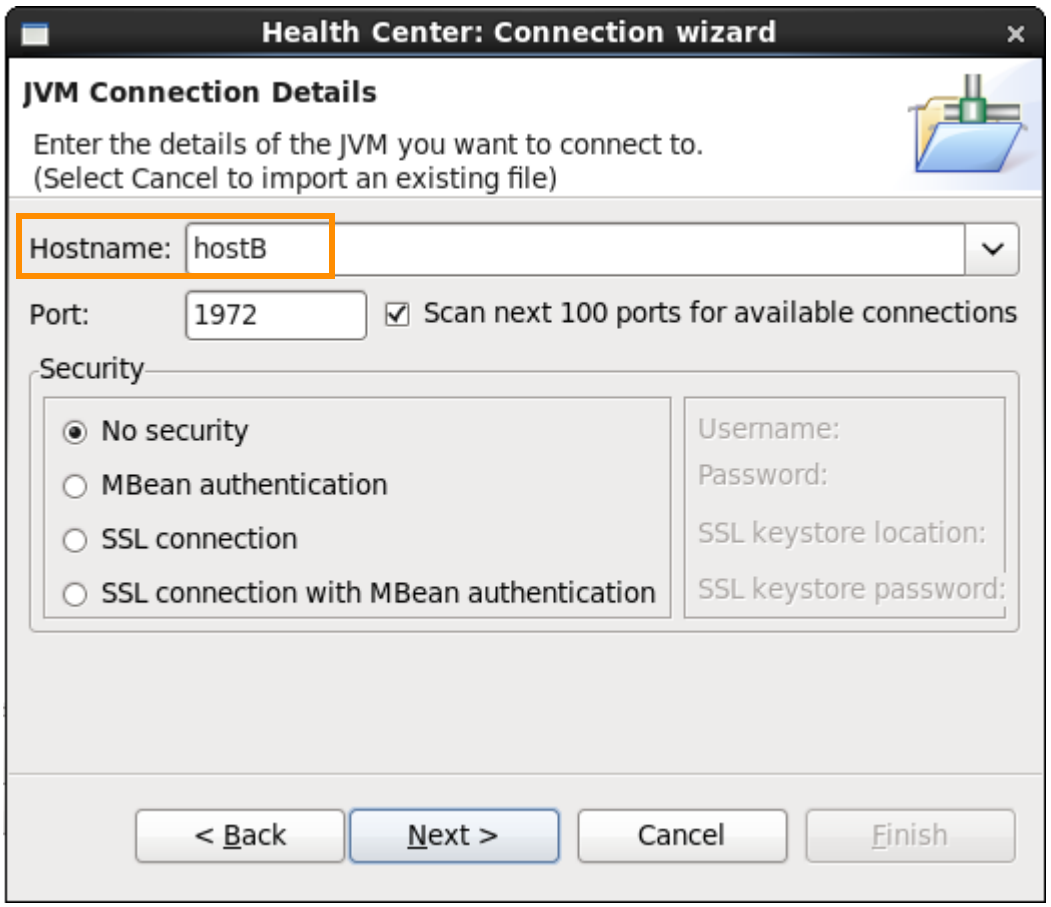


__ h. Click **Next** on the Health Center connection wizard.

Note

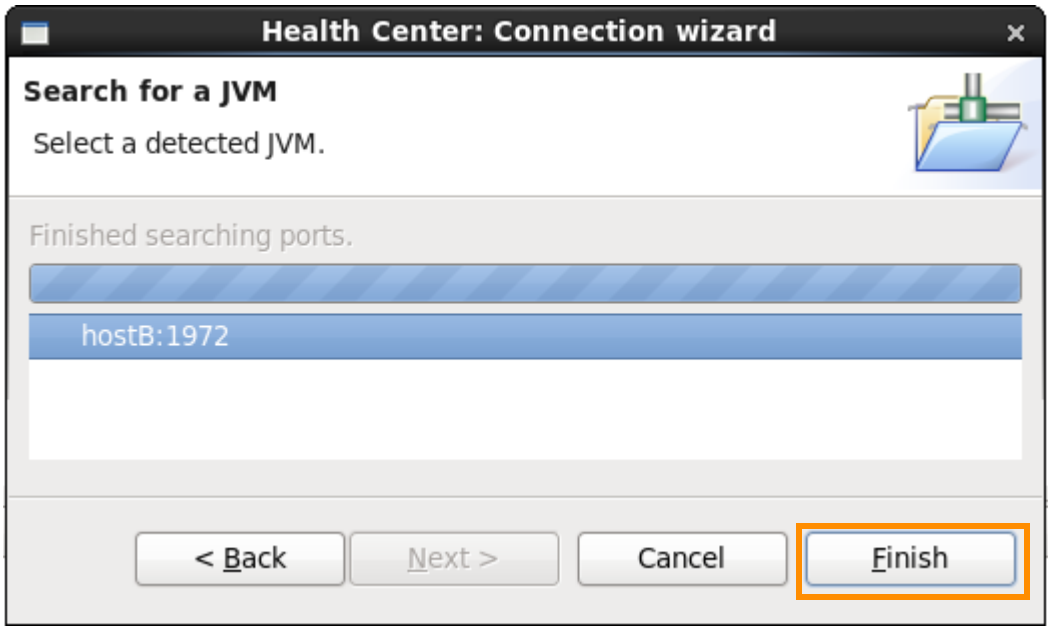
Make sure that any other applications servers such as **server1** and **TradeServer1** are not running so that BadServer is listening on port 1972. Strictly speaking, only when an application server uses the **-Xhealthcenter** argument does it listen on the default port 1972. If other servers run on the same host with the **-Xhealthcenter** argument, their port number is increment by 1 (1973, 1974, and so on).

__ i. Enter **hostB** for the host name and use the default port.



__ j. Click **Next**.

__ k. When the wizard detects the server that listens on port 1972, click **Finish**.



__ l. Wait a few minutes while the Health Center retrieves runtime data from the server. Eventually the Health Center GUI is populated with server data.

__ 5. Use the Health Center to monitor heap usage on the BadServer.

__ a. Click **Garbage Collection** on the Status pane.

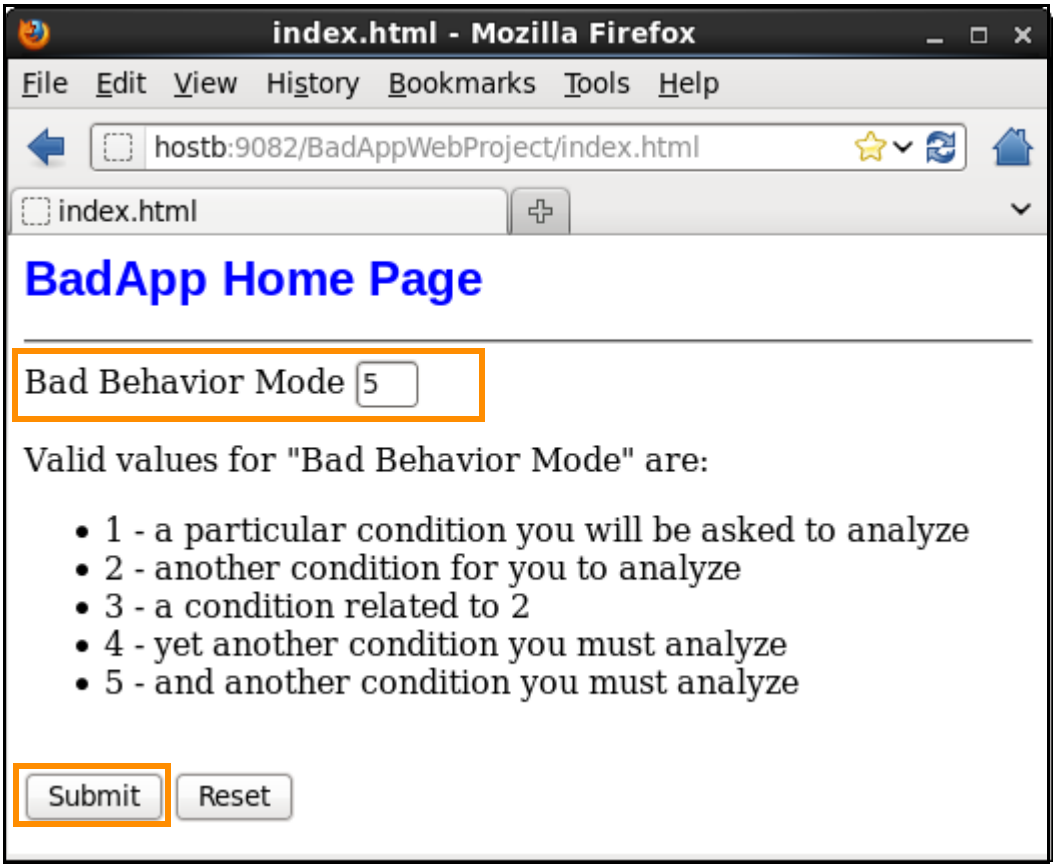
__ b. Look at the graph of the **Heap and pause times**. Pay attention to the **Heap size** and the **Used heap (after collection)**.

__ c. After the BadApp application is run, you are going to return to the Health Center to monitor JVM activity. Proceed to the next step.

__ 6. Run the BadApp enterprise application.

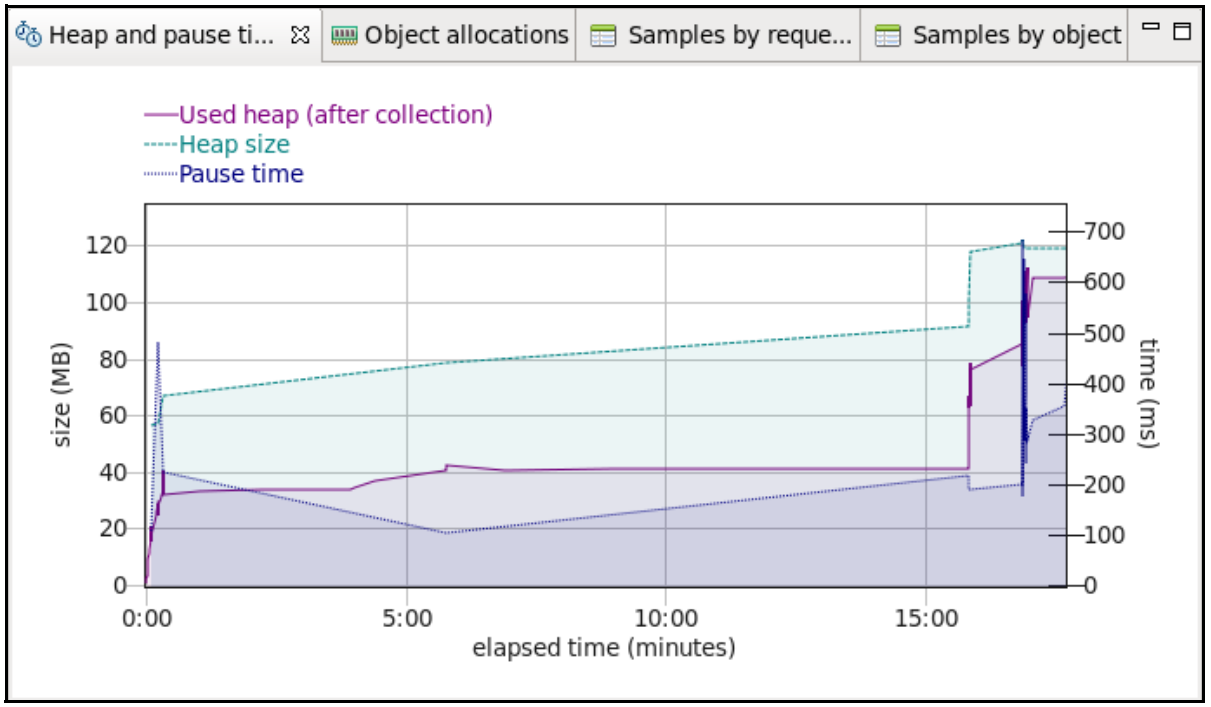
__ a. Start a new web browser and use the bookmark that you created or enter the web address: **`http://hostb:9082/BadAppWebProject/index.html`**

__ b. Enter **5** for the **Bad Behavior Mode**, and click **Submit**.



__ 7. The browser appears to hang for about 2 minutes until the application displays an error page. During this time, a number of actions exist that you can do to monitor the behavior of the application.

__ a. Use the Health Center to view the heap usage.

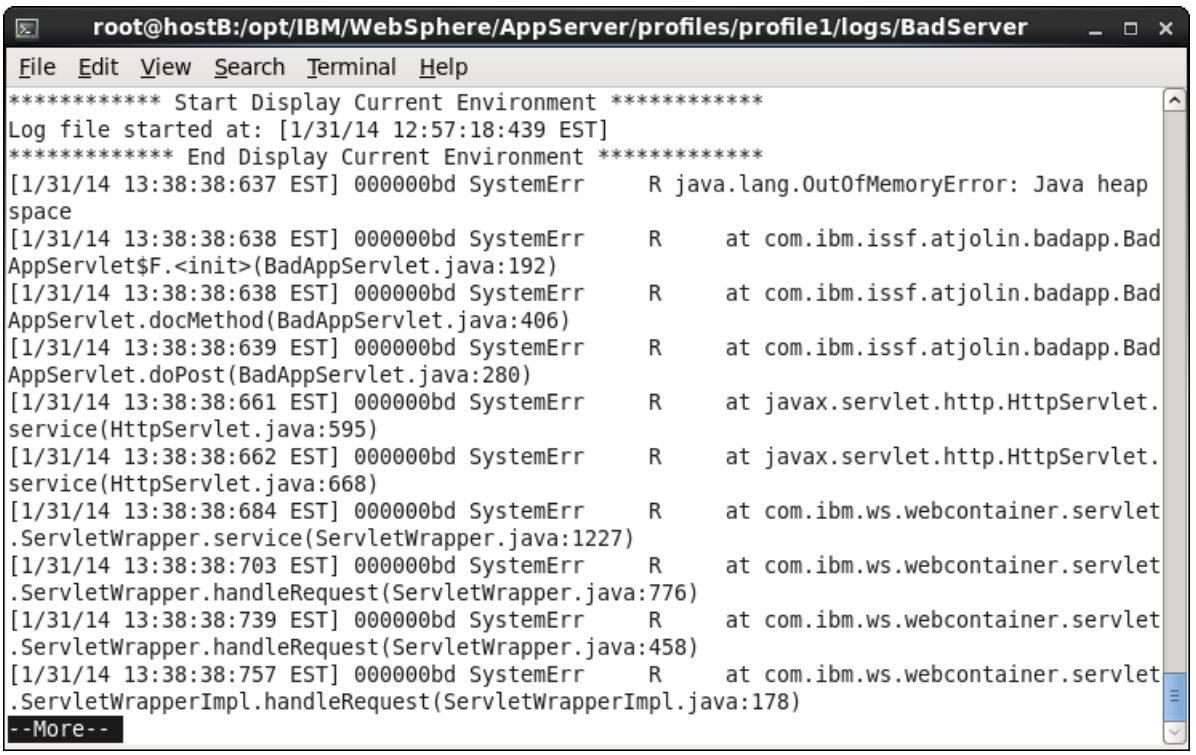


__ b. Eventually you might see an error message in the browser.

__ 8. Examine log files.

__ a. Open an ssh terminal on **hostB**, and navigate to **`<profiles_root>/profile1/logs/BadServer`**

__ b. View the **`SystemErr.log`** file.



__ c. Open the **`native_stderr.log`** file with a text editor such as vi, and search for **`JVMDUMP010I.`**

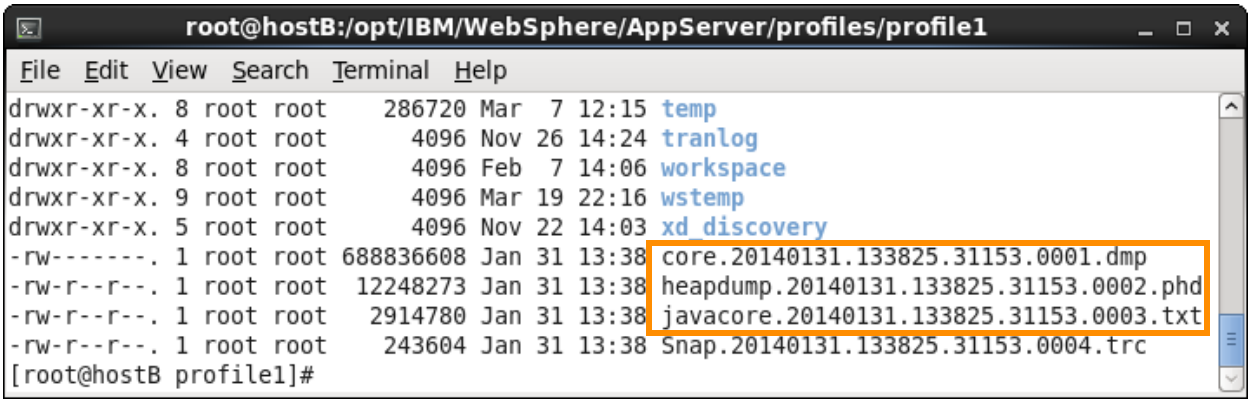
```
JVMDUMP039I Processing dump event "systhrow", detail "java/lang/OutOfMemoryErro
r" at 2014/01/31 13:38:25 - please wait.
JVMDUMP032I JVM requested System dump using '/opt/IBM/WebSphere/AppServer/profi
les/profile1/core.20140131.133825.31153.0001.dmp' in response to an event
JVMPORT030W /proc/sys/kernel/core_pattern setting "|/usr/libexec/abrt-hook-ccpp
 /var/spool/abrt %p %s %u %c" specifies that the core dump is to be piped to an
 external program.  Attempting to rename either core or core.31546.

JVMDUMP010I System dump written to /opt/IBM/WebSphere/AppServer/profiles/profil
e1/core.20140131.133825.31153.0001.dmp
<exclusive-start id="1567" timestamp="2014-01-31T13:38:36.566" intervalms="1126
2.213">
```

__ d. A dump event of “systhrow” occurred because of the OutOfMemoryError. By default, a system dump, a heap dump, a javacore, and a snap are written to the profile root directory, on the first occurrence of an OutOfMemoryError. Search again for **JVMDUMP010I** to see information about the other dumps.

```
JVMDUMP010I Heap dump written to /opt/IBM/WebSphere/AppServer/profiles/profile1
/heapdump.20140131.133825.31153.0002.phd
JVMDUMP032I JVM requested Java dump using '/opt/IBM/WebSphere/AppServer/profile
s/profile1/javacore.20140131.133825.31153.0003.txt' in response to an event
JVMDUMP010I Java dump written to /opt/IBM/WebSphere/AppServer/profiles/profile1
/javacore.20140131.133825.31153.0003.txt
JVMDUMP032I JVM requested Snap dump using '/opt/IBM/WebSphere/AppServer/profile
s/profile1/Snap.20140131.133825.31153.0004.trc' in response to an event
JVMDUMP010I Snap dump written to /opt/IBM/WebSphere/AppServer/profiles/profile1
/Snap.20140131.133825.31153.0004.trc
JVMDUMP013I Processed dump event "systhrow", detail "java/lang/OutOfMemoryError
```

__ 9. Gather the artifacts of this out-of-memory condition into a case for future analysis.

__ a. FTP the core file, the heap dump, and the javacore file from ***<profiles_root>*/profile1** to **/usr/labfiles/Case2**.



Information

Using FTP

If you are not familiar with using FTP, follow these steps.

Open a terminal window on **hostA**, and type the following commands:

- `cd /usr/labfiles/Case2`
- `ftp hostB`
- When prompted for Name, enter: `root`
- When prompted for Password, enter: `web1sphere`
- `cd /opt/IBM/WebSphere/AppServer/profiles/profile1`
- `get core.<time_stamp>.dmp`
- `get heapdump.<time_stamp>.phd`

- get javacore.<time_stamp>.txt
- quit

__ b. Rename the files in **/usr/labfiles/Case2** by using the following commands:

- **mv core.<time_stamp>.dmp core.oom.dmp**
- **mv heapdump.<time_stamp>.phd heap.oom.phd**
- **mv javacore.oom.txt javacore.<time_stamp>.txt**

Section 4: Use IBM Support Assistant and the Memory Analyzer to analyze OOM artifacts

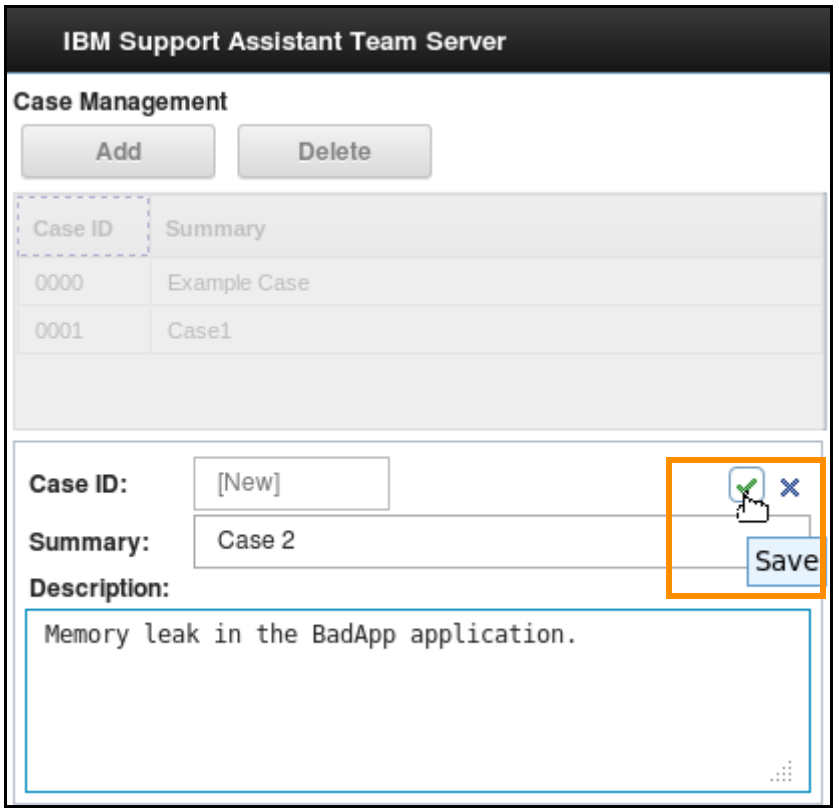
In this section, you create a case in IBM Support Assistant to hold the memory dump files that the OutOfMemoryError generated. You then use the Memory Analyzer tool to examine these memory dumps and find the root cause of the error.

__ 1. Create an ISA case and add OOM artifacts to the case.

__ a. In the IBM Support Assistant, click **Cases** to start the Case Management tool.

__ b. Click **Add**, and enter the following information about the case.

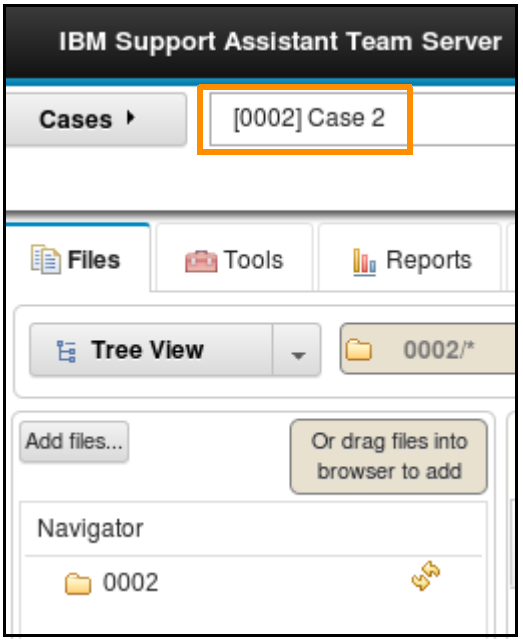
- Summary: **Case 2**
- Description: **Memory leak in the BadApp application.**



__ c. Click the green check mark to save the case.

__ d. Click anywhere outside of the Case Management tool to close it.

__ e. Make sure that **Case2** is selected in the cases menu, and click the **Files** tabs.

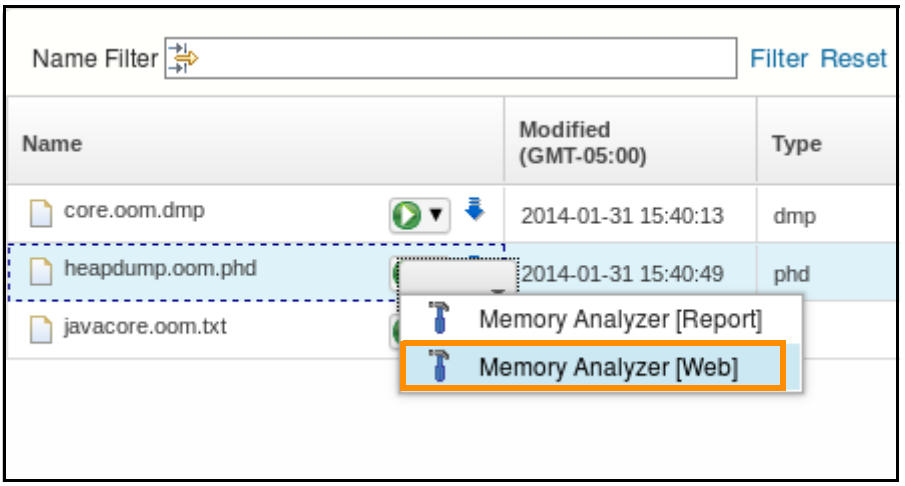


__ f. Click **Add files**, navigate to **`/usr/labfiles/Case2`**, and select the core, heap dump, and javacore files.

__ 2. Use the Memory Analyzer web tool to analyze the heap dump file.

__ a. Click the **Files** tab for Case 2**.**

__ b. You see the three files that you just added to the case. Use the Quick Launcher tool to open the Memory Analyzer web tool on the **`heap.oom.phd`** file.



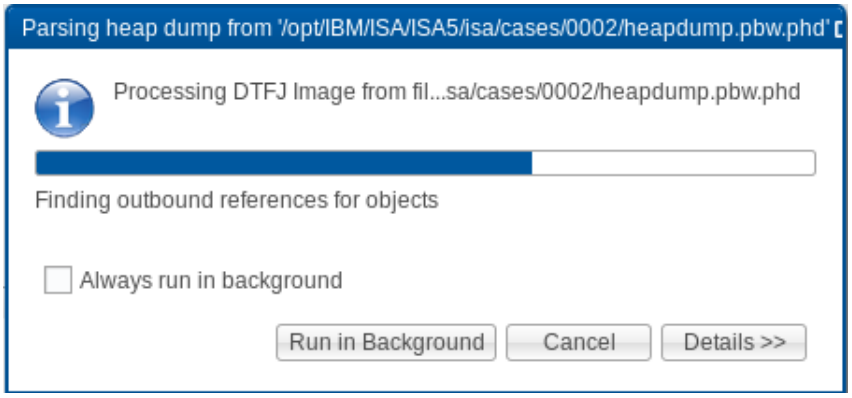
__ c. Click **Memory Analyzer [Web]** to start the tool.

__ d. On the Run Tool pane, click **Submit**.

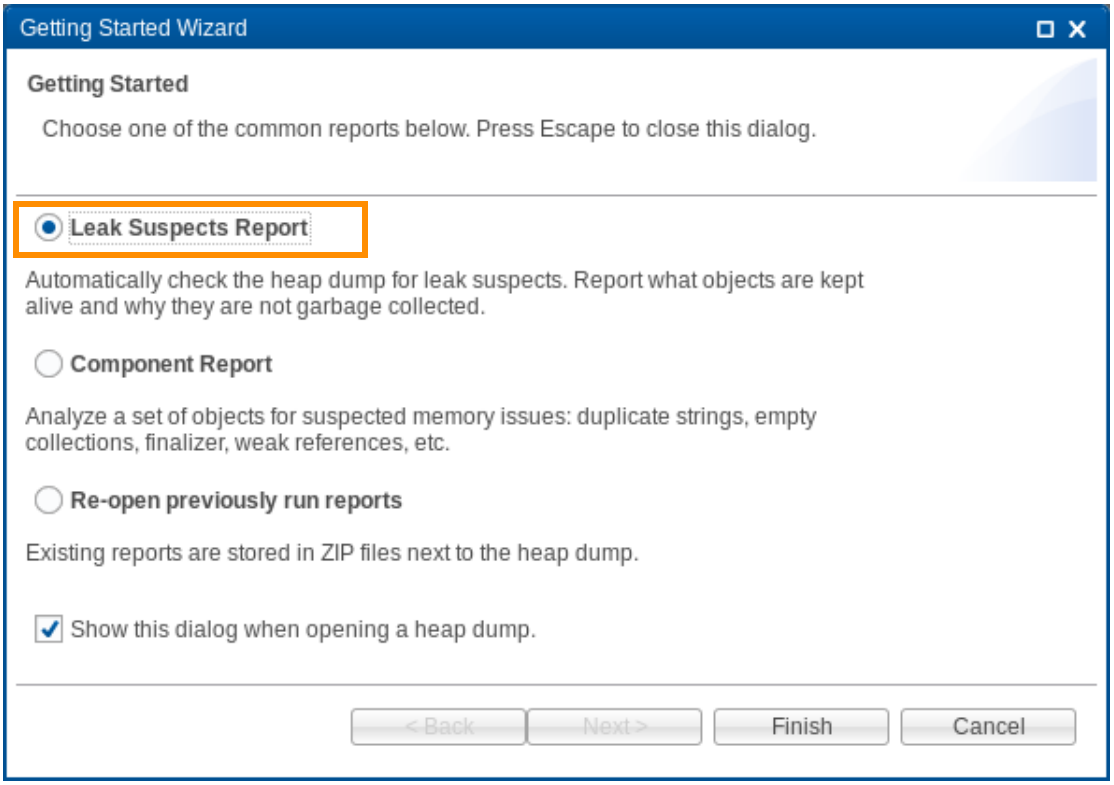
__ e. In the “This Connection is Untrusted” window, expand “**I Understand the Risks**”, and click **Add Exception**.

__ f. On the Add Security Exception pane, click **Confirm Security Exception**.

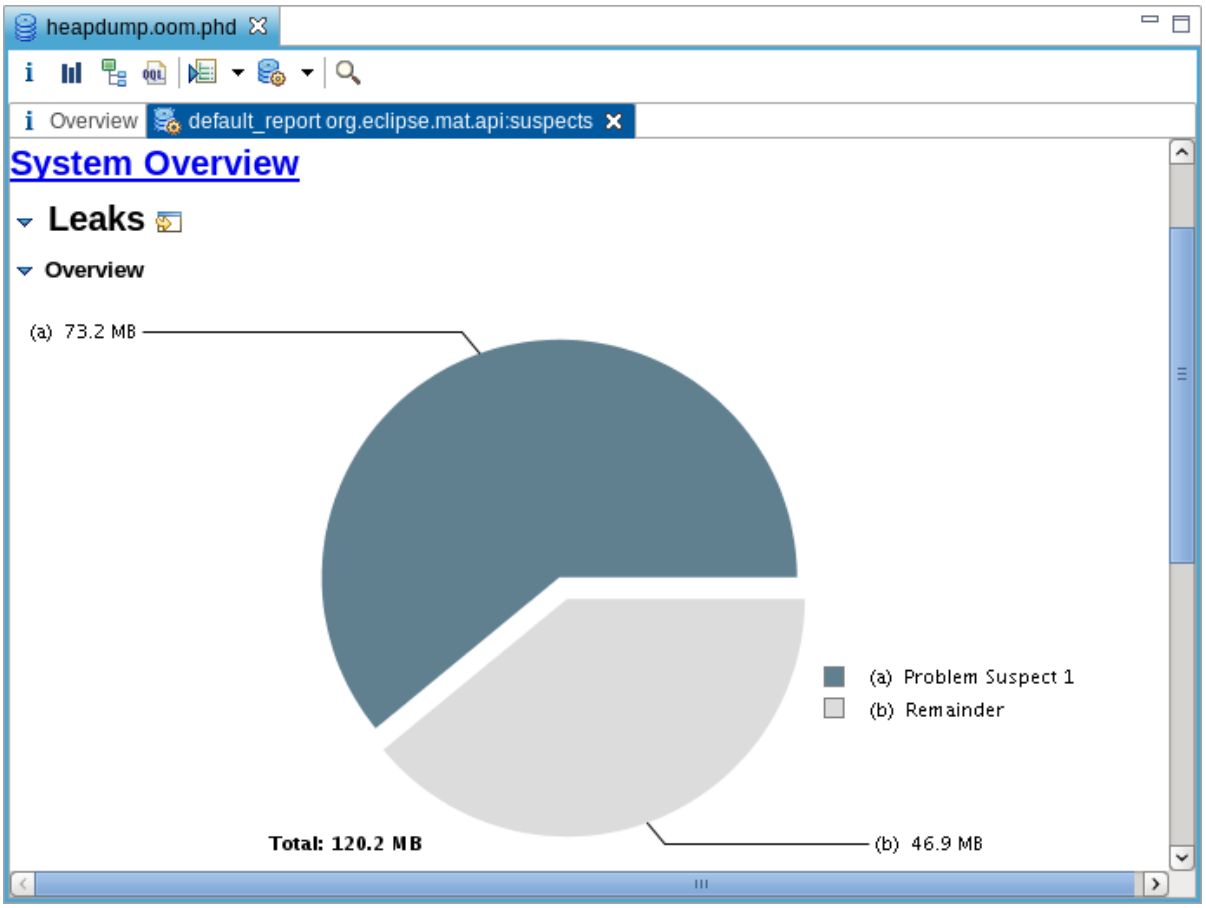
__ g. The Memory Analyzer tool starts as a new tab in the ISA web page. Wait several minutes for the tool to process the heap dump data.



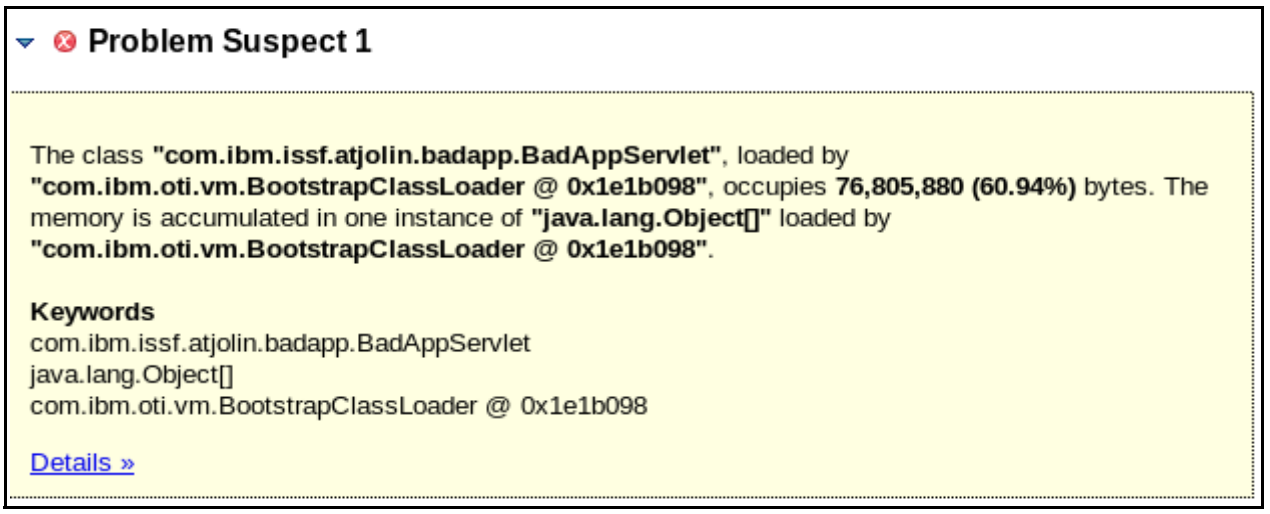
__ h. On the Getting Started Wizard, select **Leak Suspects Report**, and click **Finish**.



__ i. Wait for more processing. When complete, you see a report in the Memory Analyzer tool. Click the **default_report** tab to see the Leak Suspects report.



__ j. Scroll down to see details about the leak suspects.



__ k. The BadAppServlet is identified as the primary object that might be leaking memory. Click the **Details** link for more information.

Shortest Paths To the Accumulation Point

Class Name	Shallow Heap	Retained Heap
java.lang.Object[76] @ 0x7338f08	320	76,805,720
java.util.ArrayList @ 0x4210e48	24	76,805,744
class com.ibm.issf.atjolin.badapp.BadAppServlet @ 0x39fa9b0 **System Class**	80	76,805,880
com.ibm.oti.vm.BootstrapClassLoader @ 0x1e1b098 System Class	**104**	**109,352**

Accumulated Objects

Class Name	Shallow Heap	Retained Heap	Percentage
class com.ibm.issf.atjolin.badapp.BadAppServlet @ 0x39fa9b0	80	76,805,880	60.94%
java.util.ArrayList @ 0x4210e48	24	76,805,744	60.94%
java.lang.Object[76] @ 0x7338f08	320	76,805,720	60.94%
com.ibm.issf.atjolin.badapp.BadAppServlet$C @ 0x1edcce0	24	1,024,072	0.81%
com.ibm.issf.atjolin.badapp.BadAppServlet$C @ 0x1edccf8	24	1,024,072	0.81%

__ l. The details show the shortest path to the accumulation point and the accumulated objects. The shallow heap is the number of bytes that an object occupies. The retained heap is the number of bytes occupied by all the child objects.

Information

A note on Java heap terminology:

Shallow heap: The amount of memory that one object consumes. Depending on the operating system architecture, an object needs 32 bits or 64 bits per reference, 4 bytes per “Integer”, 8 bytes per “Long”, and others. Depending on the heap dump format, the size can be adjusted to provide a more realistic consumption of the JVM.

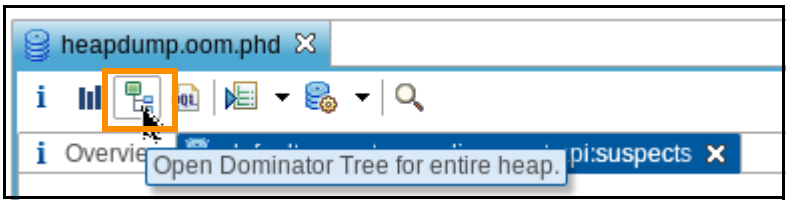
Retained set: One or more objects plus any objects that are referenced, directly or indirectly, only from those original objects. The retained set is the set of objects that garbage collection removes when an object, or multiple objects, are collected.

Retained heap, or retained size: The total heap size of all the objects in the retained set. This value is the amount of memory that the objects consume and is kept alive by the objects at the root of the retained set. In general terms, the shallow heap of an object is the size of the object in the heap. The retained size of the same object is the amount of heap memory that is freed when the object is collected.

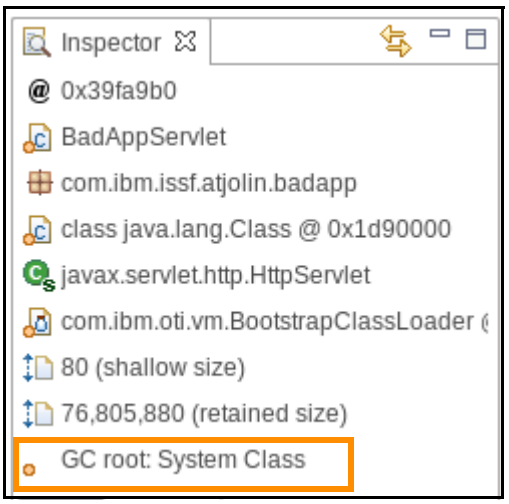
A **memory accumulation point** is where the memory of many small objects is accumulated under one object. Accumulation points are identified from the large difference in the retained size of an object and the retained size of its child objects.

__ 3. View the Denominator Tree for the entire heap.

__ a. On the heap dump tab, click the **Denominator Tree** icon in the toolbar.



__ b. The Denominator Tree is displayed in a new tab. Select any entry in the tree to see more details in the Inspector view. The object at the top of the tree(BadAppServlet) has the largest retained heap size. Click **GC root: System Class** to see information about its GC root.



Information

Garbage collection roots

A garbage collection root is an object that is accessible from outside the heap. For the BadAppServlet, the GC root is shown as "System Class". A system class is a class that the bootstrap loader loads, or the system Classloader. For example, this category includes all classes in the **`rt.jar`** file, such as those classes in the **`java.util.*`** package.

Information

Memory leaks degrade performance

When a memory leak is discovered as the source of poor performance, not much tuning on the server side can remedy the problem. Increasing the Java heap size delays the inevitable out-of-memory condition.

By using a tool such as the Memory Analyzer, administrators can gather enough information to determine the cause of OutOfMemory exceptions they are seeing in their log files. Mostly likely, the problem is the application that is installed on their server. The next step is for the administrator to communicate the results of the Memory Analyzer analysis to the application developers. Then, the developers must begin reviewing their code to determine what is causing the data structures to leak memory.

End of exercise

Exercise review and wrap-up

In this exercise, you installed a specially prepared application that was developed to exhibit certain types of poor performance behavior, such as heap exhaustion and hung threads. You used the BadApp application to create two scenarios, namely:

- Servers failing due to java.lang.OutOfMemory errors
- Servers slow down or stop working due to hanging threads

In the first scenario, you learned how to use the Memory Analyzer for analyzing a Java heap dump caused by a java.lang.OutOfMemory error due to a faulty servlet. In the second scenario, you analyzed a javacore file of a hanging application server to determine the cause of the hang by using the IBM Thread and Monitor Dump Analyzer.

Exercise 9. Tuning JDBC connection pools and enabling servlet caching

What this exercise is about

In this exercise, you examine the connection pool configuration for the Trade data sources. You use Tivoli Performance Viewer and Apache JMeter to monitor, tune, and load test connection pools.

What you should be able to do

At the end of this exercise, you should be able to:

- Examine and configure connection pool performance parameters by using the administrative console
- Configure the Tivoli Performance Viewer to monitor connection pool performance statistics
- Use Apache JMeter to load test an application
- Monitor an application by using Tivoli Performance Viewer and Apache JMeter
- Tune the connection pool performance parameters after analyzing results from the Tivoli Performance Viewer
- Enable servlet caching for an application server
- Use the Dynamic Cache Monitor to view caching activity and cache contents
- Monitor the performance of the DayTrader application when servlet caching is enabled

Introduction

Each time an application attempts to access a back-end store (such as a database), it requires resources to create, maintain, and release a connection to that data store. To ease the strain that this process can place on overall application resources, the application server enables you to establish a pool of back-end connections that applications can share. Connection pooling spreads the connection work across several user requests, conserving application resources for future requests.

The goal of tuning the connection pool is to ensure that each thread that needs a connection to the database has one, and that requests are not queued up waiting to access the database. For the DayTrader application, each task makes a query against the database. Because each thread does a task, each concurrent thread needs a database connection. Typically, all

requests come in over HTTP and run on a web container thread. Therefore, the maximum connection pool size should be at least as large as the maximum size of the web container thread pool.

Data source statement cache size specifies the number of prepared JDBC statements that can be cached per connection. The WebSphere Application Server data source optimizes the processing of prepared statements and callable statements by caching those statements that are not being used in an active connection. If your application uses many statements like DayTrader does, then increasing this parameter can improve application performance.

Requirements

Exercise 3: Installing DayTrader and **Exercise 4: Use Apache JMeter to load test DayTrader** must be successfully completed before you can complete this exercise. At the end of Exercise 4, you created an Apache JMeter test plan, **`/usr/labfiles/Test_Plans/DayTrader_baselineXX_test.jmx,`** which you are going to use in this exercise.

9.1. Tuning JDBC connection pools

In the following exercise, you explore the effects of tuning key JDBC connection pool parameters. For each application server configuration change, it is necessary to restart the server. Also, for consistent testing results, it is necessary to restore the Trade database on the DB2 server.

Note

<*profiles_root*> = /opt/IBM/WebSphere/AppServer/profiles

Section 1: Start the WebSphere servers

If the Deployment Manager, node agent, and TradeServer1 are already running, stop and restart them now.

__ 1. On **hostA**, start the Deployment Manager.

__ a. Open a terminal window and enter the command:
cd <*profiles_root*>/Dmgr/bin

__ b. Enter the command:
./startManager.sh

__ 2. Restore the DayTrader database.

__ a. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ b. Enter the command: **su - db2inst1**

__ c. Enter the command: **/usr/labfiles/db2_scripts/db2restore.sh**

__ d. Enter the command: **/usr/labfiles/db2_scripts/db2unquiese.sh**

__ 3. On **hostB**, start the node agent and TradeServer1.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
***cd* <*profiles_root*>/profile1/bin**

__ c. Enter the command:
./startNode.sh

__ d. After the node agent starts, start TradeServer1 by entering the command:
./startServer.sh TradeServer1

Section 2: Examine the connection pool properties for the Trade data source

In this section, you view the current connection pool properties for the Trade data source.

__ 1. Log in to the administrative console with the user ID **wasadmin** and password **web1sphere**

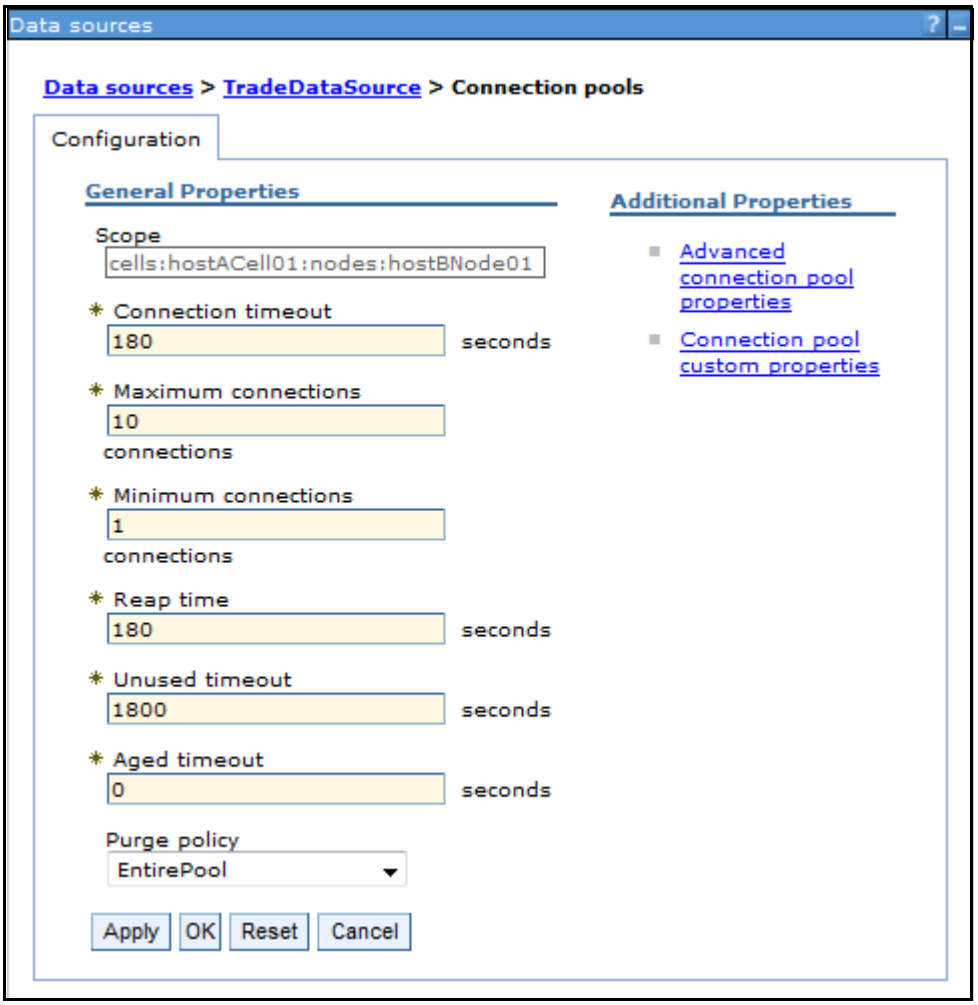
__ 2. Display the connection pool properties for the Trade data source.

__ a. Click **Resources > JDBC > Data sources**.

__ b. Verify that the scope is set to **All Scopes**.

__ c. Scroll down in the table of data sources and click **TradeDataSource**.

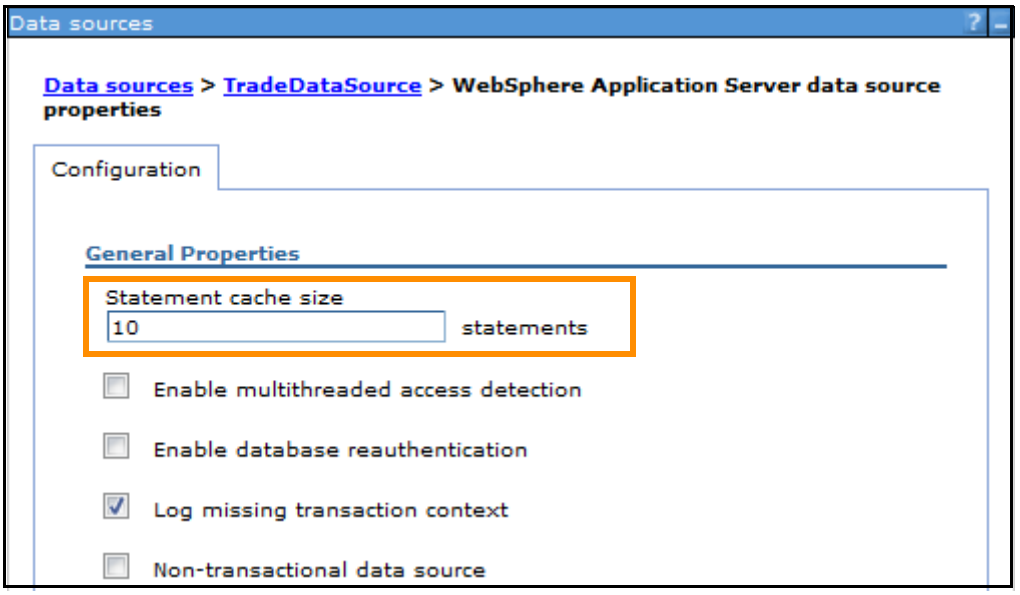
__ d. Under Additional Properties, click **Connection Pool Properties**. The connection pool properties are displayed.



__ e. This screen capture shows the default values. You typically adjust the Minimum and Maximum connections properties to what your application requires.

__ 3. Examine the statement cache for the Trade data source.

__ a. Click **TradeDataSource** in the breadcrumb trail. Under Additional Properties, click **WebSphere Application Server data source properties**.



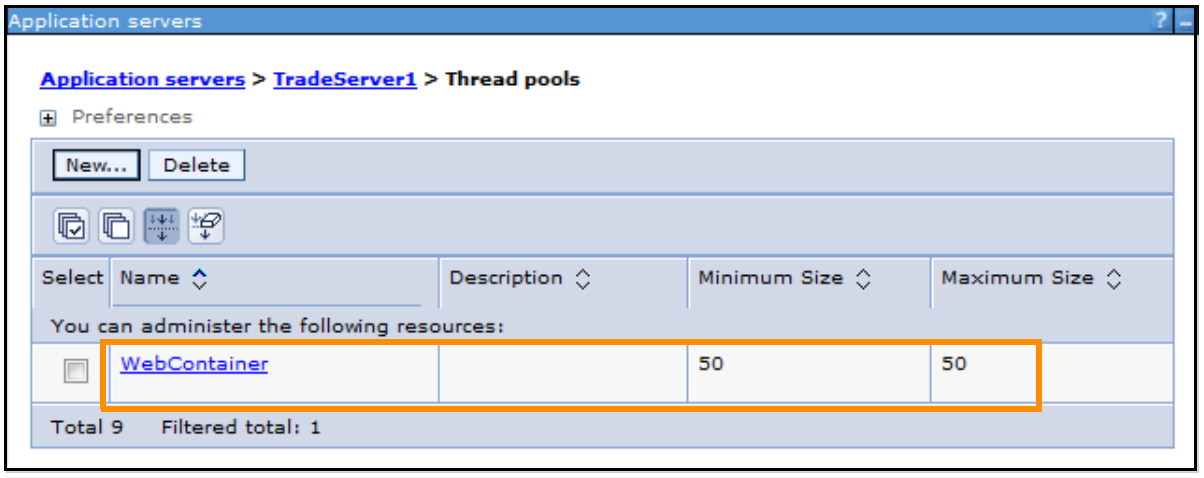
__ b. The default value of the Statement cache size is 10. This property is the number of statements that can be cached per connection. The application server caches a statement after that statement is closed in the application code. Depending on the number of prepared statements and callable statements that your application uses, you might have to increase the size of this cache.

__ c. You do not change any properties now, so click **Cancel**.

__ 4. Examine the web container thread pool size.

__ a. From the administrative console, click **Servers > Server Types > WebSphere application servers > TradeServer1**.

__ b. On the Configuration tab, scroll down to **Additional Properties**, and click **Thread pools**.



__ c. Observe that the default value of the maximum web container thread pool size is 50. Typically, all requests come in over HTTP and are run on a web container thread. Therefore, the maximum connection pool size should be no larger than the maximum size of the web container thread pool. For tuning DayTrader in this exercise, use the maximum size of the web container thread pool (50) as an upper limit for maximum connection pool size.

__ d. You do not change any properties now, so click **Cancel**.

__ 5. Start Apache JMeter if it is not already started.

__ a. On **hostA**, open a terminal window and enter the command:
`cd /opt/apache-jmeter/bin`

__ b. Enter the command:
`./jmeter.sh &`

__ 6. Open the **`DayTrader_baseline80_test.jmx`** test plan**.**

__ a. Click **File > Open**, and navigate to
`/usr/labfiles/Test_Plans/DayTrader_baseline80_test.jmx`

__ b. In the JMeter tree, click **Thread Group** and change the Loop Count to **100**.

__ 7. Prepare to load test the DayTrader.

__ a. Start a web browser, and go to **`http://hostB:9081/daytrader`**

__ b. Click the **Configuration** tab, and click the **Reset DayTrader** link. Keep this browser open as you are going to reset the DayTrader for each run of the load test.

__ c. Verify that the reset completes successfully.

__ d. In Apache JMeter, click **Summary Report**.

__ e. Clear the Summary Report if necessary by clicking the broom icon.

__ 8. Run the test plan.

__ a. In JMeter, click **Run > Start** (or click the green arrow) to start the warm-up test.

__ b. Run the test three more times and record the performance results (average response time and throughput). Remember to reset DayTrader and clear the Summary Report before each test run.

Table 15: Verify baseline test

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

__ 9. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

Section 3: Monitor a JDBC connection pool with Tivoli Performance Viewer

In this section, you use Tivoli Performance View to monitor some important PMI metrics for a data source connection pool.

__ 1. Start monitoring TradeServer1 with the Tivoli Performance Viewer.

__ a. From the administrative console, click **Monitoring and Tuning > Performance Viewer > Current activity**.

__ b. Select **TradeServer1** and click **Start Monitoring**.

__ c. When the **Collection Status** for TradeServer1 and node agent becomes **Monitored**, click the link for **TradeServer1**. This action starts the Tivoli Performance Viewer.

__ d. In the Tivoli Performance Viewer navigation tree, expand **Performance Modules**.

__ e. Expand **JDBC Connection Pools > DB2 Universal JDBC Driver Provider Only (XA)** and select the **jdbc/TradeDataSource** check box**.**

__ f. At the top of the navigation tree, click **View Modules**.

__ g. You now see the graph and the list of counters below the graph. In the list, select the **WaitingThreadCount, WaitTime,** and **PrepStmtCacheDiscardCount** check boxes. Clear the other check boxes.

Information

Connection pool basic PMI statistics

- **CreateCount:** The total number of connections that are created.
- **CloseCount:** The total number of connections that are closed.
- **PoolSize:** The size of the connection pool.
- **FreePoolSize:** The number of free connections in the pool.
- **WaitingThreadCount:** The average number of threads that are concurrently waiting for a connection.
- **PercentUsed:** Average percent of the pool that is in use. The value is based on the total number of configured connections in the ConnectionPool, not the current number of connections.
- **UseTime:** The average time in milliseconds that a connection is used; the difference between the time at which the connection is allocated and returned. This value includes the JBDC operation time.
- **WaitTime:** The average waiting time in milliseconds until a connection is granted.
- **PrepStmtCacheDiscardCount:** The number of statements that are discarded because the cache is full.

__ 2. Run the JMeter test.

__ a. Set the loop count to **Forever**.

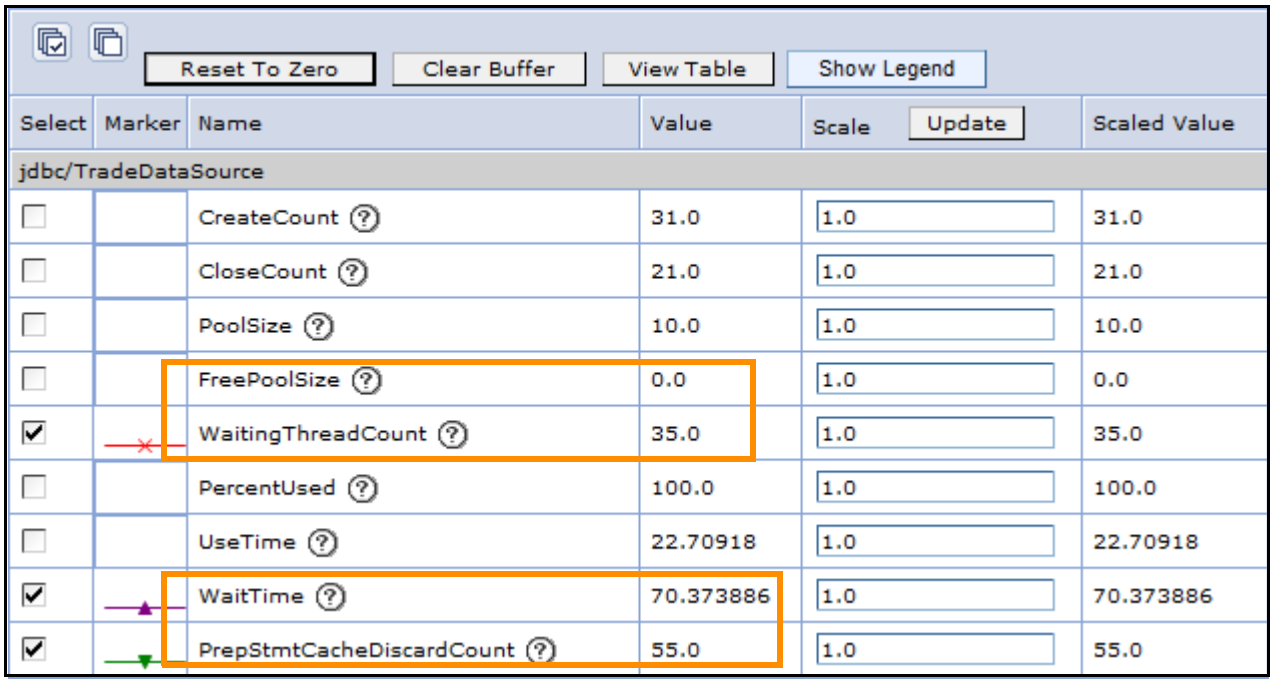
__ b. Reset DayTrader.

__ c. Start the test run.

__ 3. View the data source counters in Tivoli Performance Viewer.

__ a. View the JMeter performance results. Wait until the average response time and throughput reach steady values.

__ b. Return to the Tivoli Performance Viewer and view the statistics for the connection pool.



__ c. You are likely to find that the FreePoolSize is 0.0 and the values of the WaitingThreadCount, WaitTime, and PrepStmtCacheDiscardCount are relatively high. Record your values here.

- FreePoolSize_____
- WaitingThreadCount_____
- WaitTime_____
- PrepStmtCacheDiscardCount_____

__ 4. Stop the JMeter test.

__ 5. Stop monitoring TradeServer1 with Tivoli Performance Viewer.

Section 4: Tune and test maximum connections

In general, the performance goals for tuning the maximum connections property are to reduce the number of waiting threads, and to minimize the average time that a thread spends waiting to get a connection. It is also a goal to not have too many pooled connections that are not used.

__ 1. Change Maximum connections to 10 + your value of WaitingThreadCount (In the example, 10+35=45).

__ a. From the administrative console, click **Resources > JDBC > Data sources**.

__ b. Verify that the scope is set to **All Scopes**.

__ c. Scroll down in the table of data sources and click **TradeDataSource**.

__ d. Click **Connection pool properties**.

__ e. Increase the Maximum connections property by the amount of your measured WaitingThreadCount (10 + WaitingThreadCount).

__ f. Click **OK**.

__ g. Click **Save** to save the changes.

__ h. After the node is synchronized click **OK**.

__ 2. Stop TradeServer1.

__ 3. Restore the DayTrader database.

__ a. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ b. Enter the command: **`su - db2inst1`**

__ c. Enter the command: **`/usr/labfiles/db2_scripts/db2restore.sh`**

__ d. Enter the command: **`/usr/labfiles/db2_scripts/db2unquiese.sh`**

__ 4. Start TradeServer1.

__ 5. Reset DayTrader.

__ a. Start a web browser, and go to **`http://hostb:9081/daytrader`**

__ b. Click the **Configuration** tab, and click the **Reset DayTrader** link. Keep this browser open because you are going to reset the DayTrader for each run of the load test.

__ c. Verify that the reset completes successfully.

__ 6. Run the JMeter test plan.

__ a. In the JMeter tree, click **Thread Group** and change the Loop Count to **100**.

__ a. Click **Run > Start** (or click the green arrow) to start the warm-up test.

__ b. Run the test three more times and record the performance results (average response time and throughput). Remember to reset DayTrader before each test run.

Table 16: Test run results with increased Maximum connections

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

__ 7. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

__ 8. Start monitoring TradeServer1 with the Tivoli Performance Viewer.

__ a. From the administrative console, click **Monitoring and Tuning > Performance Viewer > Current activity**.

__ b. Select **TradeServer1** and click **Start Monitoring**.

__ c. When the **Collection Status** for TradeServer1 and node agent becomes **Monitored**, click the link for **TradeServer1**. This action starts the Tivoli Performance Viewer.

__ d. In the Tivoli Performance Viewer navigation tree, expand **Performance Modules**.

__ e. Expand **JDBC Connection Pools > DB2 Universal JDBC Driver Provider Only (XA)** and select the **jdbc/TradeDataSource** check box.

__ f. At the top of the navigation tree, click **View Modules**.

__ g. You now see the graph and the list of counters below the graph. In the list, select the **WaitingThreadCount, WaitTime,** and **PrepStmtCacheDiscardCount** check boxes. Clear the other check boxes.

__ 9. Run the JMeter test.

__ a. Set the loop count to **Forever.**

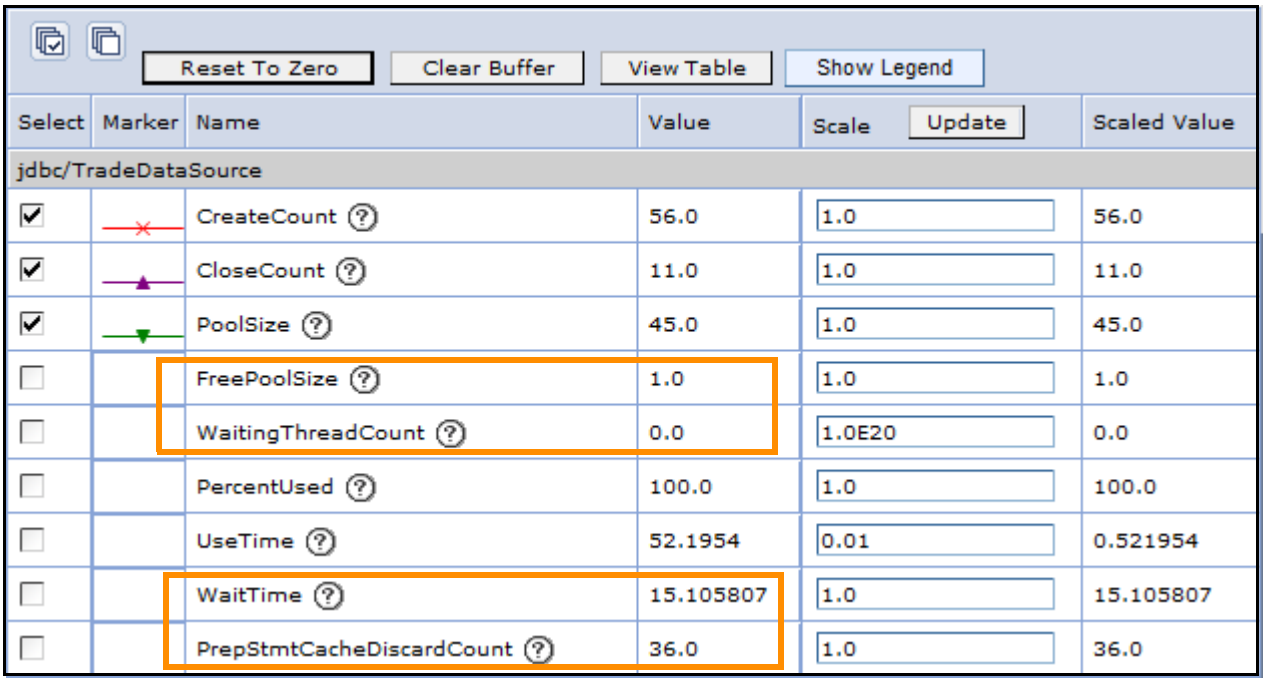
__ b. Reset DayTrader.

__ c. Start the test run.

__ 10. View the data source counters in Tivoli Performance Viewer.

__ a. View the JMeter performance results. Wait until the average response time and throughput reach steady values.

__ b. Return to the Tivoli Performance Viewer and view the statistics for the connection pool.



Reset To Zero | Clear Buffer | View Table | Show Legend

Select	Marker	Name	Value	Scale	Update	Scaled Value
jdbc/TradeDataSource						
☑		CreateCount	56.0	1.0		56.0
☑		CloseCount	11.0	1.0		11.0
☑		PoolSize	45.0	1.0		45.0
☐		FreePoolSize	1.0	1.0		1.0
☐		WaitingThreadCount	0.0	1.0E20		0.0
☐		PercentUsed	100.0	1.0		100.0
☐		UseTime	52.1954	0.01		0.521954
☐		WaitTime	15.105807	1.0		15.105807
☐		PrepStmtCacheDiscardCount	36.0	1.0		36.0

__ c. As a result of tuning the **maximum connections** property, you might see a few free connections in the pool or a value of **FreePoolSize** equal to 0.0. Most likely, you see the value of **WaitingThreadCount** is 0.0. **WaitTime** and **PrepStmtCacheDiscardCount** are likely to decease. Record your values here.

- FreePoolSize_____
- WaitingThreadCount_____
- WaitTime_____
- PrepStmtCacheDiscardCount_____

__ 11. Stop the JMeter test.

__ 12. Stop monitoring TradeServer1 with Tivoli Performance Viewer.

Section 5: Tune and test prepared statement cache

The data source statement cache size can be tuned by using a few different methods. One technique is to review the application code for all unique prepared statements, and ensure that the cache size is larger than that value. The other option is to iteratively increase the cache size and run the application under peak steady state load until the PMI metrics report no more cache discards.

__ 1. Change the Statement cache size to 10 + your value of PrepStmtCacheDiscardCount (In this example, 10+36=46, try rounding up to 50).

__ a. From the administrative console, click **Resources > JDBC > Data sources**.

__ b. Verify that the scope is set to **All Scopes**.

__ c. Scroll down in the table of data sources and click **TradeDataSource**.

__ d. Under Additional Properties, click **WebSphere Application Server data source properties.**

__ e. Change the Statement cache size to 10 + your value of PrepStmtCacheDiscardCount.

__ f. Click **OK**.

__ g. Click **Save** to save the changes.

__ h. After the node is synchronized click **OK**.

__ 2. Stop TradeServer1.

__ 3. Restore the DayTrader database.

__ a. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ b. Enter the command: **`su - db2inst1`**

__ c. Enter the command: **`/usr/labfiles/db2_scripts/db2restore.sh`**

__ d. Enter the command: **`/usr/labfiles/db2_scripts/db2unquiese.sh`**

__ 4. Start TradeServer1.

__ 5. Reset DayTrader.

__ 6. Run the JMeter test plan.

__ a. In the JMeter tree, click **Thread Group** and change the Loop Count to **100**.

__ a. Click **Run > Start** (or click the green arrow) to start the warm-up test.

__ b. Run the test three more times and record the performance results (average response time and throughput). Remember to reset DayTrader before each test run.

Table 17: Test run results with increased Statement cache size

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

__ 7. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

__ 8. Start monitoring TradeServer1 with the Tivoli Performance Viewer.

__ a. From the administrative console, click **Monitoring and Tuning > Performance Viewer > Current activity**.

__ b. Select **TradeServer1** and click **Start Monitoring**.

__ c. When the **Collection Status** for TradeServer1 and node agent becomes **Monitored**, click the link for **TradeServer1**. This action starts the Tivoli Performance Viewer.

__ d. In the Tivoli Performance Viewer navigation tree, expand **Performance Modules**.

__ e. Expand **JDBC Connection Pools > DB2 Universal JDBC Driver Provider Only (XA)** and select the **jdbc/TradeDataSource** check box.

__ f. At the top of the navigation tree, click **View Modules**.

__ g. You now see the graph and the list of counters below the graph. In the list, select the check boxes for **WaitingThreadCount, WaitTime,** and **PrepStmtCacheDiscardCount**. Clear the other check boxes.

__ 9. Run the JMeter test.

__ a. Set the loop count to **Forever.**

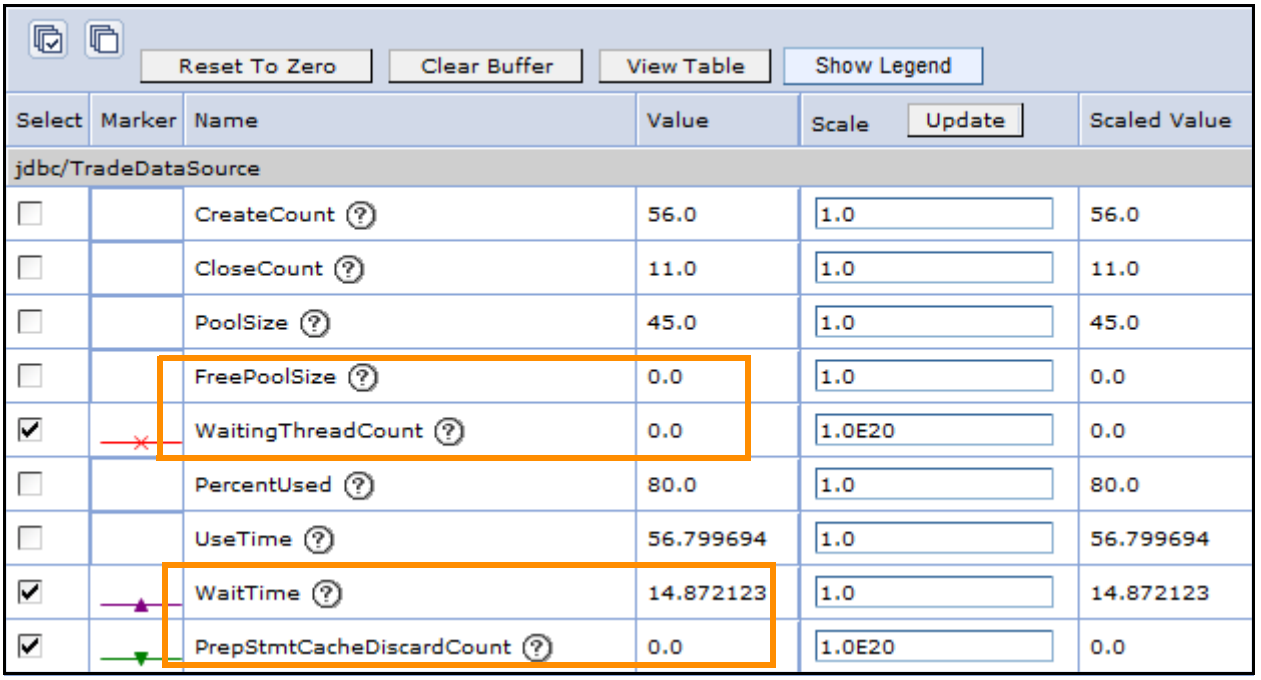
__ b. Reset DayTrader.

__ c. Start the test run.

__ 10. View the data source counters in Tivoli Performance Viewer.

__ a. View the JMeter performance results. Wait until the average response time and throughput reach steady values.

__ b. Return to the Tivoli Performance Viewer and view the statistics for the connection pool.



__ c. The free pool size might be 0.0, or you might see some free connections on the pool (FreePoolSize is not 0.0) and most likely the value of WaitingThreadCount is 0.0. WaitTime is likely to be smaller. PrepStmtCacheDiscardCount is likely to be 0.0. Record your values here.

- FreePoolSize_____
- WaitingThreadCount_____
- WaitTime_____

- PrepStmtCacheDiscardCount_____

__ 11. Stop the JMeter test.

__ 12. Stop monitoring TradeServer1 with Tivoli Performance Viewer.

9.2. Enabling servlet caching

Caching the output of servlets, commands, and JavaServer Page (JSP) improves application performance. WebSphere Application Server consolidates several caching activities that include servlets, web services, and WebSphere commands into one service that is called the dynamic cache. These caching activities work together to improve application performance, and share many configuration parameters that are set in the dynamic cache service of an application server. You can use the dynamic cache to improve the performance of servlet and JSP files by serving requests from an in-memory cache. Cache entries contain servlet output, the results of a servlet after it runs, and metadata.

The dynamic cache service works within an application server Java virtual machine (JVM), intercepting calls to cacheable objects. For example, it intercepts calls through a servlet service method, and either stores the output of the object to the cache or serves the content of the object from the dynamic cache.

Section 1: Enable servlet caching for TradeServer1

Enabling servlet caching specifies that if a servlet is started and it generates output to be cached, a cache entry is created containing not only the output, but also side effects of the invocation. These side effects can include calls to other servlets or JavaServer Pages (JSP) files, and also metadata about the entry, including timeout and entry priority information.

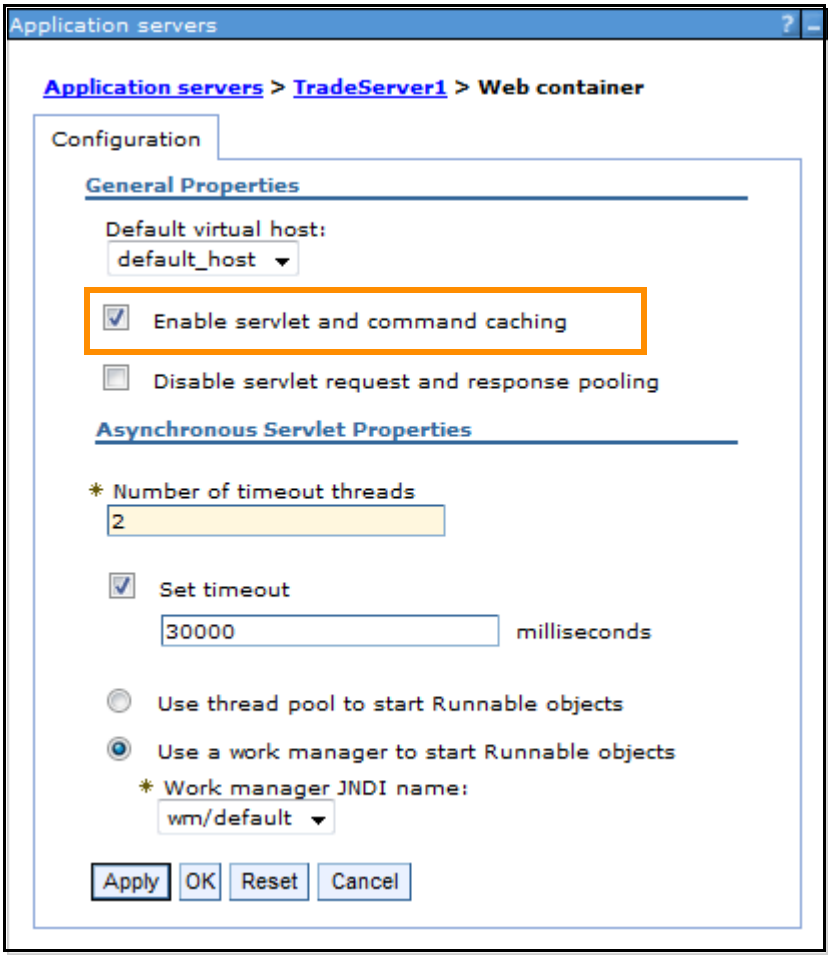
__ 1. Log in to the administrative console with the user ID **`wasadmin`** and password **`web1sphere`**

__ 2. Enable servlet caching for TradeServer1.

   __ a. In the administrative console, click **Servers > Server Types > WebSphere application servers > TradeServer1**.

   __ b. Click **Web Container Settings > Web container**.

__ c. Select the **Enable servlet and command caching** check box.



__ d. Click **OK**.

__ e. Click **Save** to save the configuration.

__ f. Wait for the node to synchronize, and click **OK**.

Information

Caching the Market Summary JSP

In the DayTrader application, a Market Summary is displayed on every access of a user's home page. This summary contains a list of the top five gaining and losing stocks, along with the current stock index and trading volume. This activity requires the execution of several database lookups and therefore significantly delays the loading of the user's home page. With servlet caching, the **`marketSummary.jsp`** can be cached, which eliminates these expensive database queries to improve the response time for the user home page.

Therefore, the Market Summary JSP is a good example to show what improvement you can achieve through caching to avoid complex server operations.

__ 3. Configure the Market Summary of the DayTrader to be cached. For every access of a user's home page a Market Summary is displayed by using the **`marketSummary.jsp`**.

Market Summary
2014-02-10

DayTrader Stock Index (TSIA)	102.76 (+1.00%)
Trading Volume	77178.0

Top Gainers

symbol	price	change
s:100	133.18	25.18
s:101	22.62	-9.38
s:102	50.74	-8.26
s:103	118.06	6.06
s:104	82.91	3.91

Top Losers

symbol	price	change
s:199	27.32	-10.68
s:198	144.23	-11.77
s:197	89.84	-13.16
s:196	134.18	-40.82
s:195	127.59	-24.41

A **`cachespec.xml`** must be created so that caching service knows what objects to cache. Here is an example of a **`cachespec.xml`** for the **`marketSummary.jsp`**.

```
<?xml version="1.0"?>
<!DOCTYPE cache SYSTEM "cachespec.dtd">
<cache>
        <cache-entry>
        <class>servlet</class>
        <name>/marketSummary.jsp</name>
        <property name="consume-subfragments">true</property>
        <property name="EdgeCacheable">true</property>
        <cache-id>
                <priority>3</priority>
                <timeout>60</timeout>
        </cache-id>
        <dependency-id>MarketSummary</dependency-id>
        </cache-entry>
</cache>
```

The refresh interval (timeout) for the cached object can be configured, and is set to 60 seconds in the example shown.

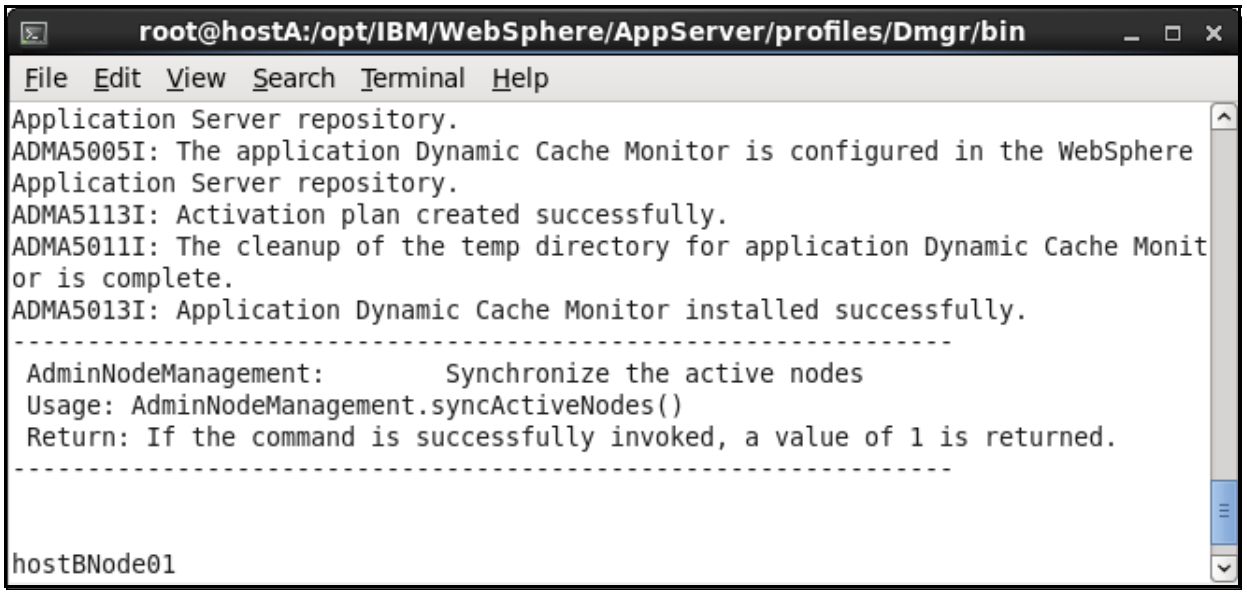
__ a. The **cachespec.xml** is typically placed in the WEB-INF directory of the application web module. Open an ssh terminal to **hostB**, and copy **/usr/labfiles/Cachespec/cachespec.xml** to **<*profiles_root*>/profile1/installedApps/hostACell01/DayTrader3-EE6.ear/web.war/WEB-INF**

__ 4. Install the Cache Monitor application on TradeServer1. Cache Monitor is an installable web application that provides a real-time view of the current state of dynamic cache. You use it to help verify that dynamic cache is operating as expected. The only way to manipulate the data in the cache is by using the Cache Monitor. It provides a GUI interface to manually change data.

__ a. On **hostA**, open a terminal window and enter the command: **cd <*profiles_root*>/Dmgr/bin**

__ b. Run the following command: **./wsadmin.sh -f /usr/labfiles/Cachespec/install_cachemon.py -user wasadmin -password web1sphere**

__ c. When the installation process completes, verify that you see the following message.



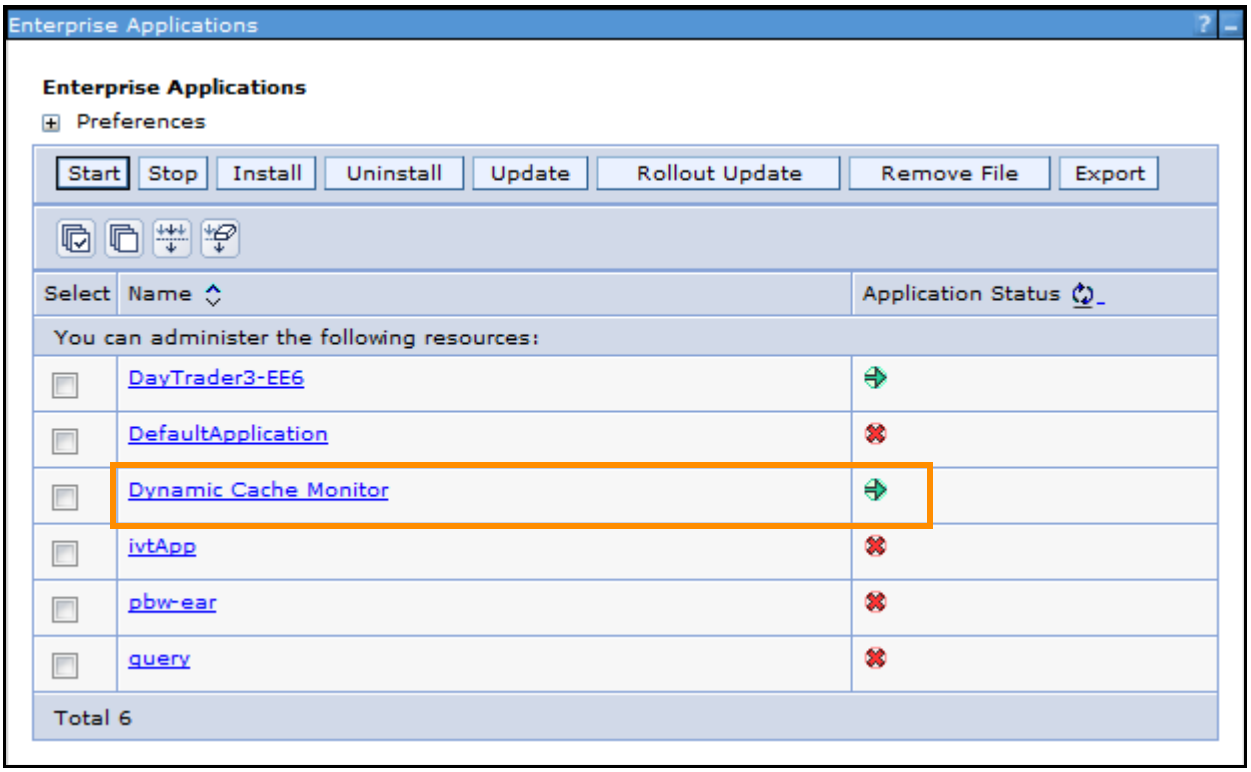
__ 5. Start TradeServer1 and test the Cache Monitor application.

__ a. Log in to the administrative console with the user **wasadmin** with password **web1sphere**

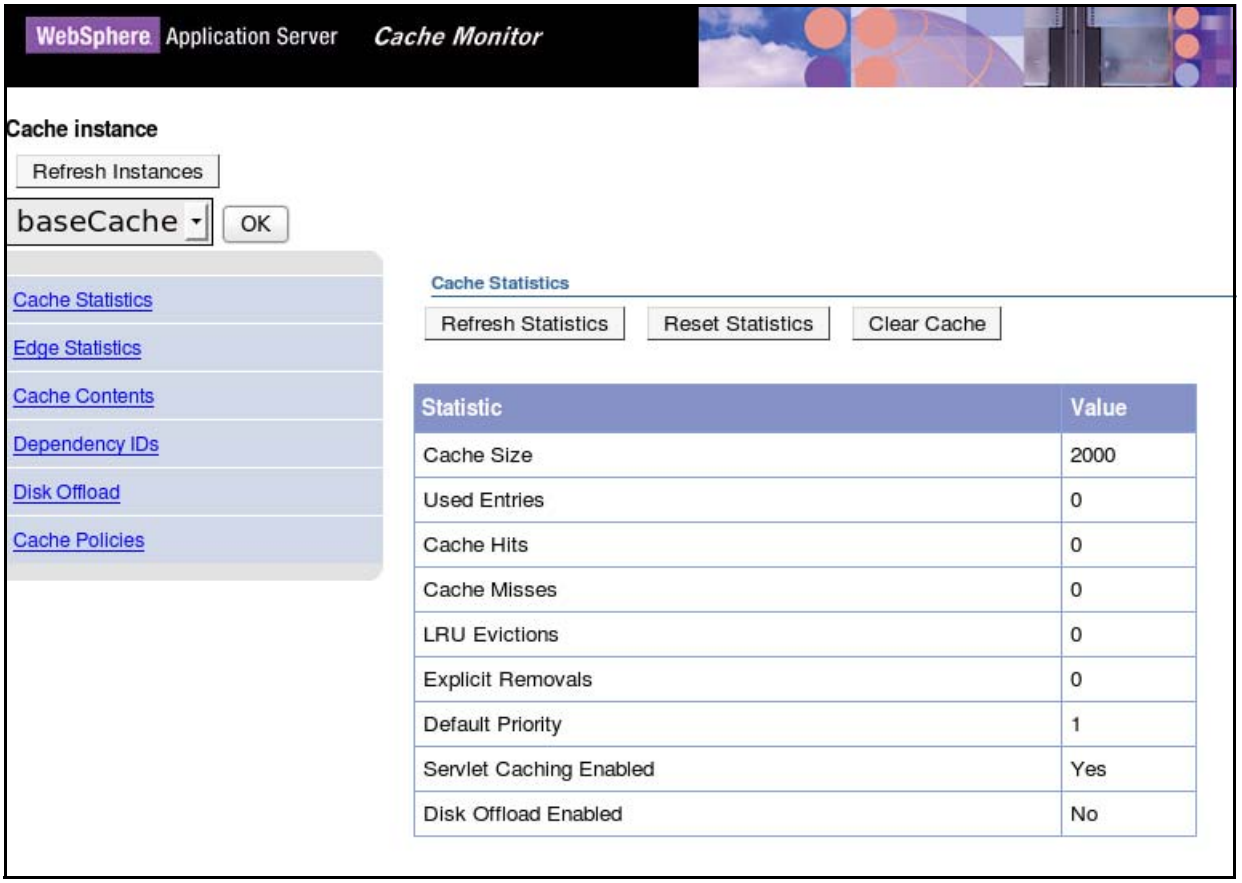
__ b. Start **TradeServer1**.

__ c. Click **Applications > Application Types > WebSphere enterprise applications**.

__ d. Verify that the **Dynamic Cache Monitor** is started.



__ e. Open a browser and enter the web address: **`http://hostb:9081/cachemonitor`**

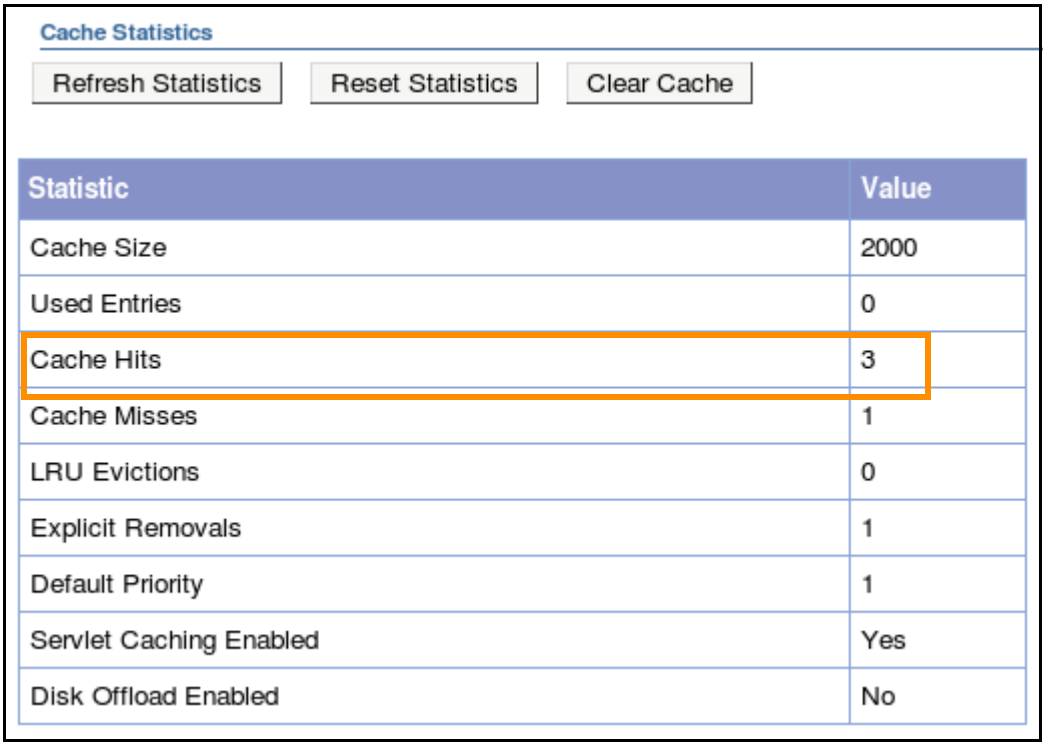


__ f. Open another web browser and go the DayTrader home page: **`http://hostb:9081/daytrader`**.

__ g. Click the **Configuration** tab.

__ h. Click the **Test DayTrader Scenario** link and use the reload key (F5) of the browser to step through the DayTrader scenario a few times. Make sure that you see the Market Summary that is displayed at least three times.

__ i. Go back to the Cache Monitor page and look for cache hits.



Statistic	Value
Cache Size	2000
Used Entries	0
Cache Hits	3
Cache Misses	1
LRU Evictions	0
Explicit Removals	1
Default Priority	1
Servlet Caching Enabled	Yes
Disk Offload Enabled	No

__ j. You might have to click **Refresh Statistics**. Then, you see some value for **Cache Hits**.

__ k. For this exercise, you use Cache Monitor to verify that caching is working correctly. Go back to DayTrader and press the Reload key (F5) a few more times to display the Market Summary. Then, back on the Cache Monitor, click **Cache Contents**.

Template	Cache ID	Timeout (seconds)	Dependency IDs
/daytrader/marketSummary.jsp	/daytrader/marketSummary.jsp:requestType=GET	60	MarketSummary

__ l. Verify that you see the Market Summary JSP in the cache.

Section 2: Run the Apache JMeter load test and examine the effects of caching

In this section, you load test the DayTrader with caching of the Market Summary enabled. You should see some performance improvement.

__ 1. Stop TradeServer1.

__ 2. Restore the DayTrader database.

__ a. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ b. Enter the command: **`su - db2inst1`**

__ c. Enter the command: **`/usr/labfiles/db2_scripts/db2restore.sh`**

__ d. Enter the command: **`/usr/labfiles/db2_scripts/db2unquiese.sh`**

__ 3. Start TradeServer1.

__ 4. Reset DayTrader.

__ 5. Run the JMeter test plan.

__ a. In the JMeter tree, click **Thread Group** and change the Loop Count to **100**.

__ a. Click **Run > Start** (or click the green arrow) to start the warm-up test.

__ b. Run the test three more times and record the performance results (average response time and throughput). Remember to reset DayTrader before each test run.

Table 18: Test run results with servlet caching enabled

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

__ 6. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

__ 7. Do you see a significant performance improvement after servlet caching is enabled?

__ 8. View the cache statistics in the Cache Monitor. Verify that thousands of objects are cached, and that there are hundreds of cache hits.

Section 3: Cleanup

If this exercise is the last exercise you do for several hours, you should stop the WebSphere servers.

__ 1. On **hostA**, stop the Deployment Manager.

__ a. Open a terminal window and enter the command:
cd <*profiles_root*>/Dmgr/bin

__ b. Enter the command:
./stopManager.sh -username wasadmin -password web1sphere

__ 2. On **hostB**, stop the node agent and TradeServer1.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
cd <*profiles_root*>/profile1/bin

__ c. Enter the command:
./stopNode.sh -username wasadmin -password web1sphere

__ d. Enter the command:
./stopServer.sh TradeServer1 -username wasadmin -password web1sphere

End of exercise

Exercise review and wrap-up

Tuning JDBC connection pools

In this exercise, you examined and configured connection pool parameters by using the administrative console. You configured the Tivoli Performance Viewer to monitor connection pool performance statistics. You used Apache JMeter to load test the DayTrader application. You also monitored the performance of the DayTrader application by using Tivoli Performance Viewer and Apache JMeter. Finally, you tuned some connection pool performance parameters after analyzing results from the Tivoli Performance Viewer.

Enable servlet caching.

In this exercise, you enabled servlet caching for TradeServer1 and configured a cachespec.xml to direct the dynamic cache service to cache the Market Summary JSP. You installed the Dynamic Cache Monitor Application and used it to examine caching activity and the contents of the cache. Finally, you used Apache JMeter to load test and monitor the performance of the DayTrader application with servlet caching enabled.

Exercise 10.Application Profiling with Java Health Center

What this exercise is about

The Java Health Center has many features for examining what is happening in a running JVM. This exercise covers the use of the Java Health Center for analyzing application performance. You use the Java Health Center to examine JVM characteristics that are related to application execution time, memory usage, and concurrency.

What you should be able to do

At the end of this exercise, you should be able to:

- Use the Java Health Center to find application methods with the longest execution time
- Use the Java Health Center to examine garbage collection behavior
- Use the Java Health Center to find the location of large object allocations
- Use the Java Health Center to analyze calls to System.gc()
- Use Java tracing to find the location of calls to Sytem.gc()
- Use the Java Health Center to examine locking behavior

Introduction

This exercise examines the performance characteristics of the sample PlantsByWebSphere application. The sample application was modified to have some deliberate performance problems.

The sample application is started on server1 on hostB. The JMeter load generation tool is started on hostA. The load generation tool sends continuous requests to the application. Some requests that are sent trigger the code that was modified with deliberate coding mistakes.

The Java Health Center is used to locate the methods within the application that are taking the most time to run. After the location of a possible problem is identified, the code is examined to find any programming errors.

The Java Health Center is used to examine garbage collection behavior. Analysis points to a possible problem with large object allocations. The location in the code where large object allocations are occurring is identified. The code is examined to see whether the large object allocations can be avoided. Garbage collection analysis also points to the use of the System.gc() method call, which can cause performance problems. The

location of the calls to System.gc() is identified. The code is examined to determine whether the System.gc() call can be removed.

The Java Health Center is used to examine the locking behavior of the application. No locking problems are found. The information that is shown in the locking view is explained.

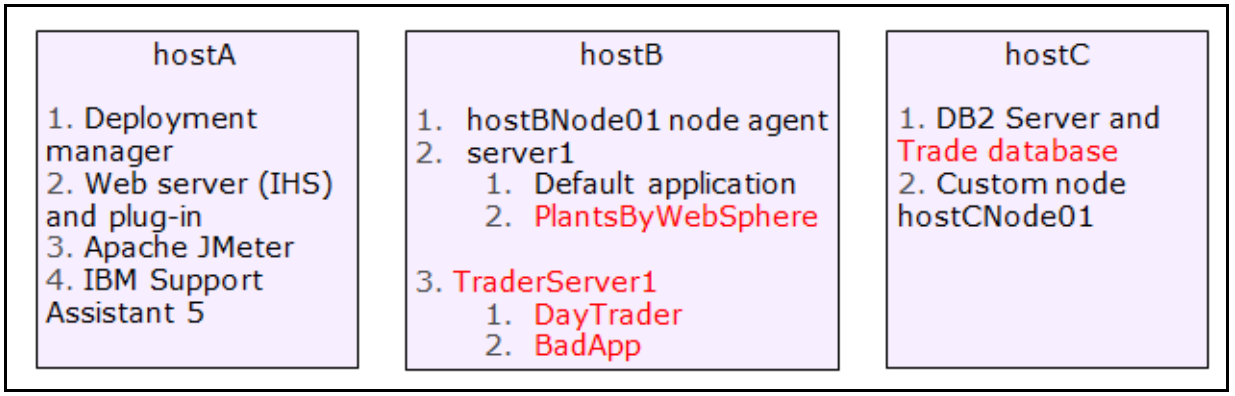
Requirements

You must have the modified PlantsByWebSphere sample application to install on server1. The JMeter load generation tool must be available on hostA. The IBM Support Assistant with the Java Health Center must be available on hostA. All of these requirements are available on the images that this course provides.

10.1. Configure the environment

In this part of the exercise, you configure the WebSphere environment to allow application monitoring by the Java Health Center. The application that is monitored is a modified version of the PlantsByWebSphere sample. The sample was modified to have deliberate performance problems.

After completing this exercise, the PlantsByWebSphere application is deployed to hostB along with the Derby database that it uses.



Note

<*profiles_root*> = /opt/IBM/WebSphere/AppServer/profiles

Section 1: Install the sample PlantsByWebSphere application

If the Deployment Manager, node agent, and server1 are already running, stop and restart them now.

__ 1. On **hostA**, start the Deployment Manager.

__ a. Open a terminal window and enter the command:
cd <*profiles_root*>/Dmgr/bin

__ b. Enter the command:
./startManager.sh

__ 2. On **hostB**, start the node agent.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
cd <*profiles_root*>/profile1/bin

__ c. Enter the command:
./startNode.sh

__ 3. Create the Derby database for the PlantsByWebSphere application.

__ a. Open an ssh terminal window for **hostB**.

__ b. Enter the command:
cd /opt/IBM/WebSphere/AppServer/derby/databases

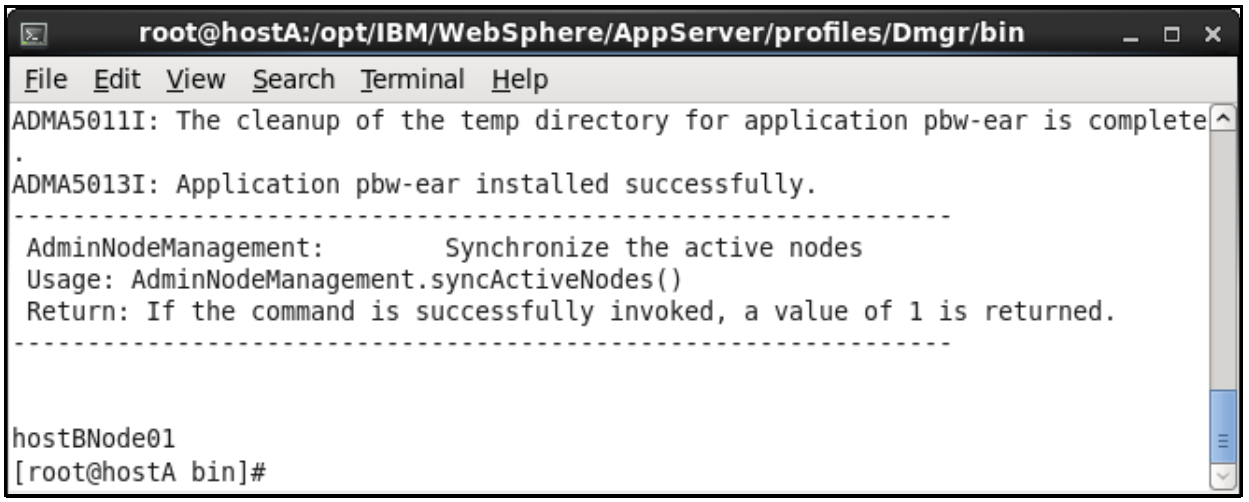
__ c. Run the following command:
jar xf ../../samples/PlantsByWebSphere/Derby/PLANTSDB/pbw-db.jar

__ 4. Use a script to install the PlantsByWebSphere application.

__ a. On **hostA**, open a terminal window and enter the command:
cd <profiles_root>/Dmgr/bin

__ b. Run the following command:
./wsadmin.sh -f /usr/labfiles/AppProfiling/install_pbw.py -user wasadmin -password web1sphere

__ 5. When the installation process completes, verify that you see the following message.



Section 2: Enable the Health Center agent

The Java Health Center communicates with an agent in the application server. Specify the generic JVM argument that allows communication between the application server and the Java Health Center.

__ 1. Log on to the administrative console

__ a. Browse to `http://hostA:9060/ibm/console`

__ b. Log in with the username **wasadmin** and password **web1sphere**

__ 2. Add a generic JVM argument to server1 on hostB.

__ a. Click **Servers > Server Types > WebSphere application servers**.

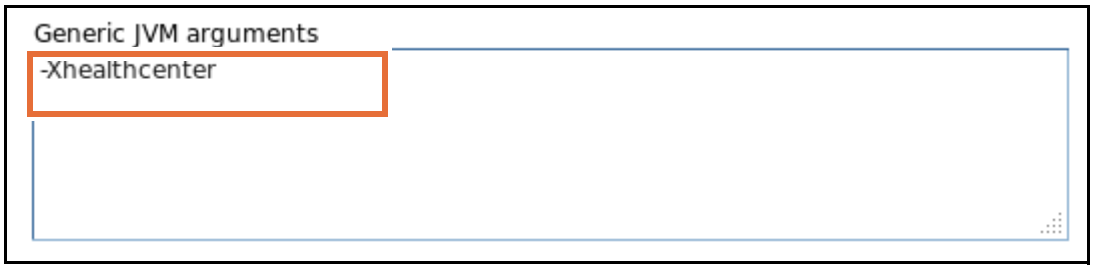
__ b. Click **server1**.

__ c. On the right side, under **Server Infrastructur**e, expand **Java and Process Management**.

__ d. Click **Process definition**.

__ e. On the right side, under **Additional properties**, click **Java Virtual Machine**.

__ f. Add `-Xhealthcenter` to the **Generic JVM arguments** field.



__ g. Click **OK**.

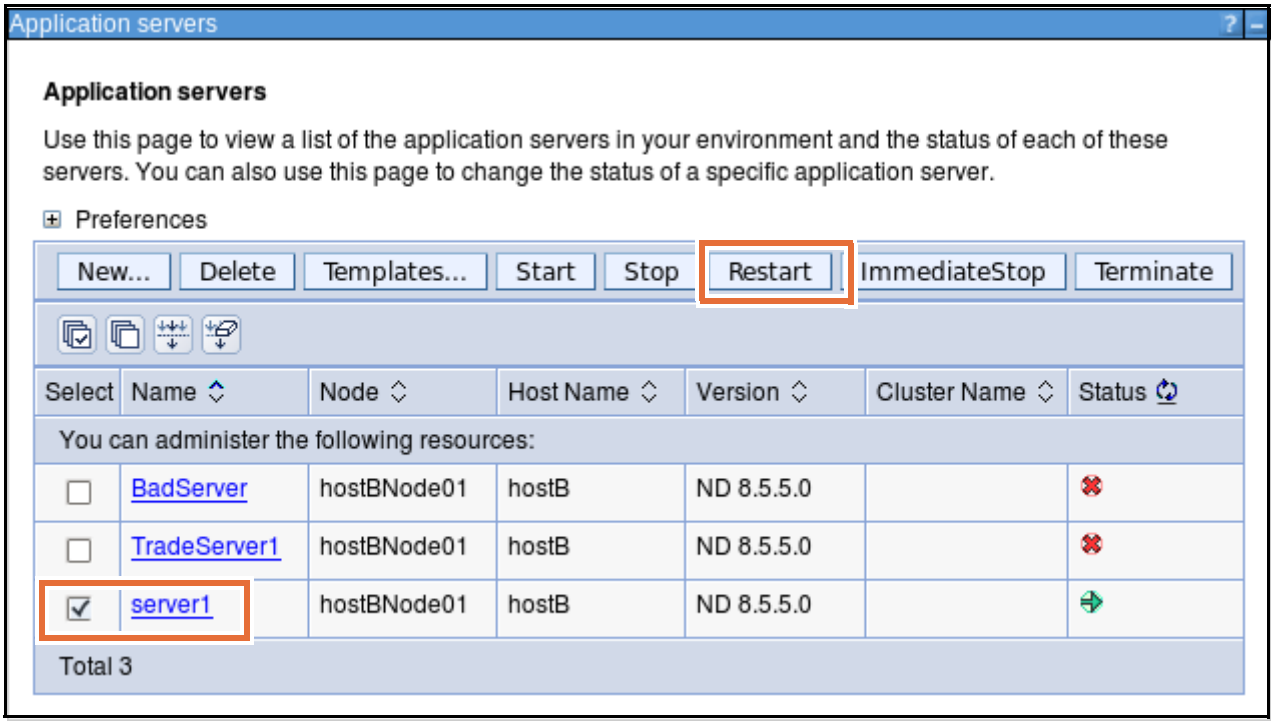
__ h. **Save** to the master configuration.

__ 3. Restart server1 on hostB.

__ a. Click **Application servers** in the breadcrumb trail.

__ b. Select **server1**.

__ c. Click **Restart**.



__ d. Wait for the server to restart.

__ 4. Stop any other application servers that are still running from previous exercises such as **TradeServer1** and **BadServer**.

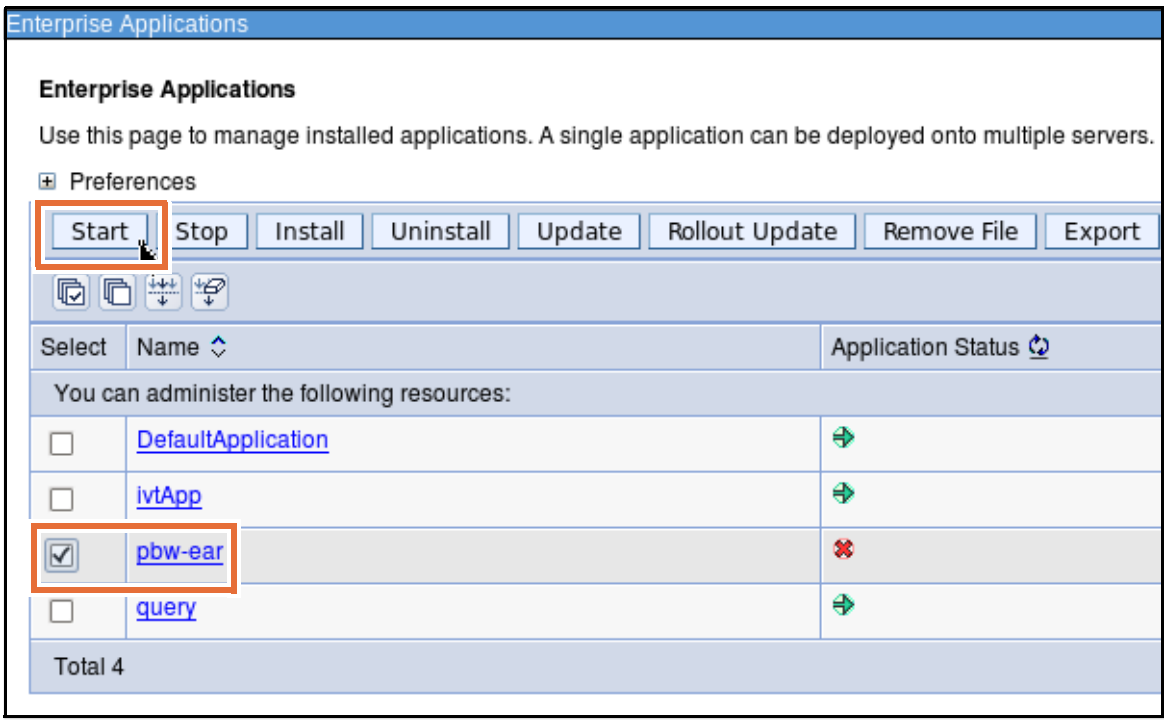
Section 3: Start the PlantsByWebSphere application

Characteristics of the modified PlantsByWebSphere application are examined. Performance problems in the application are found.

__ 1. Start the PlantsByWebSphere sample application

__ a. Click **Applications > Application Types > WebSphere enterprise applications**.

__ b. If the pbw-ear application is not started, select **pbw-ear** and click **Start**.



__ c. Wait for the application to start.

Section 4: Start the Java Health Center

The Java Health Center monitors application characteristics.

__ 1. Start the IBM Support Assistant if it is not already started.

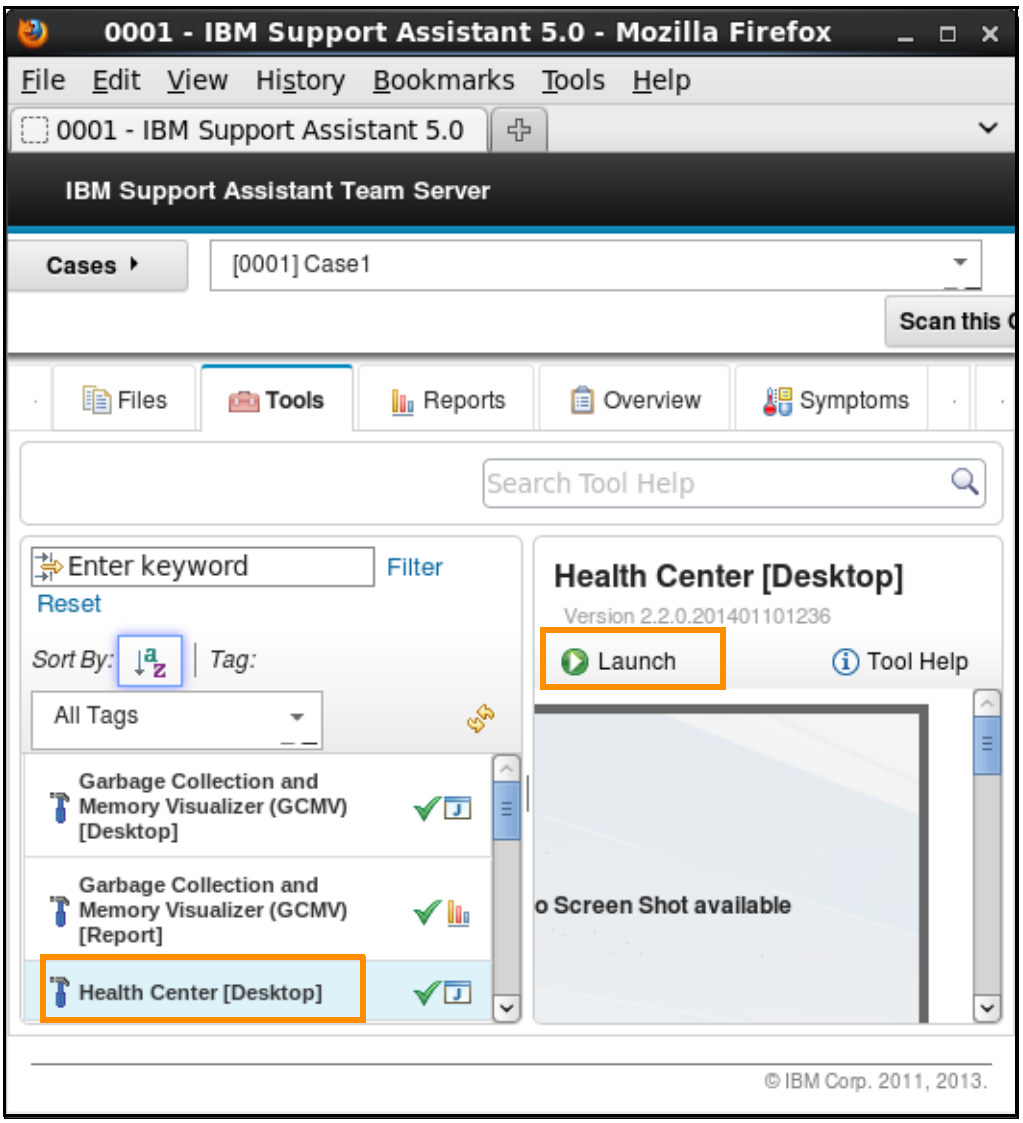
__ a. In a terminal window on **hostA**, enter **`/opt/ibm/ISA5/start_isa.sh`**

__ 2. Use the browser to open the IBM Support Assistant at `https://hosta:10943/isa5`

__ 3. Start the Health Center tool.

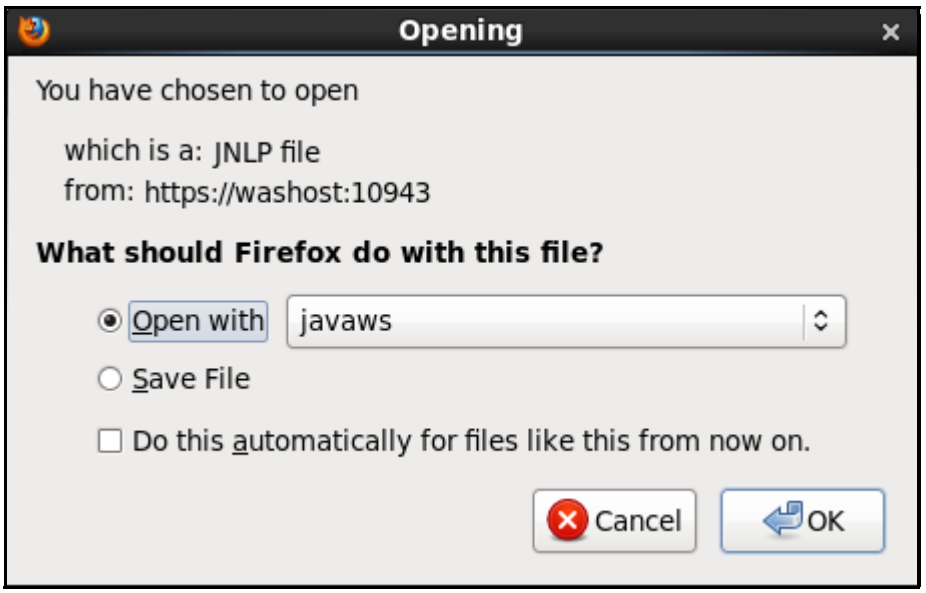
__ a. In the IBM Support Assistant, click the **Tools** tab and select **Health Center[Desktop]**.

__ b. On the right pane, click **Launch**.



__ c. In the Run Tool window, click **Submit**.

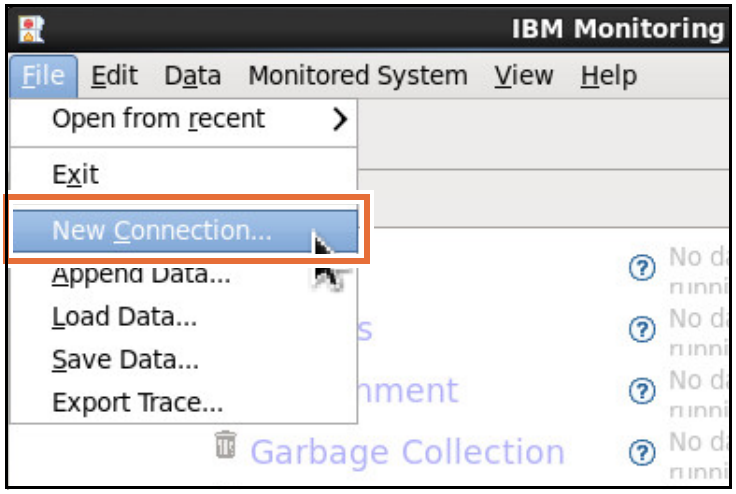
__ d. In the Opening window, select **Open with javaws**.



__ e. Click the menu for the “Open with” option, browse to `/opt/IBM/WebSphere/AppServer/java_1.7_64/jre/bin` and select **javaws**.

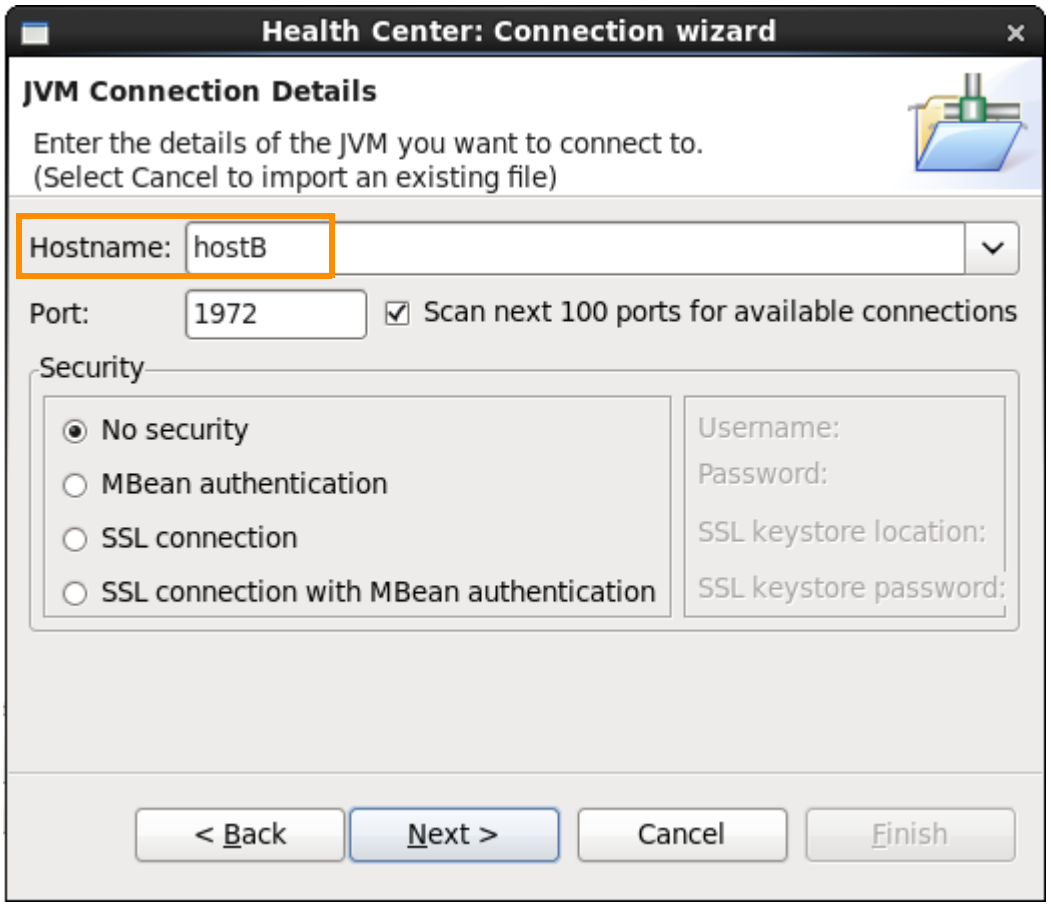
__ f. Click **OK**.

__ g. When the Health Center opens, click **File > New Connection**.



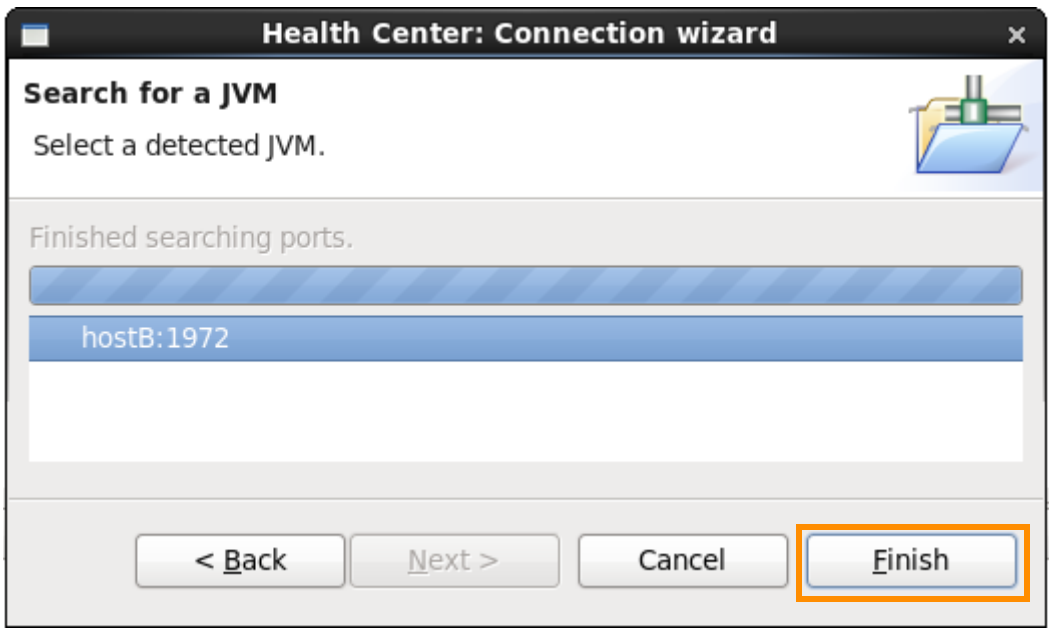
__ h. Click **Next** on the Health Center connection wizard.

__ i. Enter **hostB** for the hostname and use the default port.

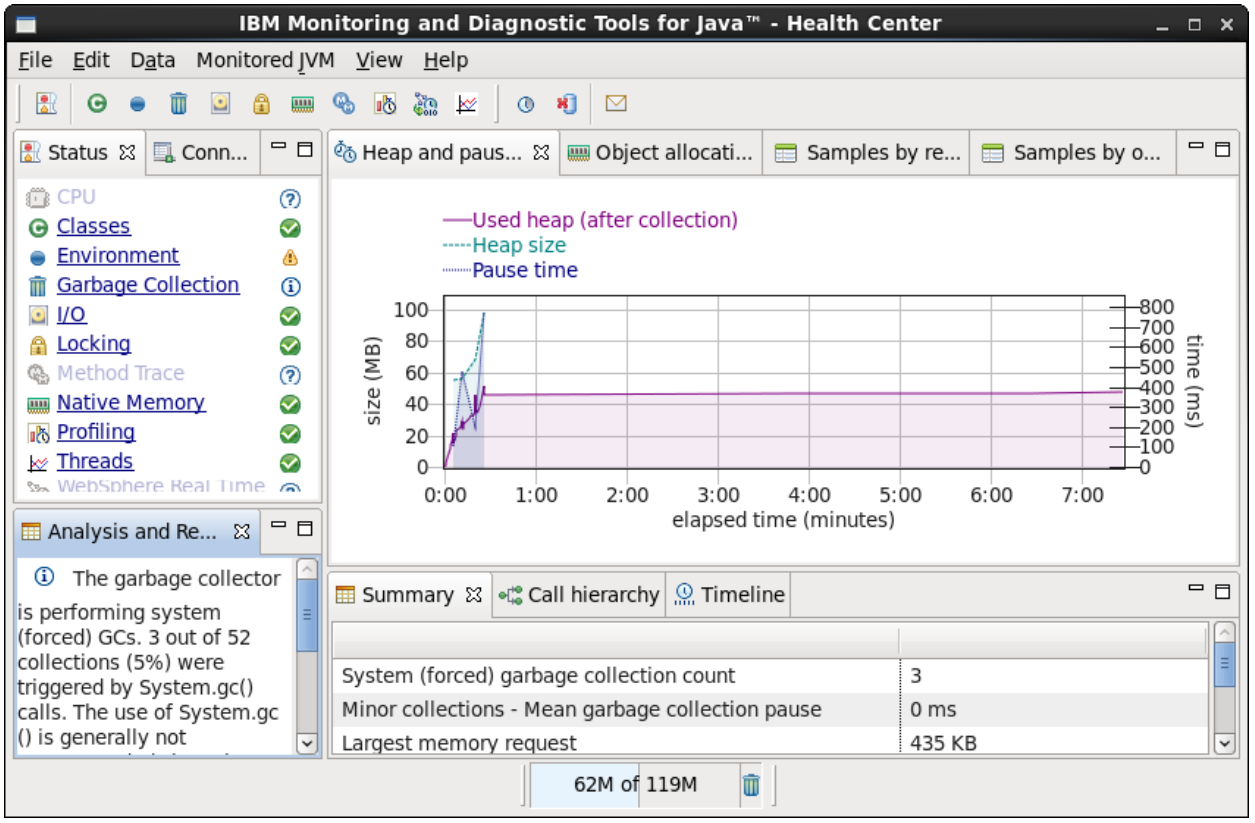


__ j. Click **Next**.

__ k. When the wizard detects the server that listens on port 1972, click **Finish**.



__ l. Wait a few minutes while the Health Center retrieves runtime data from the server. Eventually the Health Center GUI is populated with server data.



10.2. Use the Health Center to analyze the application

The IBM Java Health Center is a tool that runs in the IBM JVM. The Health Center has a low impact on server performance. Health Center provides information on method profiling, garbage collection, lock analysis, threads, I/O, native memory, and more. It is good for analyzing performance issues.

Section 1: Examine the application

The PlantsByWebSphere application provides an implementation of a shopping website. Functions for browsing and buying from a catalog of gardening supplies are available. The sample was modified to have deliberate performance problems. In this section of the exercise, examine the application behavior, but do not click any products in the Accessories tab.

__ 1. Open the PlantsByWebSphere site.

__ a. In a web browser, go to the URL **`http://hostB:9080/PlantsByWebSphere`**

__ 2. Navigate to the different parts of the application. Do not click any products in the **Accessories** tab.

Section 2: Check the application for slow running methods

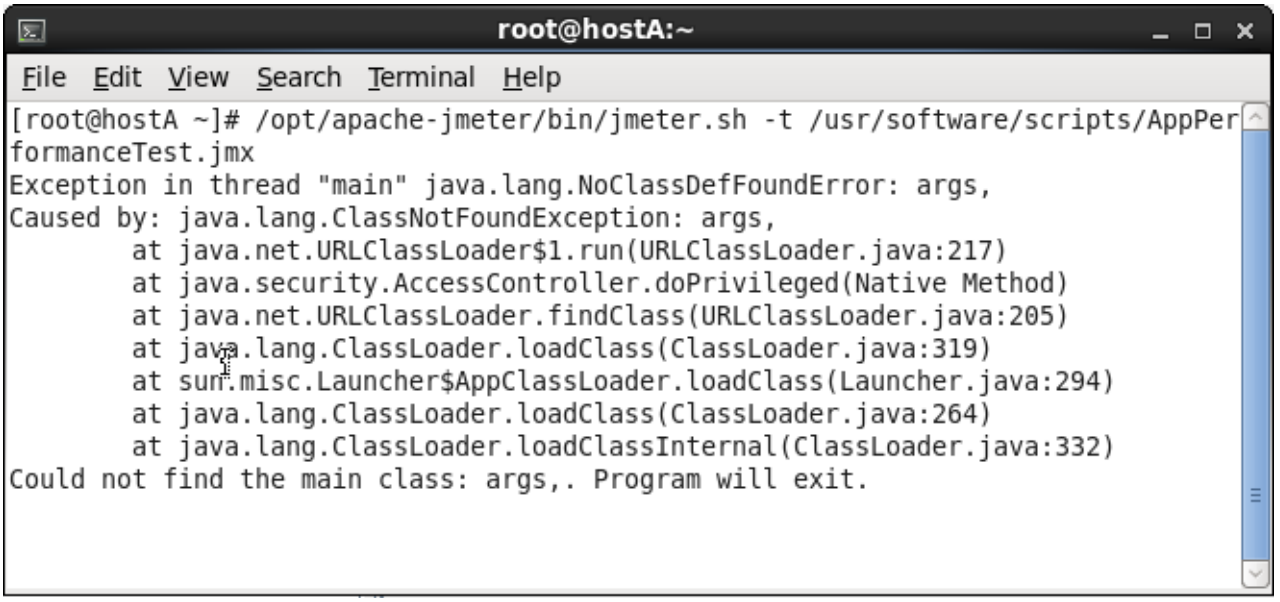
Start by analyzing where the WebSphere JVM is spending most of its time to see whether any optimizations can be made. The Apache JMeter load generation tool is used to send application requests to the PlantsByWebSphere application. Some requests that are sent trigger the code that was modified with deliberate coding mistakes.

__ 1. Start Apache JMeter.

__ a. Open a terminal window on **hostA**.

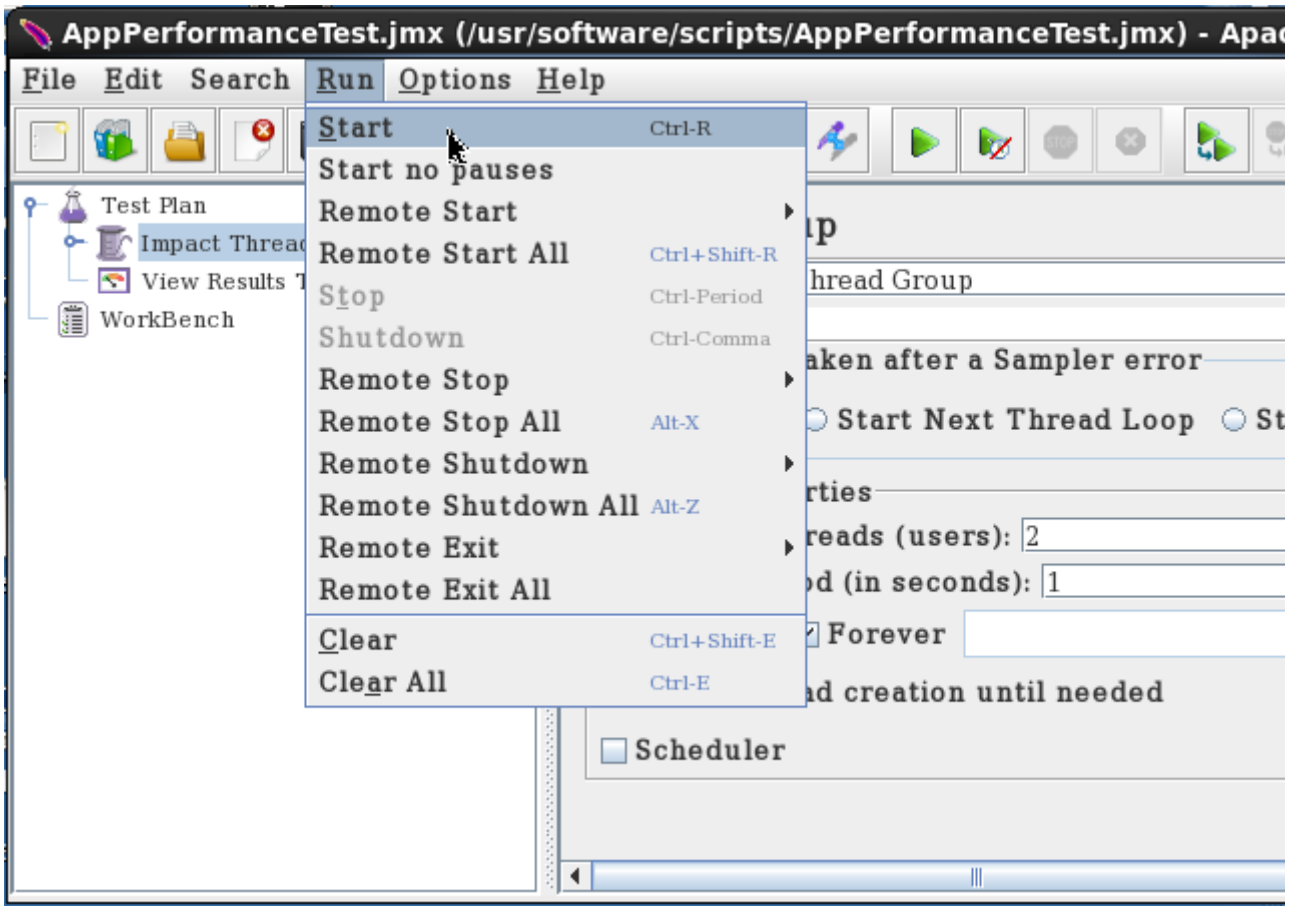
__ b. Enter the following command:

```
/opt/apache-jmeter/bin/jmeter.sh -t
/usr/labfiles/AppProfiling/AppPerformanceTest.jmx -H hostB -P 9080
```

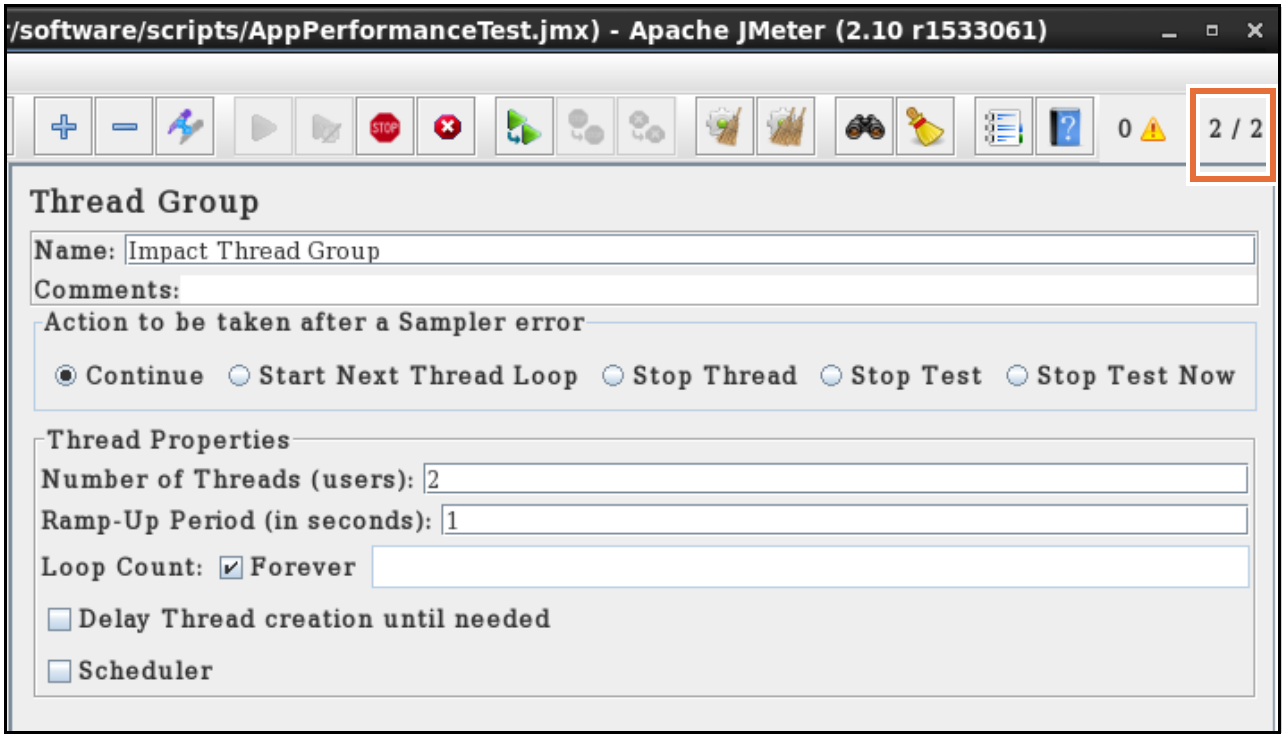


__ c. If you see the **`ClassNotFoundException`** exception that is shown in the screen capture, you can ignore it.

__ d. In the JMeter window, click **Run > Start**.



__ e. Wait a few moments until the number of started threads reaches **2**, as indicated in the upper-right corner of the JMeter window.



__ 2. Switch to the **Health Center** window, and click the **Status** tab.

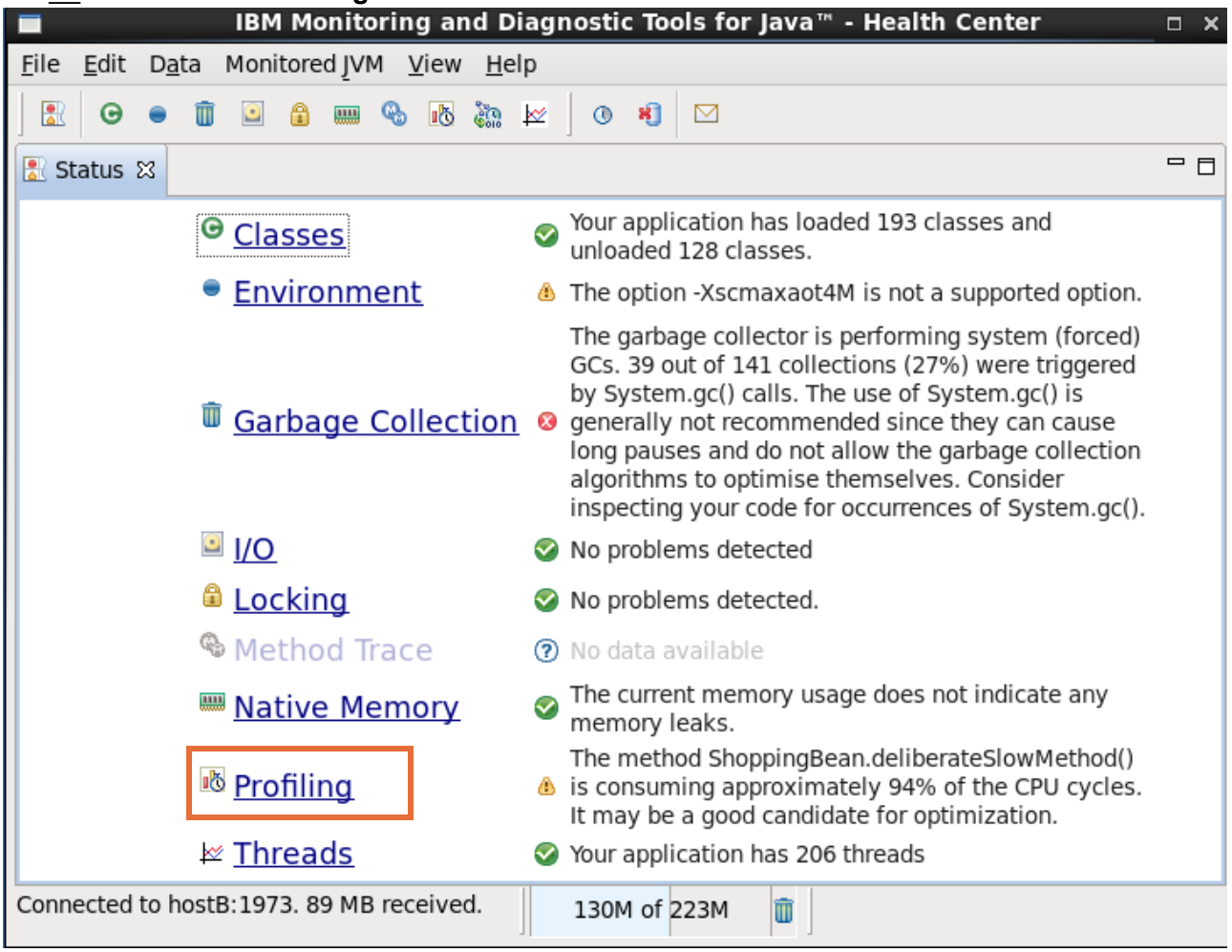
Information

Health Center status view

The Health Center status pane summarizes the main categories of data that Health Center is monitoring, and also summarizes current recommendations. The data categories to be collected can also be customized from the Monitored JVM menu.

__ 3. Start by analyzing where the WebSphere JVM is spending most of its time to see whether any optimizations can be made

__ a. Click the **Profiling** link.

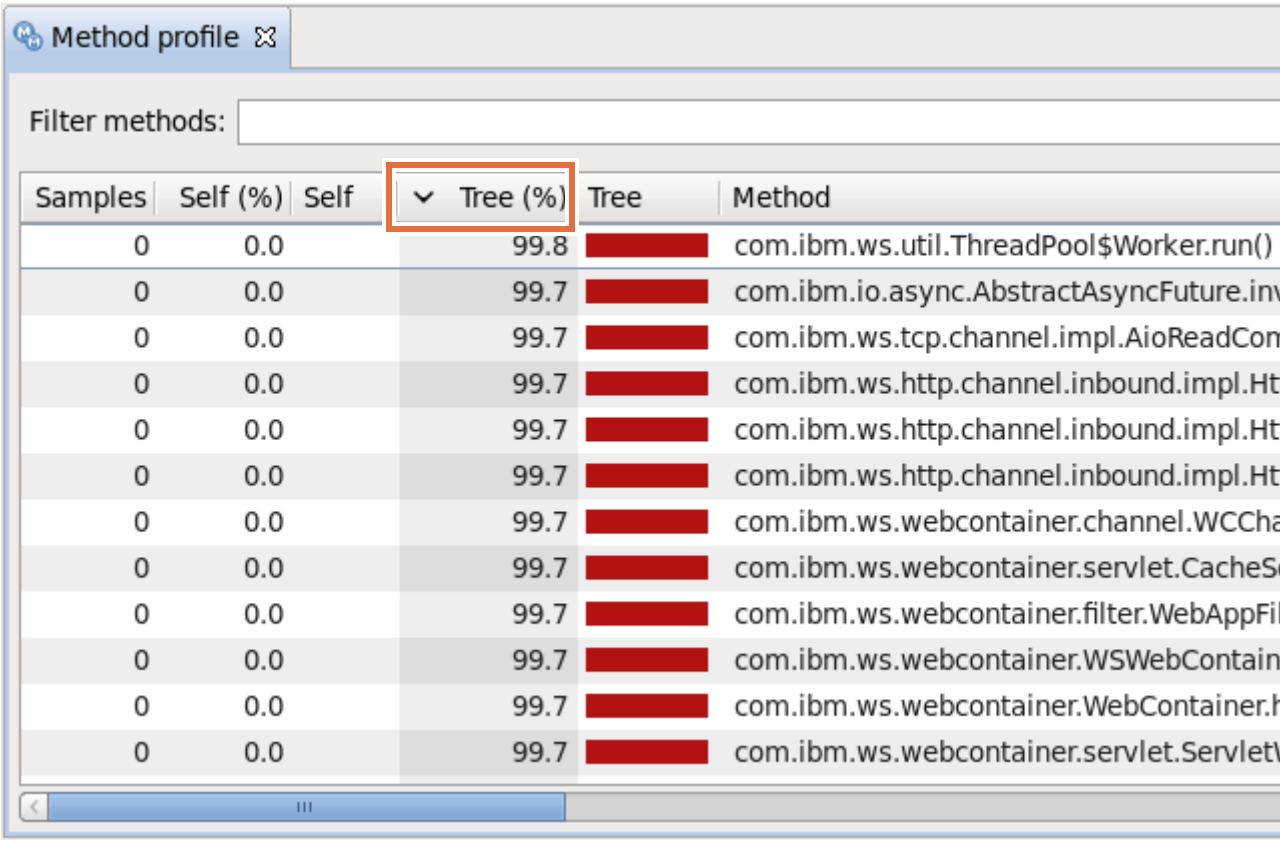


Information

Health Center profiling

Health Center uses a “sampling based” method profiler. This behavior means that it takes a periodic sample of the methods that are running and reports those methods that are using the most time in the JVM.

__ b. Sort the table of data by “Tree %” by clicking the “**Tree %**” column heading.



Information

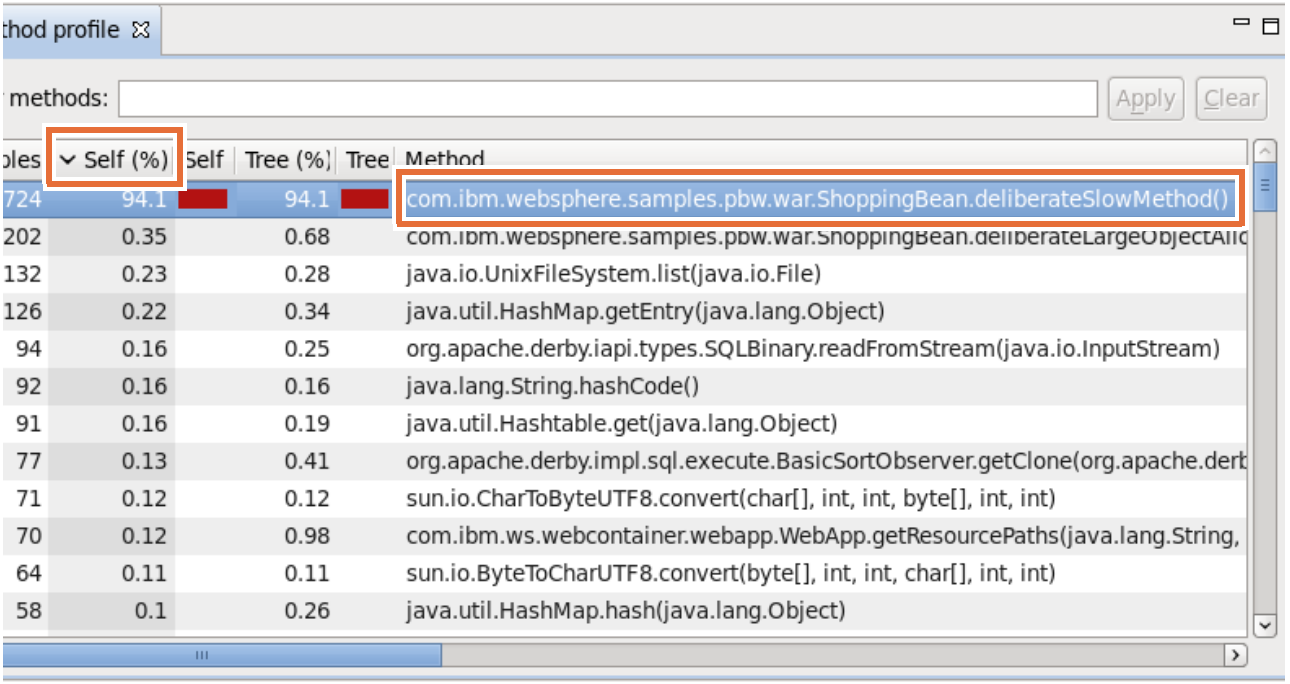
Interpreting method data

Within the Health Center, collections of methods are organized into structures that are called trees. You should see that in this case, a “**ThreadPool$Worker.run()**” method represents the top of a tree that is using a high percentage of the JVM’s time.

However, also note the value in the “Self (%)” column, which indicates that the method “ThreadPool$Worker.run()” is using a **low percentage** of the JVM’s time. Therefore, the problem must be in some code that the “ThreadPool$Worker.run()” method calls, that is, further down the tree / method call stack.

Because WebSphere handles the incoming HTTP requests with the “ThreadPool$Worker” class, this information gives a clue that something is wrong in a running web application.

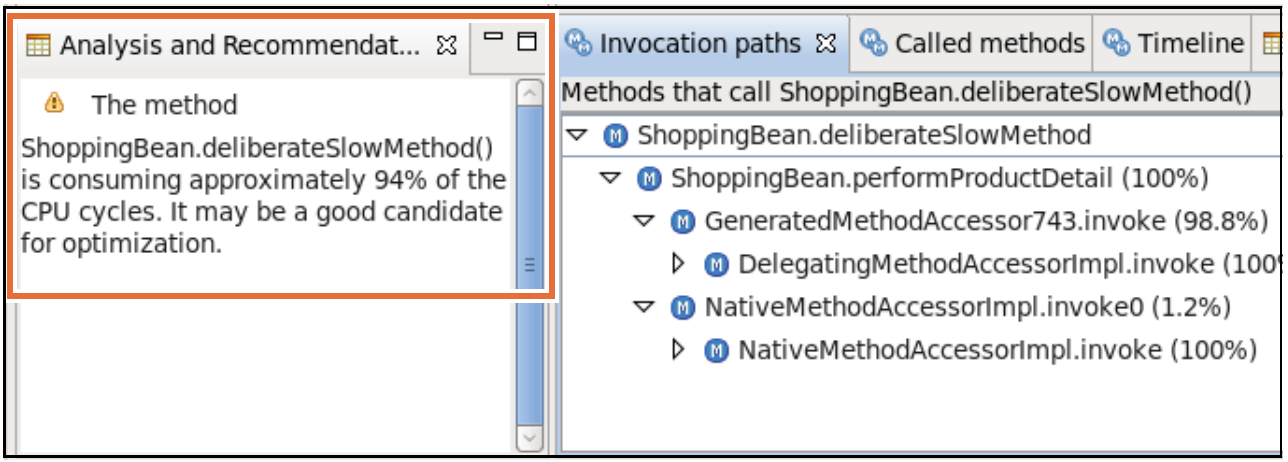
__ c. Reorder the table to see the results for individual methods by clicking **Self**.



Note

Now you can see the individual method "**deliberateSlowMethod**" in the **ShoppingBean** class is using a high percentage of the JVM's time. The "Self" and "Tree" columns (without the % symbol) are a graphical indication of the same data.

__ d. Select the expensive method in the table by clicking it once.



Information

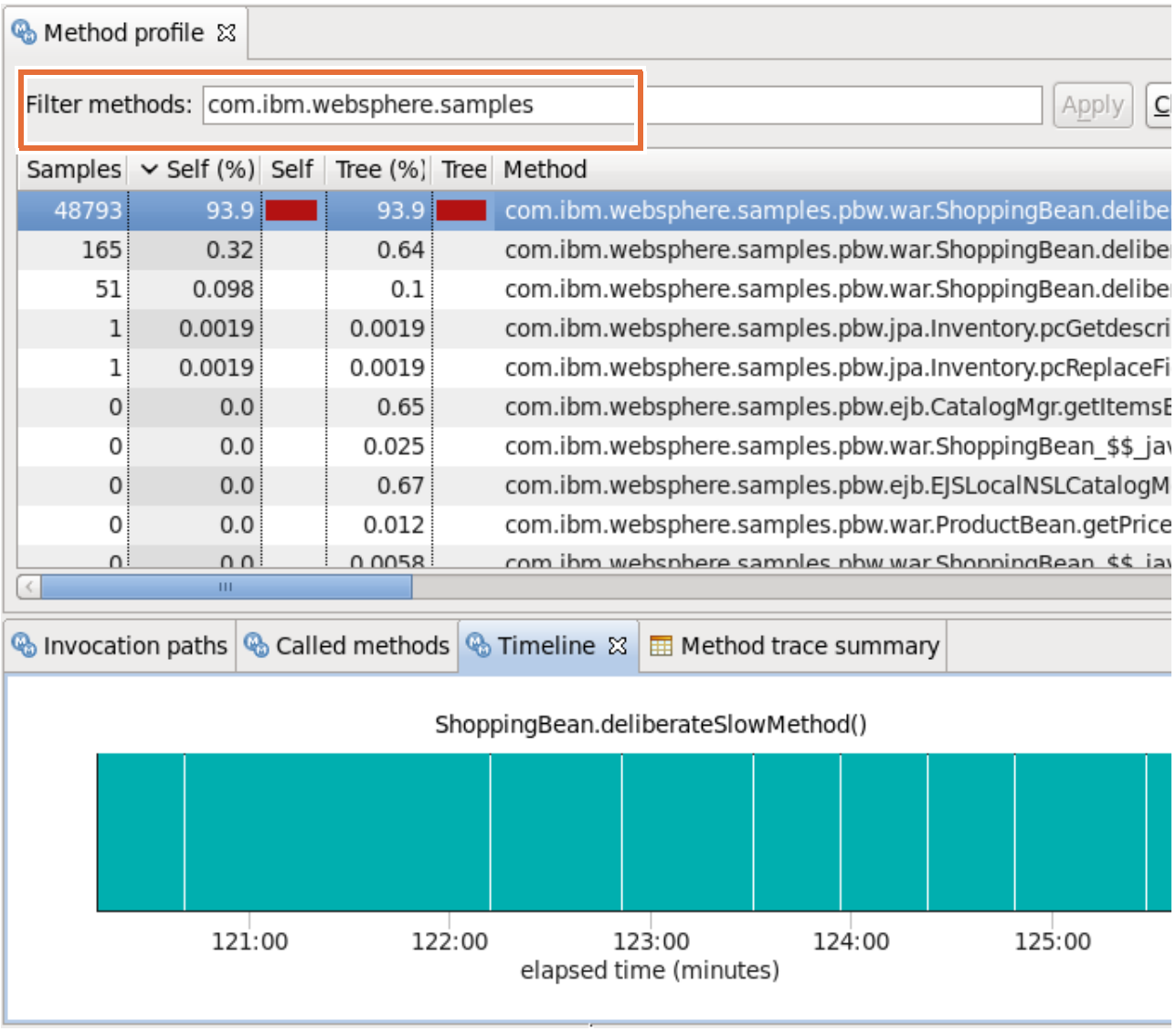
Invocation paths and timeline

The “Invocation paths” tab shows what is calling the “deliberateSlowMethod”. The “timeline” tab shows when the “deliberateSlowMethod” was invoked.

Also, notice that Health Center automatically identifies the time consuming method and highlights this fact in the “Analysis and Recommendations” section.

__ e. The Plants sample is clearly suffering with at least one slow method, hence, type **`com.ibm.websphere.samples`** in the **Filter Methods** field and click **Apply**.

__ f. Notice that only the “**ShoppingBean.deliberateSlowMethod**” in the Plants sample has a high value for the “Self (%)”.



__ g. Click **Clear** on the Method profile tab.

__ 4. Examine the application code for the slow method. The source code for the ShoppingBean is provided. Normally, you would use your development tool to examine the code. In this environment, you use a simple editor.

__ a. In a terminal window on **hostA**, enter the following command:

```
gedit /usr/labfiles/AppProfiling/ShoppingBean.java
```

__ b. **Maximize** the window.

__ c. Press Ctrl + F, and in the **Search for:** field, enter **deliberateslowmethod**, and click **Find**.

__ d. Notice that the first instance found is the invocation of the method when the user selects Tulips.

__ e. Click **Find** again; then click **Close** on the Find pop-up menu. You found the implementation of the method.

__ f. Examine the method code to find what is causing the long execution time. The “deliberateSlowMethod” runs a tight loop, which does not end until a 10-second wait time.

```
private void deliberateSlowMethod() {

        // ------------------------------------------------------
        // User clicked on the Tulips, let's tip toe through a
        // slow method
        // ------------------------------------------------------

        System.out.println("==> STARTING SLOW METHOD");

        long timestamp = System.currentTimeMillis();
        long target = timestamp + 10000;

        System.out.println("timestamp="+timestamp);
        System.out.println("resume at="+target);
        while(timestamp < target) {
                timestamp = System.currentTimeMillis();
        }

        System.out.println("==> ENDING SLOW METHOD");
}
```

__ g. Close the editor window.

Note

In this case, finding the problem with the code was trivial. In a real case, the analysis of the code might be more difficult. The value of the Health Center is in narrowing down exactly what method should be analyzed.

Section 3: Check the application for memory issues

The Garbage collection perspective of the Health Center monitors performance of garbage collection and memory usage. The Analysis and Recommendations view might contain general recommendations for tuning the size of the heap. The view might also contain application-specific analysis and recommendations.

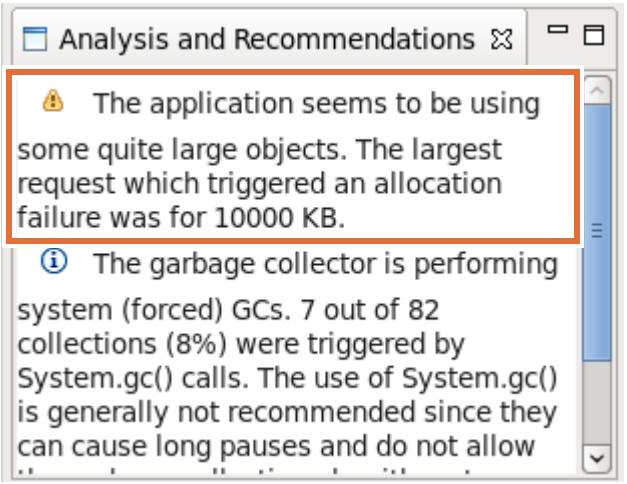
Examine memory usage

By using the Health Center, you can view the size, time, and code location of a large object allocation request that meets specific threshold criteria.

__ 1. Click the **Garbage Collection** link.

__ 2. Observe the Analysis and Recommendations window.

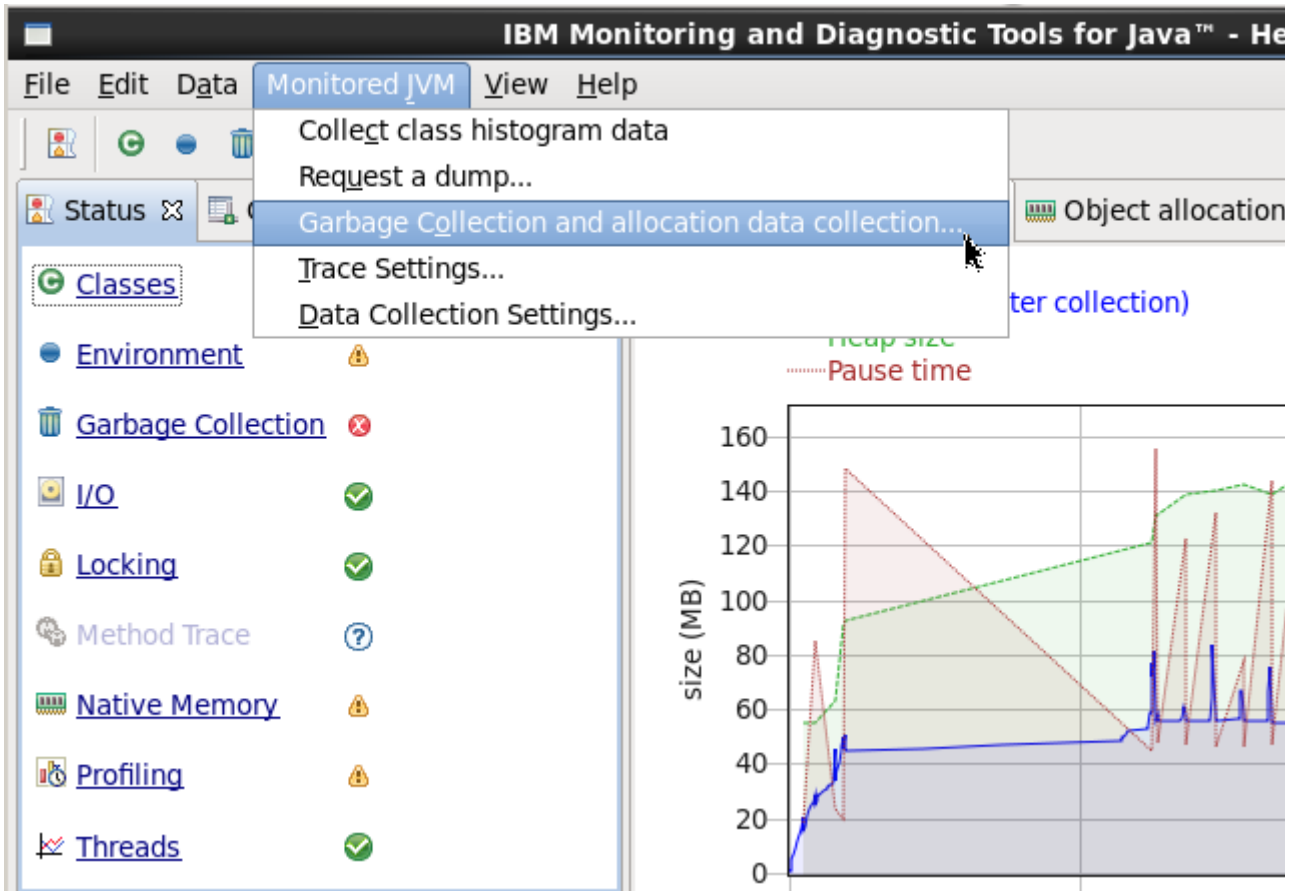
__ a. Notice that the analysis warns of large object allocations. You might have to scroll down to see the message.



Large object allocations are likely to trigger frequent garbage collections and might indicate that the application code can be optimized.

__ 3. Investigate further.

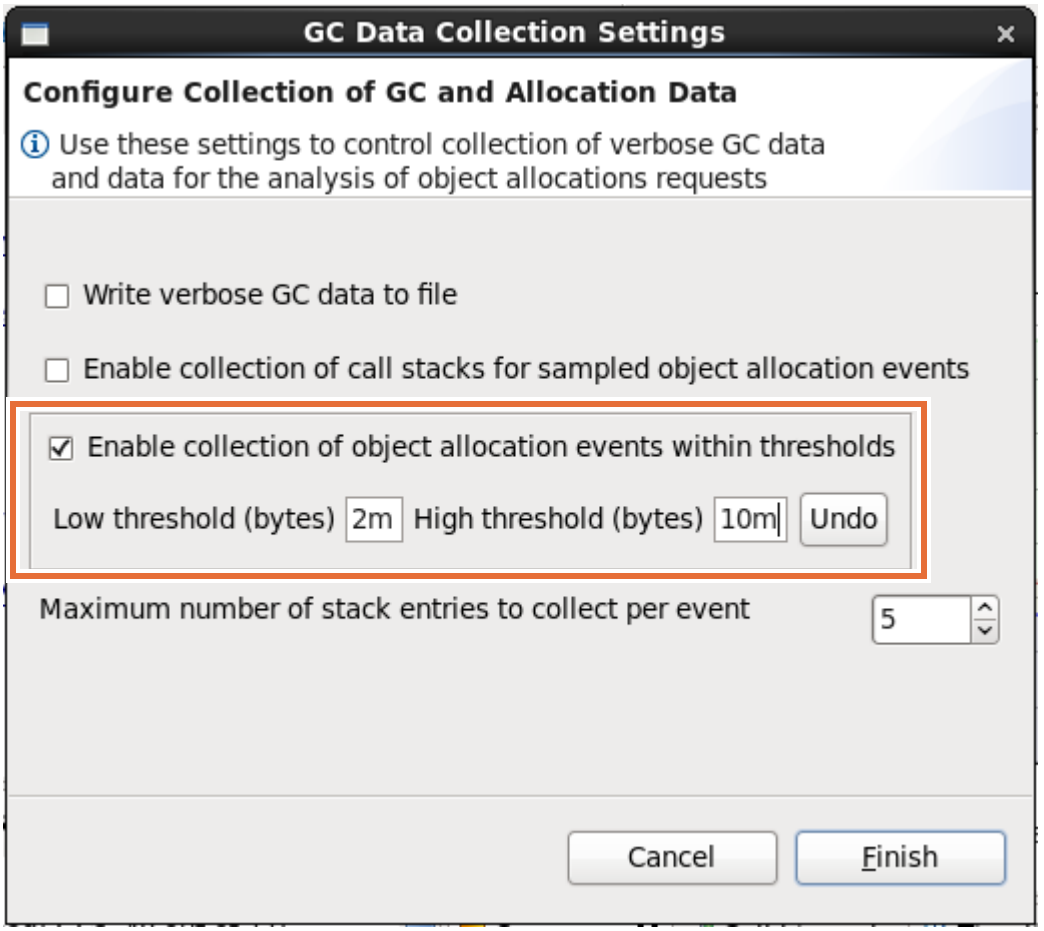
__ a. Click **Monitored JVM > Garbage Collection and allocation data collection...**



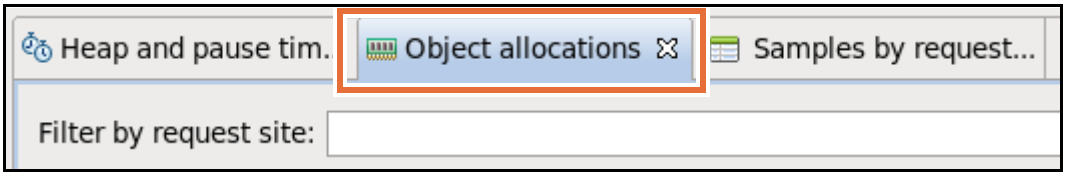
__ b. Select **Enable collection of object allocation events within thresholds**.

__ c. For the **Low threshold**, enter **2 m**. For the **High threshold**, enter **10 m**. The thresholds that are entered cause the collector to gather data on the largest objects.

__ d. Click **Finish**.

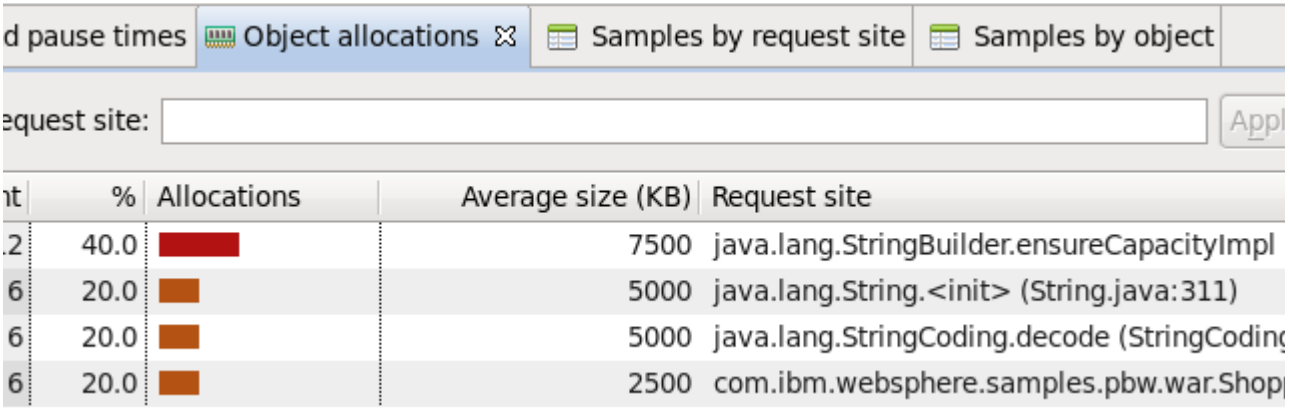


__ e. Click the **Object allocations** tab.



__ f. Wait a few moments until the Health Center client parses the large object allocation data. This process might take up to 1 minute.

__ g. Notice that some object allocations met the defined size threshold. (The large object allocations are caused by creating a large String).

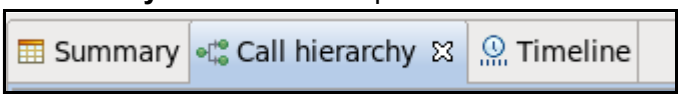


__ 4. Determine what code is causing the large allocations.

__ a. Click the **Average size (KB)** column header to sort by allocation size.

__ b. Select the top element in the list. This method should the **java.lang.StringBuilder.ensureCapacityImpl** method. If that method is not in the list yet, wait a few more moments for the Health Center to gather more allocation data, then select the **java.lang.StringBuilder.ensureCapacityImpl** method.

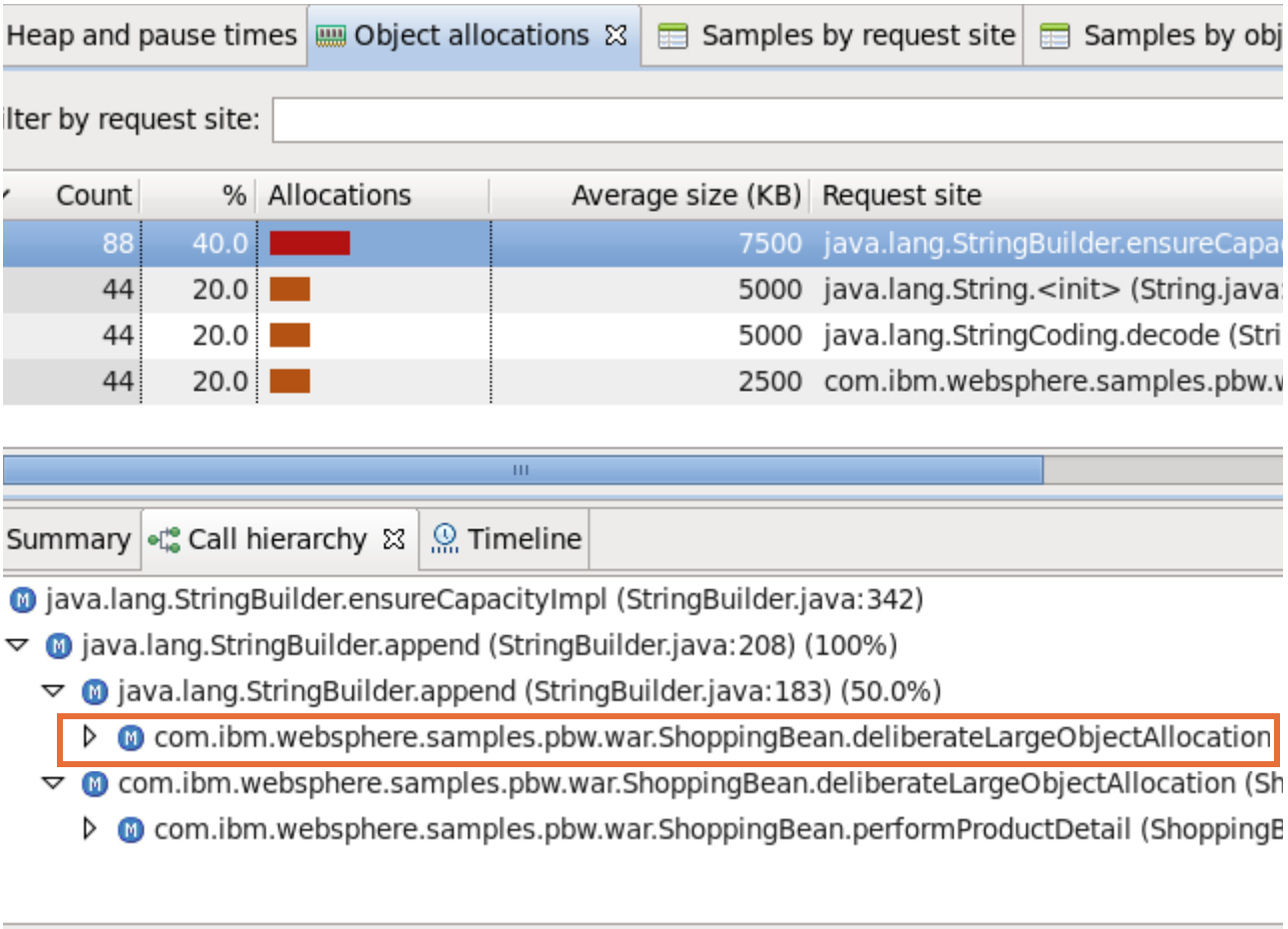
__ c. Click the **Call hierarchy** tab in the lower pane.



__ d. Examine the call hierarchy to find the application method that makes the allocation call.

__ e. Notice that the application method that is making the allocation call is the **ShoppingBean deliberateLargeObjectAllocation** method.

Examining the call hierarchy of a large object allocation points to a location in the application code.



__ 5. Examine the application code to determine whether the large object allocation is necessary.

__ a. In a terminal window on **hostA**, enter the command:

```
gedit /usr/labfiles/AppProfiling/ShoppingBean.java
```

__ b. **Maximize** the window.

__ c. Press Ctrl + F, and in the **Search for:** field, enter **`deliberatelargeobjectallocation`**, and click **Find**.

__ d. Notice that the first instance found is the invocation of the method when the user selects Grapes.

__ e. Click **Find** again, then click **Close**. You found the implementation of the method.

__ f. Examine the method code to find where the large object is allocated. The “deliberateLargeObjectAllocation” method allocates a large local byte array.

```
private void deliberateLargeObjectAllocation() {

        // ------------------------------------------------------------------------
        // User clicked on the Grapes, let's make them whine with a
        // large object allocation
        // ------------------------------------------------------------------------


        System.out.println("==> STARTING LARGE OBJECT ALLOCATION");

        // Handle to a large object.  Not a memory leak, just a LOA that will be GC'd
        HashSet<String> largeObject = null;

        largeObject = new HashSet<String>();
        long timestamp = System.currentTimeMillis();
        byte[] array = new byte[2560000];
        Arrays.fill(array, (byte) 88);
        largeObject.add(new String(array) + (timestamp));


        System.out.println("==> ENDING LARGE OBJECT ALLOCATION");
}
```

__ g. Close the editor window.

Note

In this case, determining that the large object allocation is not required for correct functioning of the application is trivial. In the real case, it would be necessary to examine the design decision that required the allocation. If the allocation is causing performance problems, it might be necessary to come up with a different design. Again, the value of the Health Center is in finding the code that needs to be examined.

Examine the use of System.GC()

The Java code “System.gc()” forces a full garbage collection cycle. This practice is generally not recommended as the garbage collector should manage its own schedule of garbage collection, and does not always need to run the compaction phase of GC, which is the most processor intensive. An application that calls System.gc() always triggers the most expensive compaction phase.

Health Center can be used to filter method innovations (for example, looking for System.gc). However, the method profiling works on a sampling basis, so infrequent method calls might not appear in Health Center's profiling results. Therefore, the best way to find the method that is calling System.gc() is to set a JVM trace, and use Health Center to extract this information from the JVM.

__ 1. Stop running the load on the application.

__ a. In the JMeter window, click **Run > Stop**.

__ 2. Add a trace specification for generating call stacks when System.gc() is called.

__ a. In the administrative console, navigate to **Servers > Server Types > WebSphere application servers**.

__ b. Click **server1**.

__ c. On the right side, under **Server Infrastructure**, expand **Java and Process Management**.

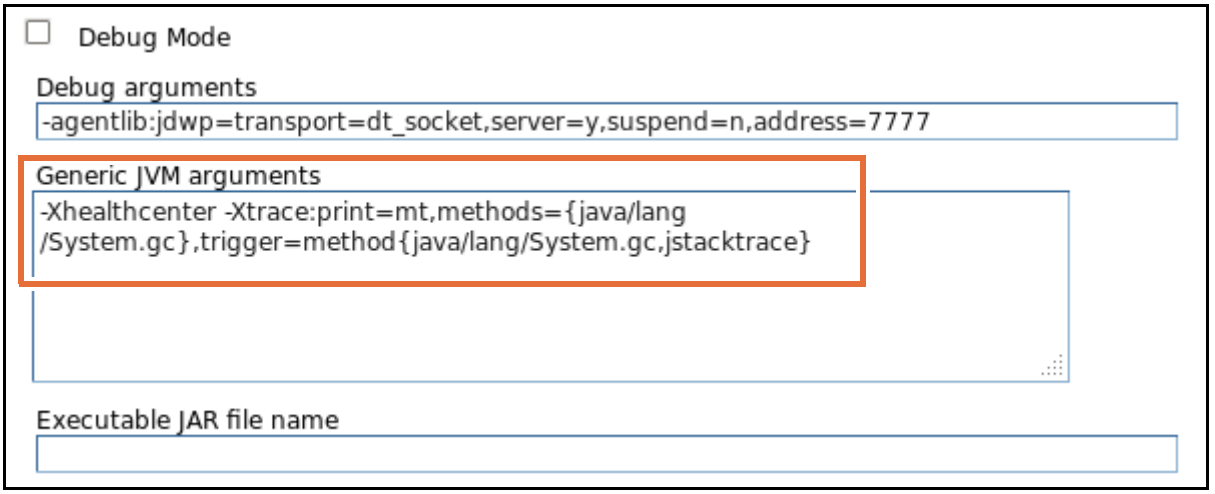
__ d. Click **Process definition**.

__ e. On the right side, under **Additional properties**, click **Java Virtual Machine**.

__ f. Add the following line to the **Generic JVM arguments** field. You can copy this string from the file **`/usr/labfiles/AppProfiling/Systemgc-trace.txt`**.

`-Xtrace:print=mt,methods={java/lang/System.gc},trigger=method{java/lang/System.gc,jstacktrace}`

This argument causes the JVM to print a stack trace each time the **`java/lang/System.gc`** is called. Normally, this output goes to the `native_stderr.log`. However, if the Health Center agent is also configured on the JVM (with `-Xhealthcenter`, as in this lab exercise), then the output is instead directed to the Health Center client.



__ g. Click **OK**.

__ h. **Save** to the master configuration.

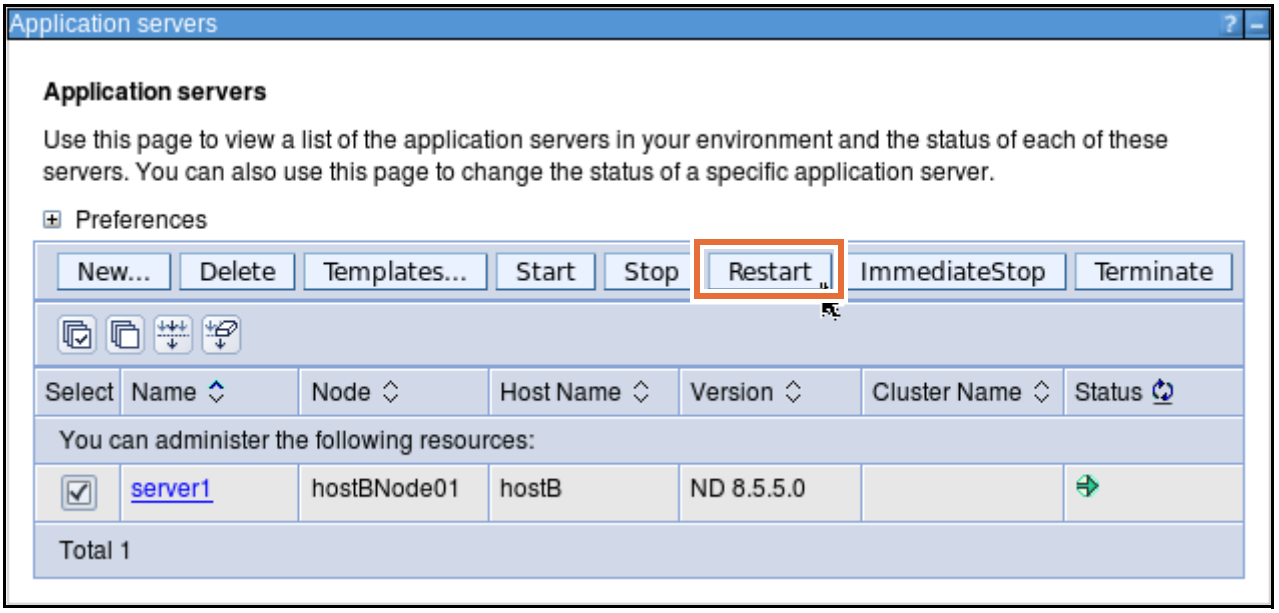
__ 3. Restart server1 on hostB.

__ a. Click **Application servers** in the breadcrumb trail.

__ b. Select **server1**.

__ c. Click **Restart**.

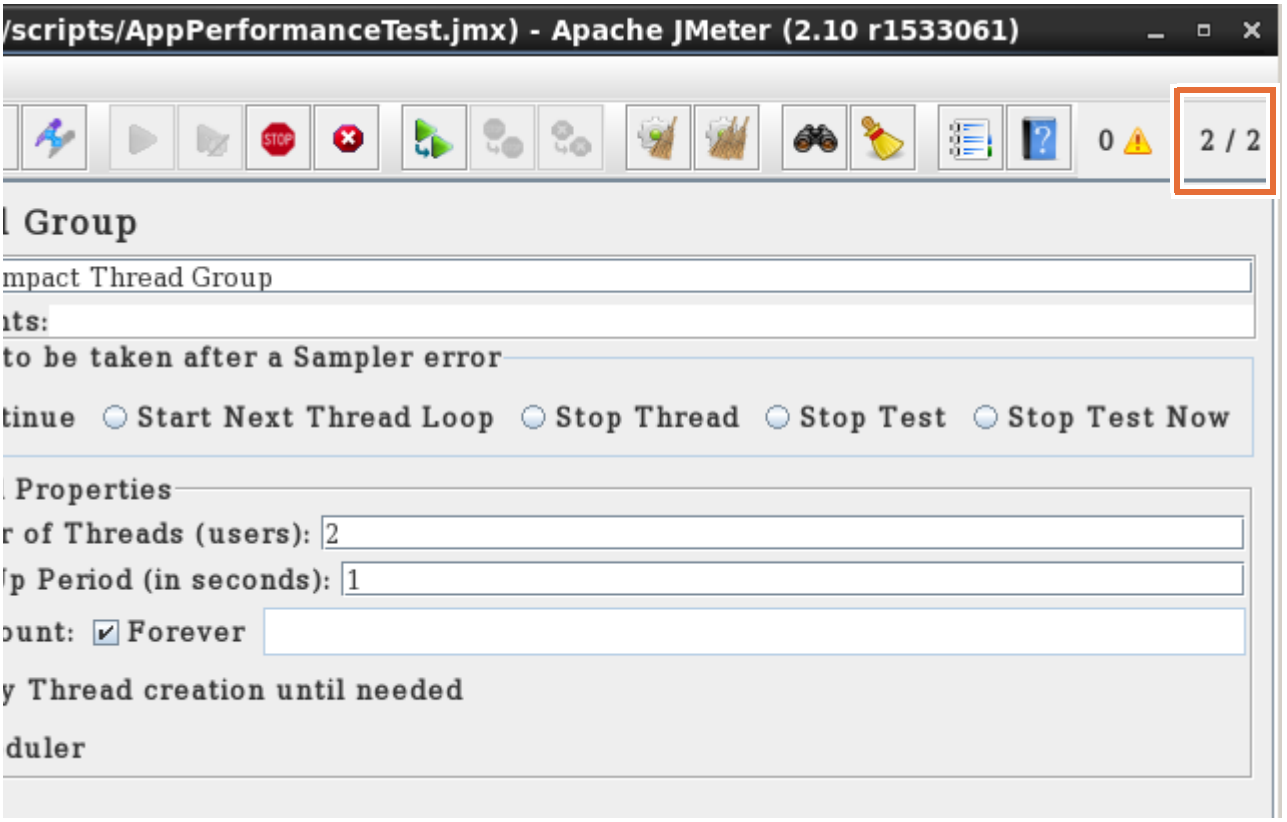
__ d. Wait for the server to restart.



__ 4. Start the load on the application.

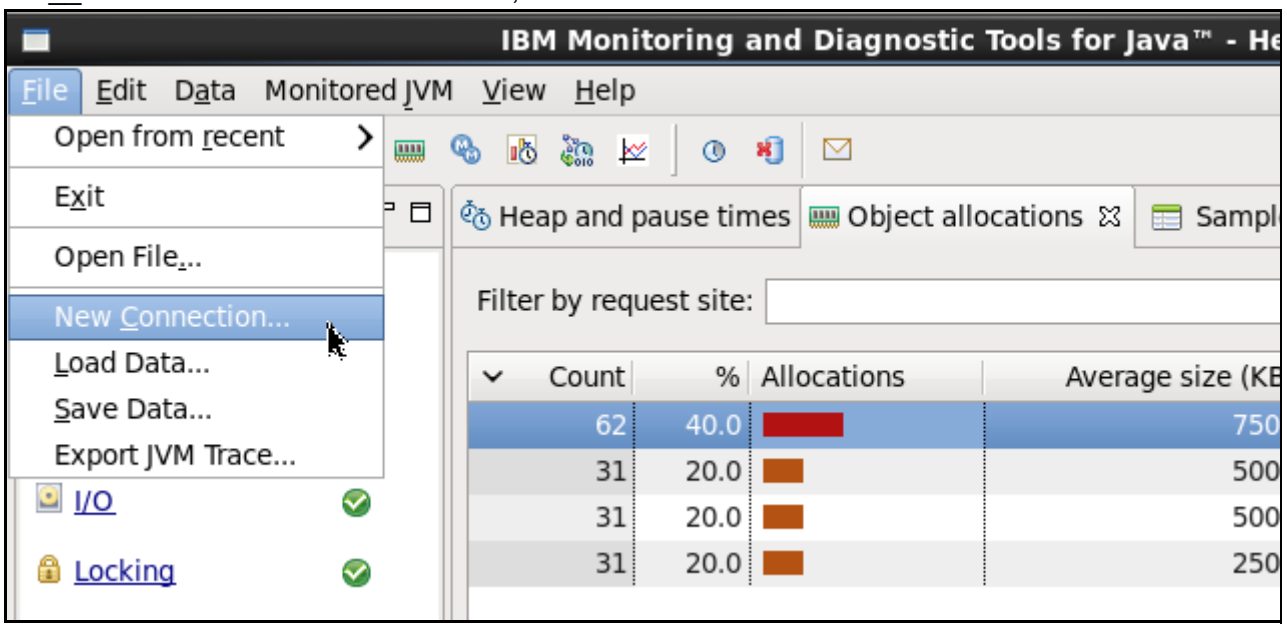
__ a. In the JMeter window, click **Run > Start**.

__ b. Wait a few moments until the number of started threads reaches **2**, as indicated in the upper-right corner of the JMeter window.

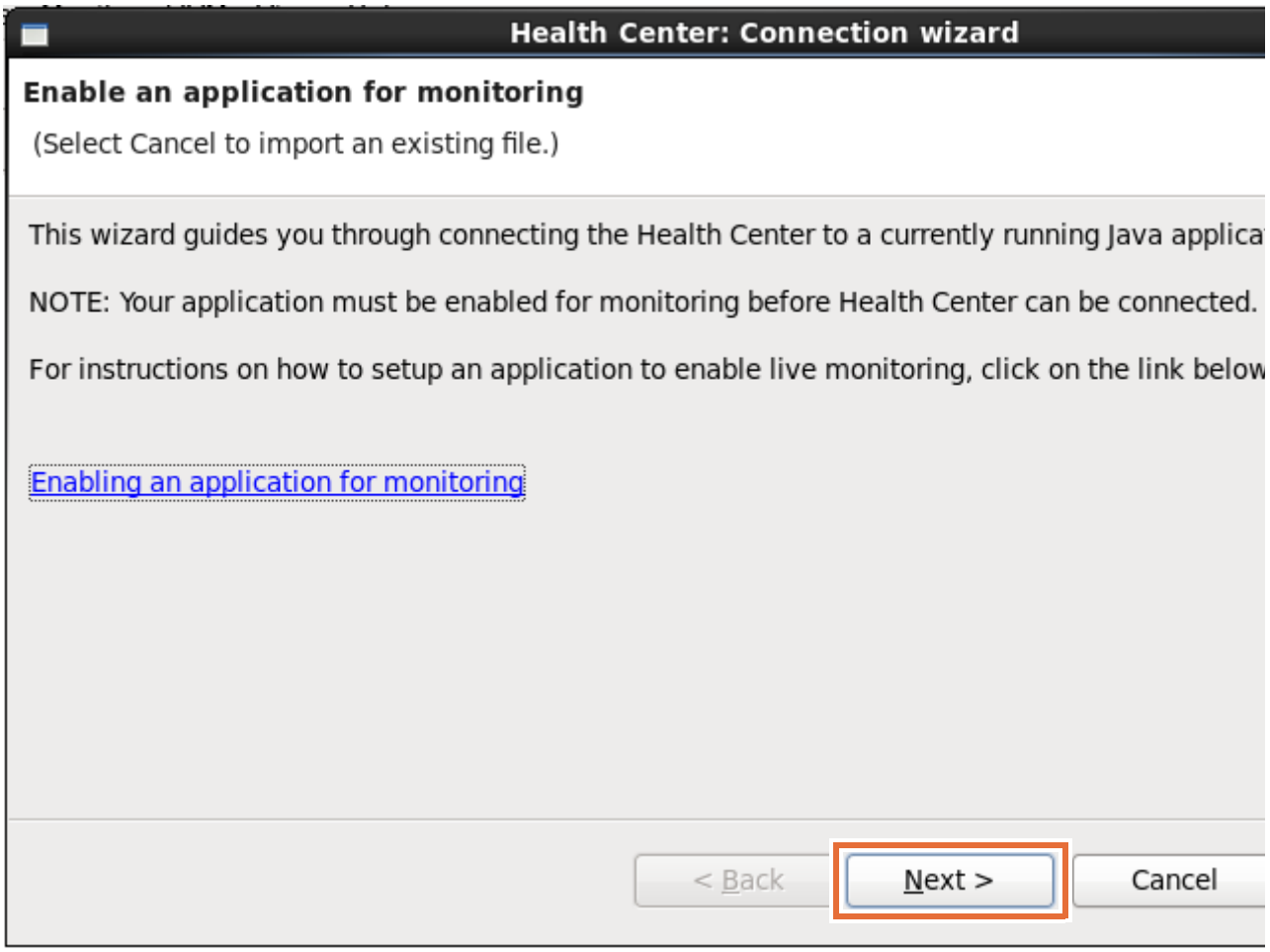


__ 5. Reconnect the Health Center to the restarted server1.

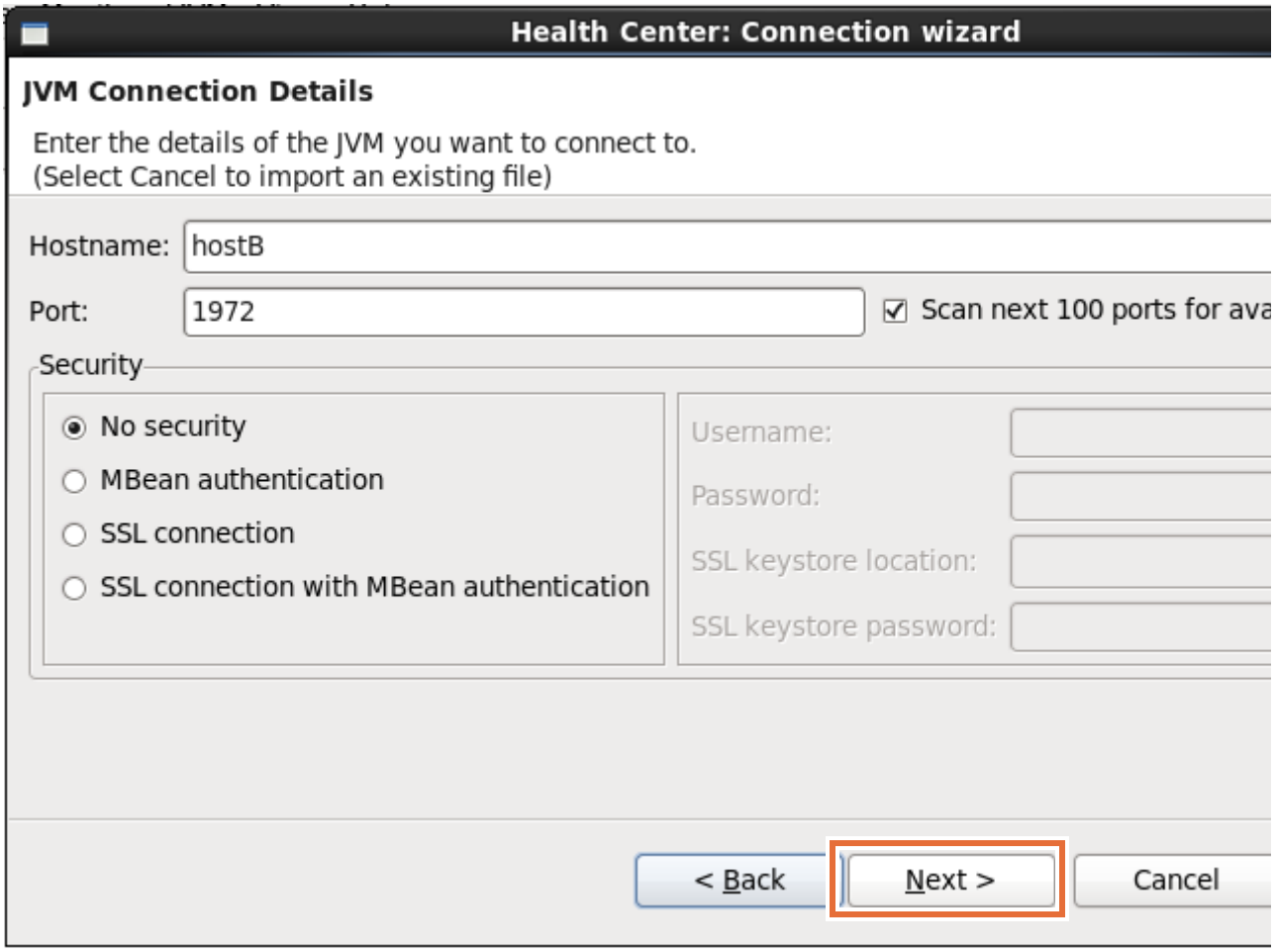
__ a. In the Health Center window, click **File > New Connection...**.



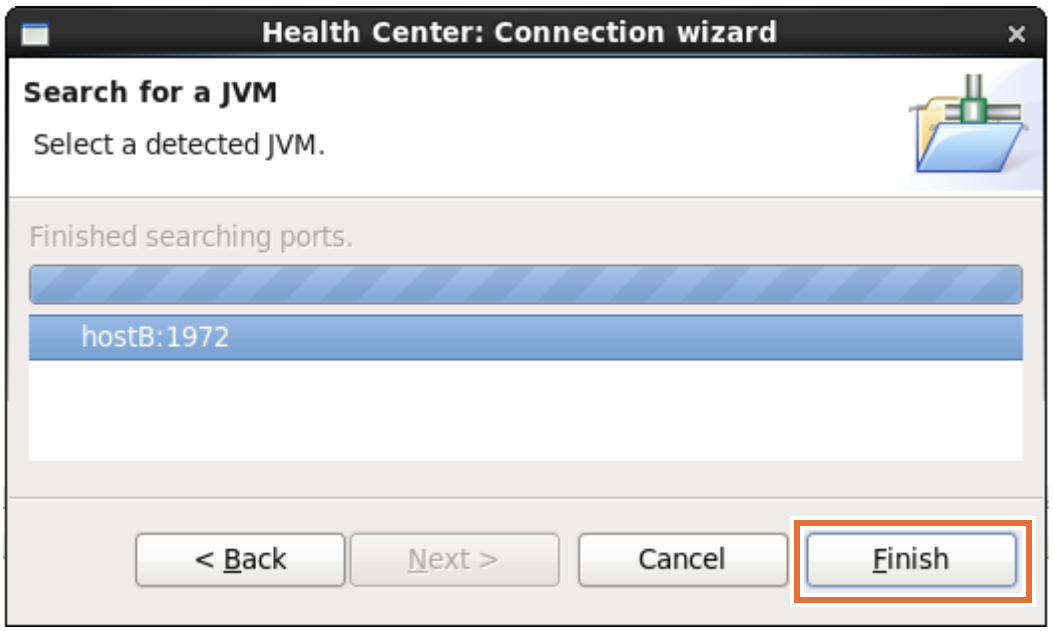
__ b. On the **Health Center: Connection wizard** screen, click **Next**.



__ c. Ensure that **No security** is selected, and click **Next**.



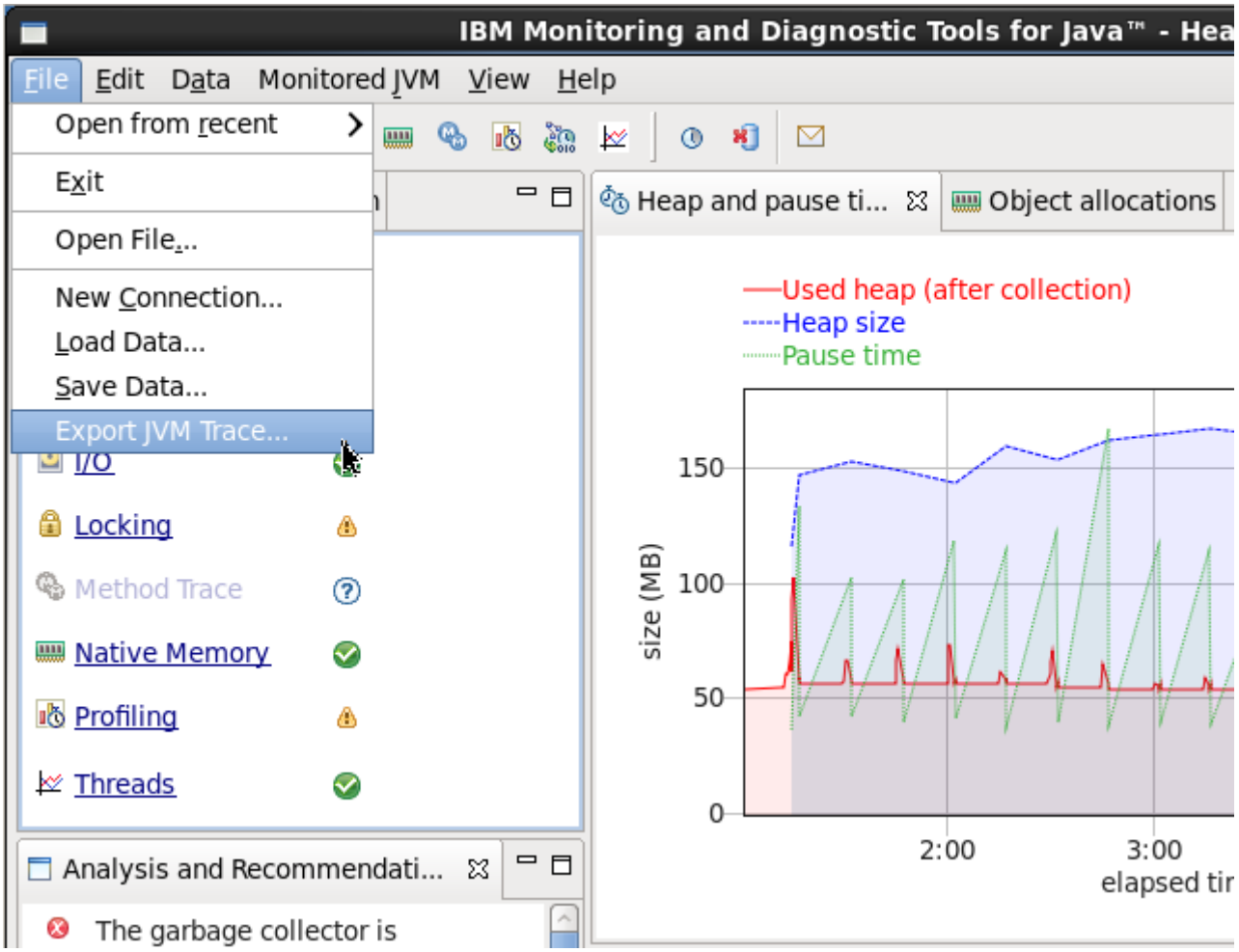
__ d. Select the **hostB** entry and click **Finish**.



__ e. Wait a few moments until the Health Center retrieves information from the application server.

__ 6. Obtain the generated trace file from server1.

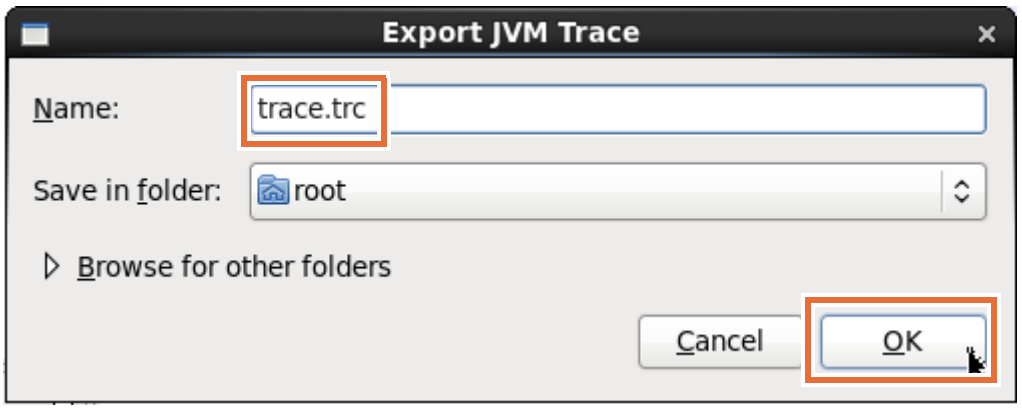
__ a. In the **Health Center** window, click **File > Export JVM Trace...**.



__ b. In the **Export JVM Trace** window, for the **Name**, enter **`trace.trc`**

__ c. For the **Save in folder**, leave **root** selected.

__ d. Click **OK**.



__ e. Wait for the export to complete. This process might take a few minutes.

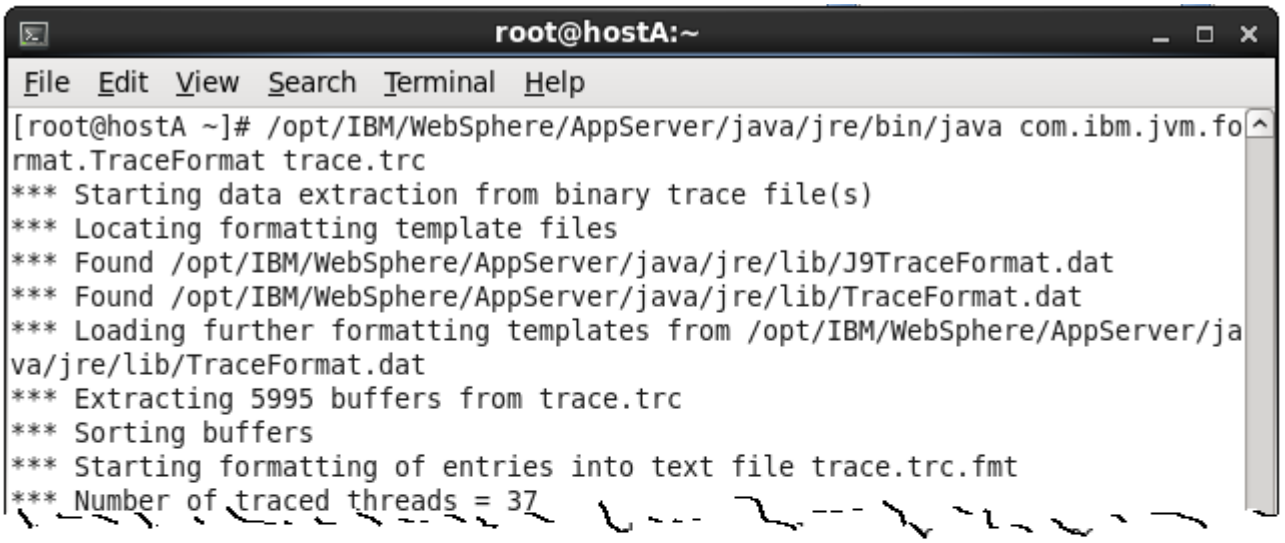
Troubleshooting

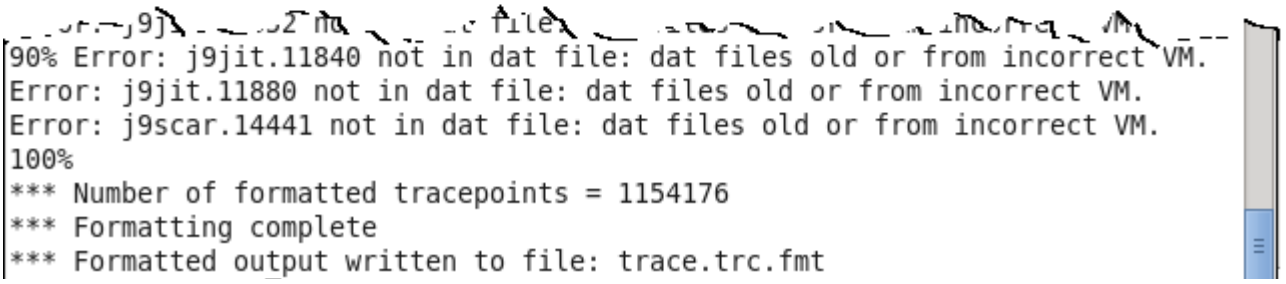
The Export JVM Trace operation might cause the Health Center to hang. The trace file is still exported successfully. If the Health Center hangs, you can close the Export JVM Trace window, then Force Quit from the application. You are also asked to terminate the existing connection on port 1972. You must restart the Health Center from the IBM Support Assistant.

__ f. Open a terminal window on **hostA**.

__ g. Make sure that you are in the root directory (enter **cd** if not).

__ h. Enter the command: **/opt/IBM/WebSphere/AppServer/java/jre/bin/java com.ibm.jvm.format.TraceFormat trace.trc**





__ i. Wait for the command to complete.

__ 7. Find the location of the System.gc() method calls.

The Java trace contains a stack trace for each time the java.lang.System.gc() method was called.

__ a. In the terminal window, enter the command: **gedit trace.trc.fmt**

__ b. **Maximize** the window.

__ c. Press Ctrl + F, and in the **Search for:** field, enter **`java.lang.System.gc`**, and click **Find**.

__ d. Click **Close**.

__ e. Notice that the stack trace shows that the System.gc method was called from the ShoppingBean deliberateSystemGC method.

```
03:47:56.761929000          0x1454b600         mt.3    Entry      >java/lang/System.gc()V Bytecode static method
03:47:56.761931000          0x1454b600  j9trc_aux.0    Event       jstacktrace:
03:47:56.761941000          0x1454b600  j9trc_aux.1    Event       [1] java.lang.System.gc (System.java:312)
com.ibm.websphere.samples.pbw.war.ShoppingBean.deliberateSystemGC (ShoppingBean.java:324)
```

__ f. Close the editor window.

__ 8. Examine the application code.

__ a. In a terminal window, enter the command:

```
gedit /usr/labfiles/AppProfiling/ShoppingBean.java
```

__ b. **Maximize** the window.

__ c. Press Ctrl + F, and in the **Search for:** field, enter **`deliberatesystemgc`**, and click **Find**.

__ d. Notice that the first instance found is the invocation of the method when the user selects Gloves.

__ e. Click **Find** again, then click **Close**. You found the implementation of the method.

__ f. Examine the method code to find where the **`System.gc()`** method is called. It is generally bad for performance for an application to explicitly ask for garbage collection.

```
private void deliberateSystemGC() {

        // ---------------------------------------------------------
        // User clicked on the Gloves, let's make them
        // wear with a System.gc
        // ---------------------------------------------------------

        System.out.println("==> STARTING SYSTEM.GC");

        System.gc();

        System.out.println("==> ENDING SYSTEM.GC");
}
```

__ g. Close the editor window.

Note

In most cases, an application should not call **`System.gc()`**. The Health Center was useful in determining that the application called the **`System.gc()`** method. The location of the **`System.gc()`** call might be determined by examining the log files of the application server. In this exercise, the Health Center was used to retrieve the remote trace log so that it might be examined on the local system.

Reset the environment.

__ 1. Stop the JMeter load generation.

__ a. In the JMeter window, click **Run > Stop**.

__ 2. Stop tracing System.GC calls.

__ a. In the administrative console, click **Servers > Server Types > WebSphere application servers**.

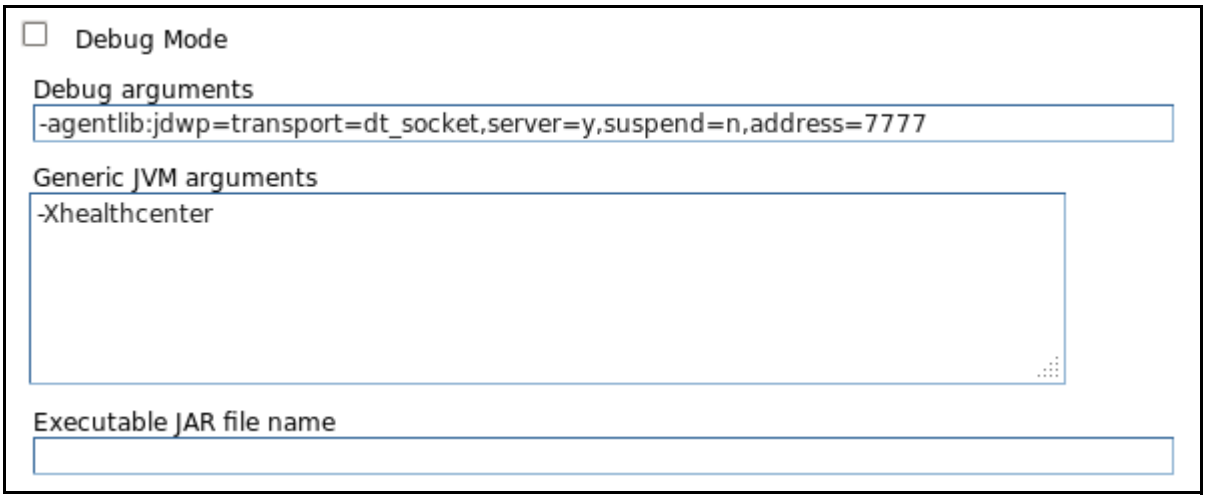
__ b. Click **server1**.

__ c. On the right side, under **Server Infrastructure**, expand **Java and Process Management**.

__ d. Click **Process definition**.

__ e. On the right side, under **Additional properties**, click **Java Virtual Machine**.

__ f. Remove the **`-Xtrace:print=mt,methods={java/lang/System.gc},trigger=method{java/lang/System.gc,jstacktrace}`** string from the **Generic JVM arguments** field.



Important

Make sure that you leave the **`-Xhealthcenter`** argument.

__ g. Click **OK**.

__ h. **Save** to the master configuration.

__ i. Click **Application servers** in the breadcrumb trail.

__ j. Select **server1**.

__ k. Click **Restart**.

__ l. Wait for the server to restart.

Section 4: Check the locking and threading behavior of the application

Lock contention occurs when a lock is in use and another thread tries to acquire it. High levels of contention can occur on locks that are obtained frequently, held for a long time, or both. A highly contended lock (one in which many threads try to gain access) can become a bottleneck in the system. Each running thread pauses its execution until the lock it requires becomes available, limiting application performance. Synchronization is especially likely to hinder application scalability. The Health Center visualizes lock activity and highlights locks that are blocking requests and potentially affecting performance.

No synchronization errors were added to the modified PlantsByWebSphere sample application. This section of the exercise shows how the Health Center is used to examine locking and threading information.

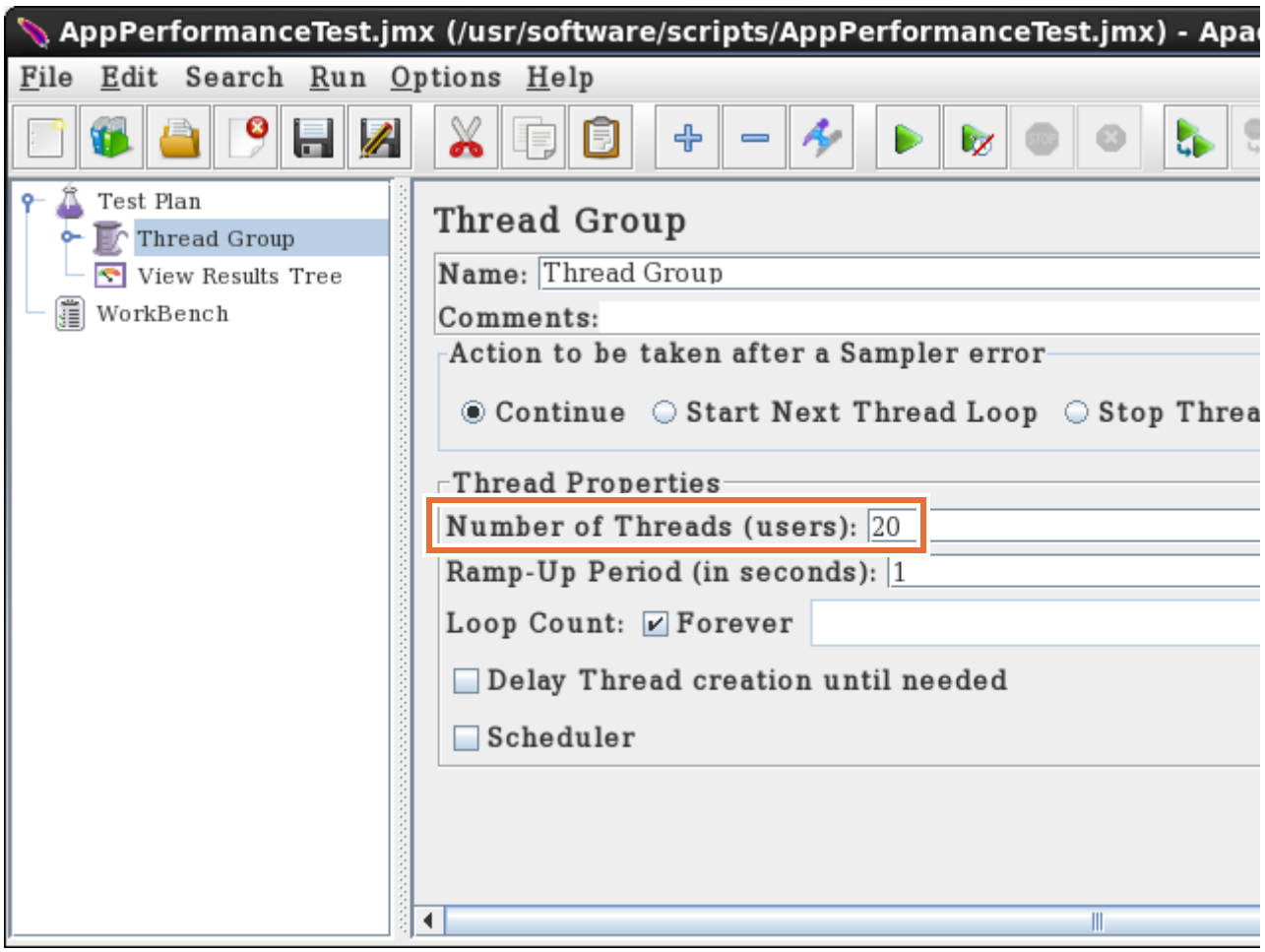
__ 1. Generate load for the application.

__ a. If JMeter is not started, in a terminal window, enter the command:
`/opt/apache-jmeter/bin/jmeter.sh -t /usr/labfiles/AppProfiling/AppPerformanceTest.jmx -H hostB -P 9080`

__ b. In the JMeter window, on the left side, expand **Test Plan** if it is not already expanded.

__ c. Click **Thread Group**.

__ d. On the right side, in the **Number of Threads (users)** field, enter **20**.



__ e. Click **Run > Start**.

__ f. Wait a few moments until the number of started threads reaches **20**, as indicated in the upper-right corner of the JMeter window.

__ 2. Reconnect the Health Center to the restarted server1.

__ a. In the Health Center window, click **File > New Connection...**.

__ b. On the **Health Center: Connection wizard** screen, click **Next**.

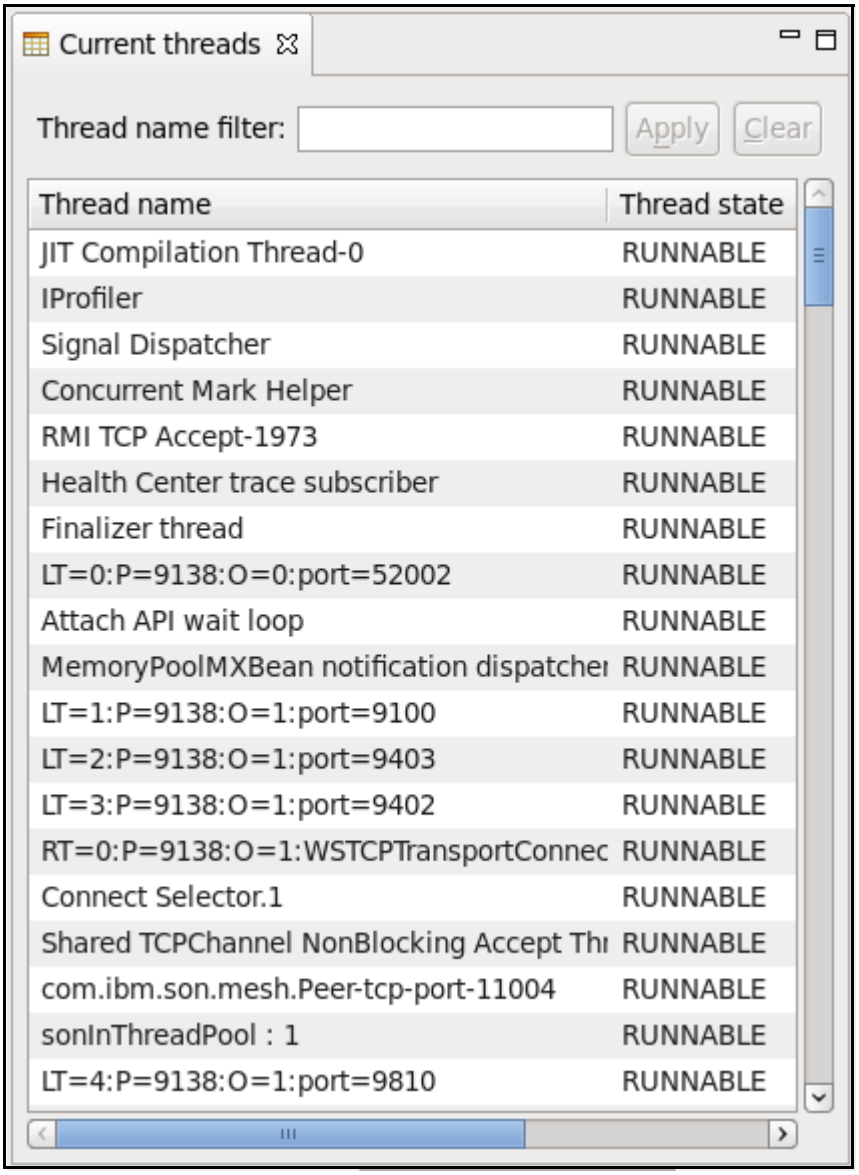
__ c. Ensure that **No security** selected, and click **Next**.

__ d. Select the **hostB** entry and click **Finish**.

__ e. Wait a few moments until the Health Center retrieves information from the application server.

__ 3. Examine the threads of the application.

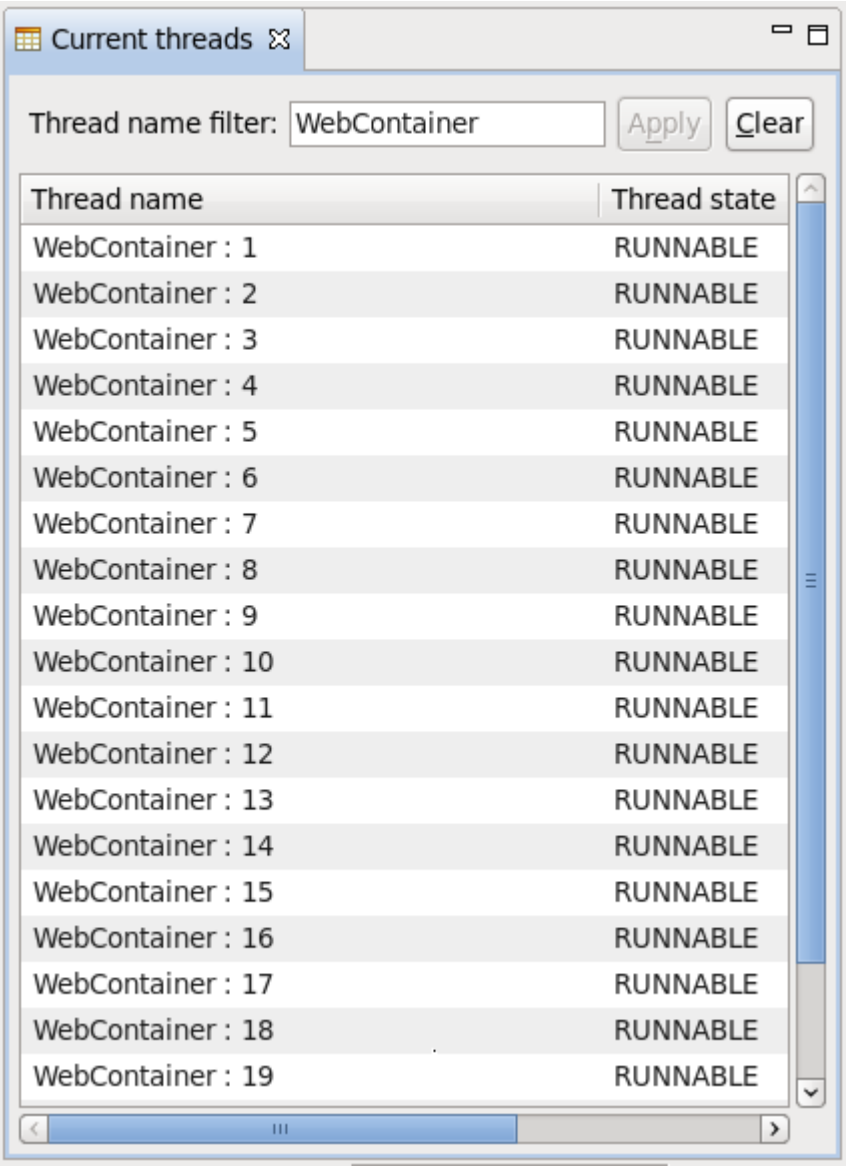
__ a. In the Health Center Status pane, click the **Threads** link.



__ b. Web container threads do application work. In the **Thread name filter** field, enter **`WebContainer`**, and click **Apply**.

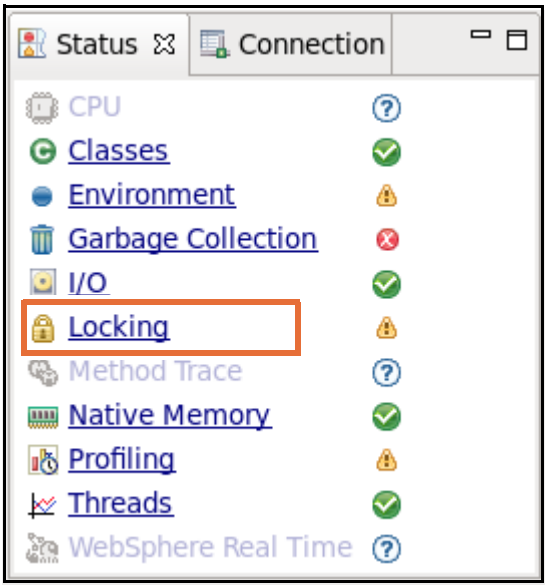
__ c. If you see many threads with a **Thread state** of **BLOCKED**, it might indicate a synchronization problem with your application. You should not see many (if any) threads in the BLOCKED state for the sample application.

__ d. Selecting a BLOCKED thread shows the monitor (or lock) that the thread is waiting for in the Thread details view.

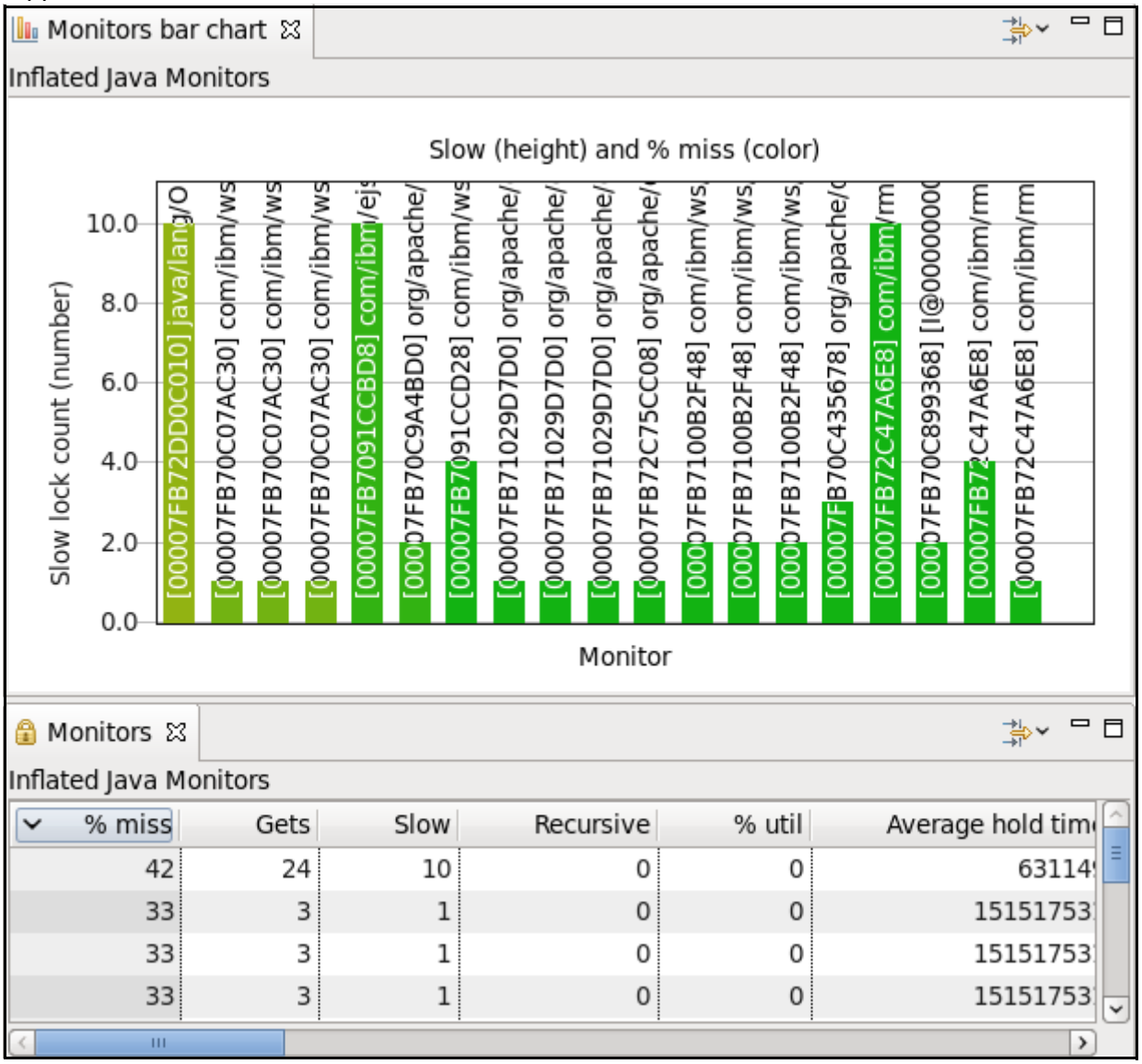


__ 4. Examine the locks of the application.

__ a. In the Health Center window, click the **Locking** link.



The Locking perspective shows a graph and a table with information about the locks in the application.



Information

Interpreting the monitor data

The bars in the graph are color-coded green through to red to indicate how contended the locks are. Orange or red color indicates most of the attempts to synchronize were blocked. A tall bar indicates that many requests were blocked. The color and the height together therefore show which locks are the most contended.

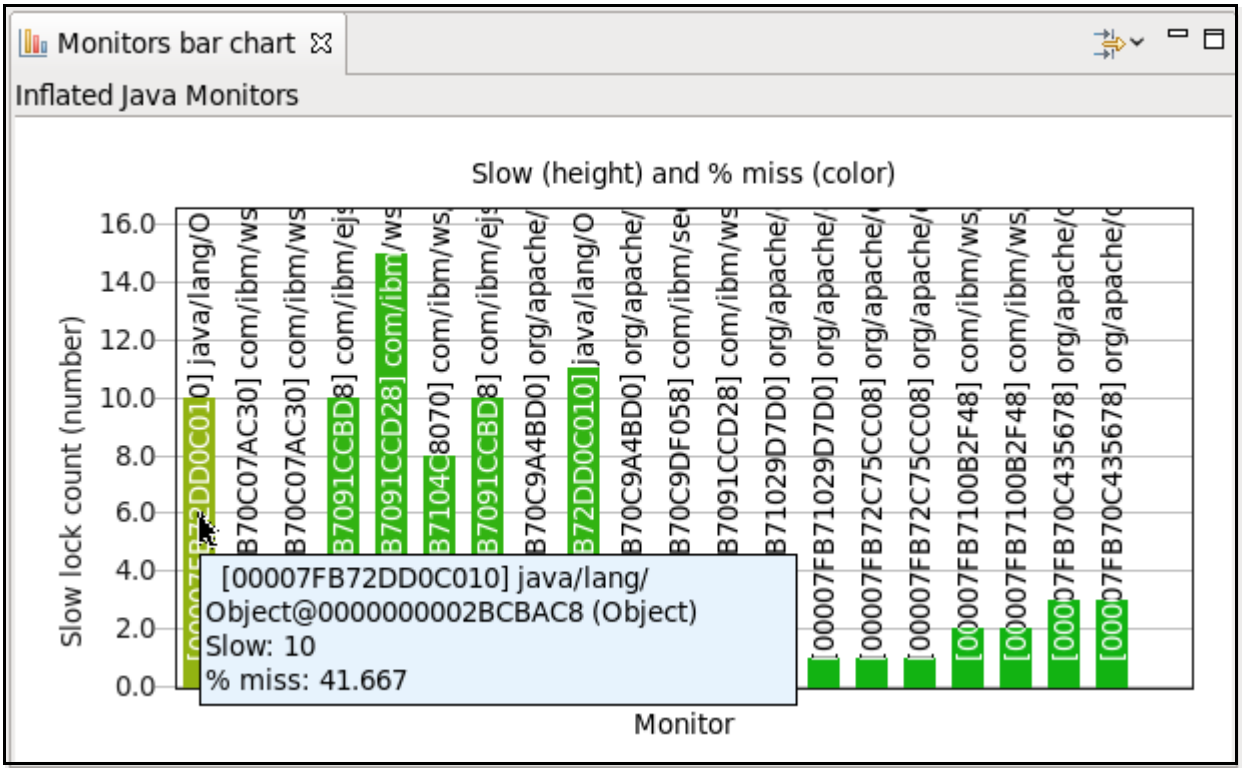
A lock can have a high bar but still be a green color. This observation would mean that although the lock had a high number of blocked requests, overall it had a high number of successful acquires so the percentage of overall blocked requests would be low. It is still possible that this behavior might affect performance.

__ 5. Examine the Locking perspective graph.

__ a. Notice whether any of the bars are of a color other than green. This observation is a good indication that the application has lock contention issues. It is unlikely that any of the bars are other than green for the sample application.

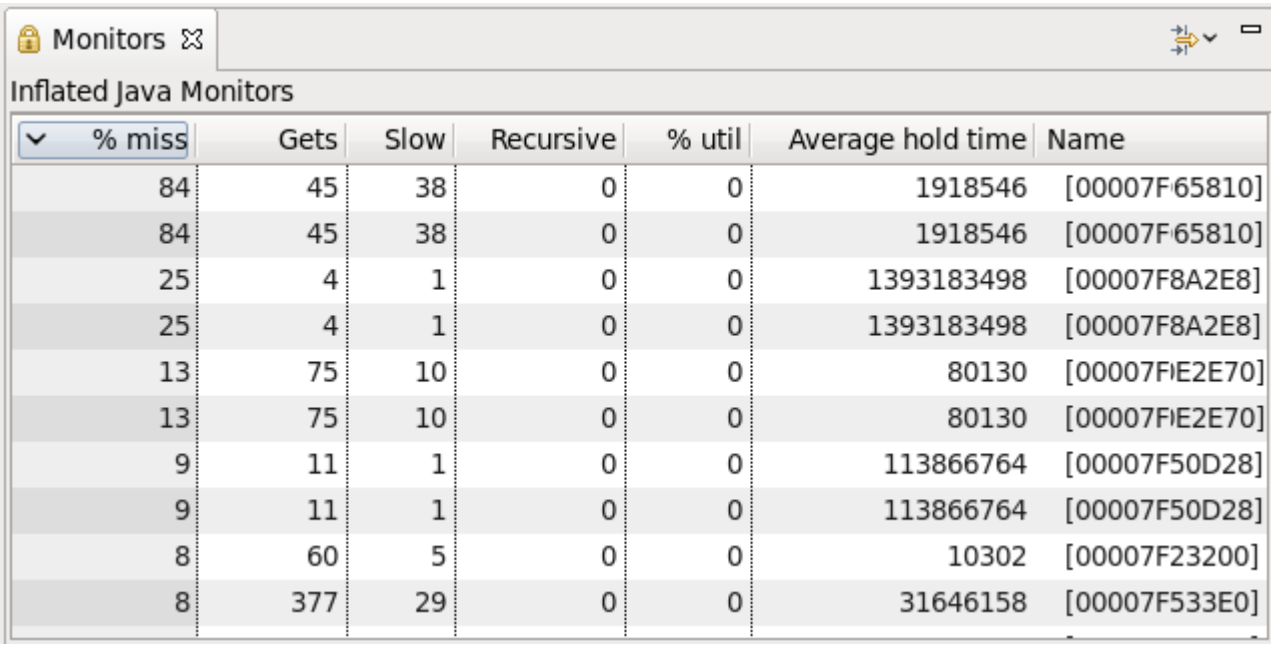
__ b. Hover over one of the bars. The identity of the lock object is shown.

Sometimes the identity information is enough to identify the code that uses the lock. In many cases, the name of the lock is not enough. You can take a series of thread dumps to see the call stacks of blocked threads. Alternatively, you can use the address of the lock object. The identity string begins with the address of the lock object in square brackets. You might take a system dump to find what code references that object.



__ 6. Examine the locking perspective table.

__ a. Click the **% miss** column. The locks that had the highest percentage of threads that had to wait for the lock are at the top.



% miss	Gets	Slow	Recursive	% util	Average hold time	Name
84	45	38	0	0	1918546	[00007F65810]
84	45	38	0	0	1918546	[00007F65810]
25	4	1	0	0	1393183498	[00007F8A2E8]
25	4	1	0	0	1393183498	[00007F8A2E8]
13	75	10	0	0	80130	[00007FE2E70]
13	75	10	0	0	80130	[00007FE2E70]
9	11	1	0	0	113866764	[00007F50D28]
9	11	1	0	0	113866764	[00007F50D28]
8	60	5	0	0	10302	[00007F23200]
8	377	29	0	0	31646158	[00007F533E0]

__ b. Examine the **Average Hold Time** column. If this number is large, and the %miss is high, you might be trying to do too much work while holding the lock.

__ c. Examine the **%util** column. If this number is large, and the % miss is high, you might have too many parts of your code with the same lock.

__ d. Examine the **Name** column. This field shows the thread identity in the same format as in the graph. If the thread name is not enough to identify the code that uses the lock, take thread dumps or system dumps to find the code that references the lock.

Section 5: Clean up

__ 1. Stop load generation.

__ a. In the JMeter window, click **Run > Stop**.

__ b. Close the JMeter window. Do not save the changes to the modified test.

__ 2. Close the Java Health Center.

__ 3. Close the IBM Support Assistant.

End of exercise

Exercise review and wrap-up

In the first part of the exercise, you configured the environment to allow application analysis with the Java Health Center. In the second part of the exercise, you analyzed the application for slow performing methods, and for memory problems. You also used the Health Center to examine the Threading and locking behavior of the application.

Exercise 11. Load testing an application server cluster

What this exercise is about

In this exercise, you create a horizontal cluster for the DayTrader application. You then use the IBM HTTP plug-in to route requests to DayTrader. Finally, you run load tests to measure the scalability of a horizontal cluster.

What you should be able to do

At the end of this exercise, you should be able to:

- Install the DayTrader Application to a cluster
- Configure the IBM HTTP plug-in to route requests to the cluster
- Use Apache JMeter to load test the cluster
- Use the Apache server-status tool to monitor web server usage
- Monitor CPU usage on cluster member hosts
- Reconfigure cluster member weights to make better use of system resources

Introduction

Clusters enable enterprise applications to scale beyond the amount of throughput that you can achieve with a single application server. In this exercise, you install the DayTrader application to a cluster called TradeCluster. The cluster consists of two members, TradeServer1 on hostB and TradeServer2 on hostC. You use the IBM HTTP plug-in, which runs on hostA to route requests to the cluster by using round robin load balancing.

You use Apache JMeter to load test the TradeCluster and observe performance metrics such as response time and throughput. You also explore the configuration of cluster member weights to make better use of system resources such as processor usage.

Requirements

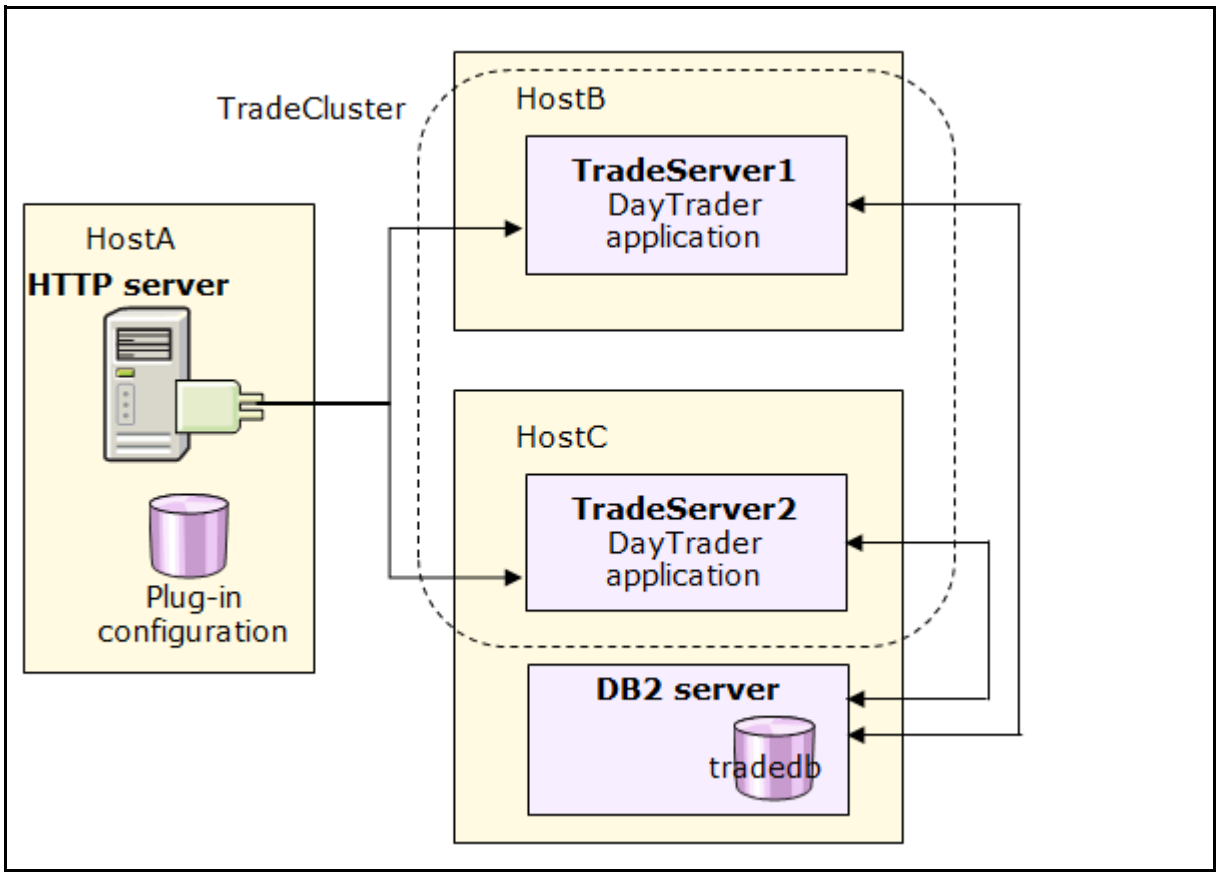
This exercise depends on the first exercise **Pod Configuration** and **Exercise 4: Use Apache JMeter to load test DayTrader** must be successfully completed before you can do this exercise. At the end of Exercise 4, you created an Apache JMeter test plan,

`/usr/labfiles/Test_Plans/DayTrader_baselineXX_test.jmx`, which you are going to use in this exercise.

Exercise instructions

Preface

- Most steps in this exercise are done on **hostA** unless otherwise specified.
- In this exercise, you run Jython scripts to create a horizontal cluster environment and deploy the DayTrader application to the cluster. Certain application resources such as the Trade database data source are scoped to the cell rather than the node as they were for the single-server environment.



Note

`<profiles_root> = /opt/IBM/WebSphere/AppServer/profiles`

Section 1: Start the WebSphere servers and verify connectivity

If the Deployment Manager and node agents are not already running, start them now.

__ 1. On **hostA**, start the Deployment Manager.

__ a. Open a terminal window and enter the command:
cd <*profiles_root*>/Dmgr/bin

__ b. Enter the command: **./startManager.sh**

__ 2. On **hostB**, start the node agent.

__ a. Double-click the **ssh hostB** icon on your desktop to access a terminal window on **hostB**.

__ b. In the terminal window, enter the command:
cd <*profiles_root*>/profile1/bin

__ c. Enter the command: **./startNode.sh**

__ 3. On **hostC**, start the node agent.

__ a. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ b. In the terminal window, enter the command:
cd <*profiles_root*>/profile2/bin

__ c. Enter the command: **./startNode.sh**

__ 4. Make sure that your three hosts can successfully ping each other.

__ a. From **hostA,** ping hostB and hostC.

__ b. From **hostB,** ping hostA and hostC.

__ c. From **hostC,** ping hostA and hostB.

Section 2: Uninstall the DayTrader and remove its resources

In this section, you use a script to uninstall DayTrader and clean up the single-server environment. If for some reason you never installed DayTrader, skip this section.

__ 1. Log in to the administrative console with the user ID **wasadmin** and password **web1sphere**

__ 2. Stop TradeServer1 if it is running.

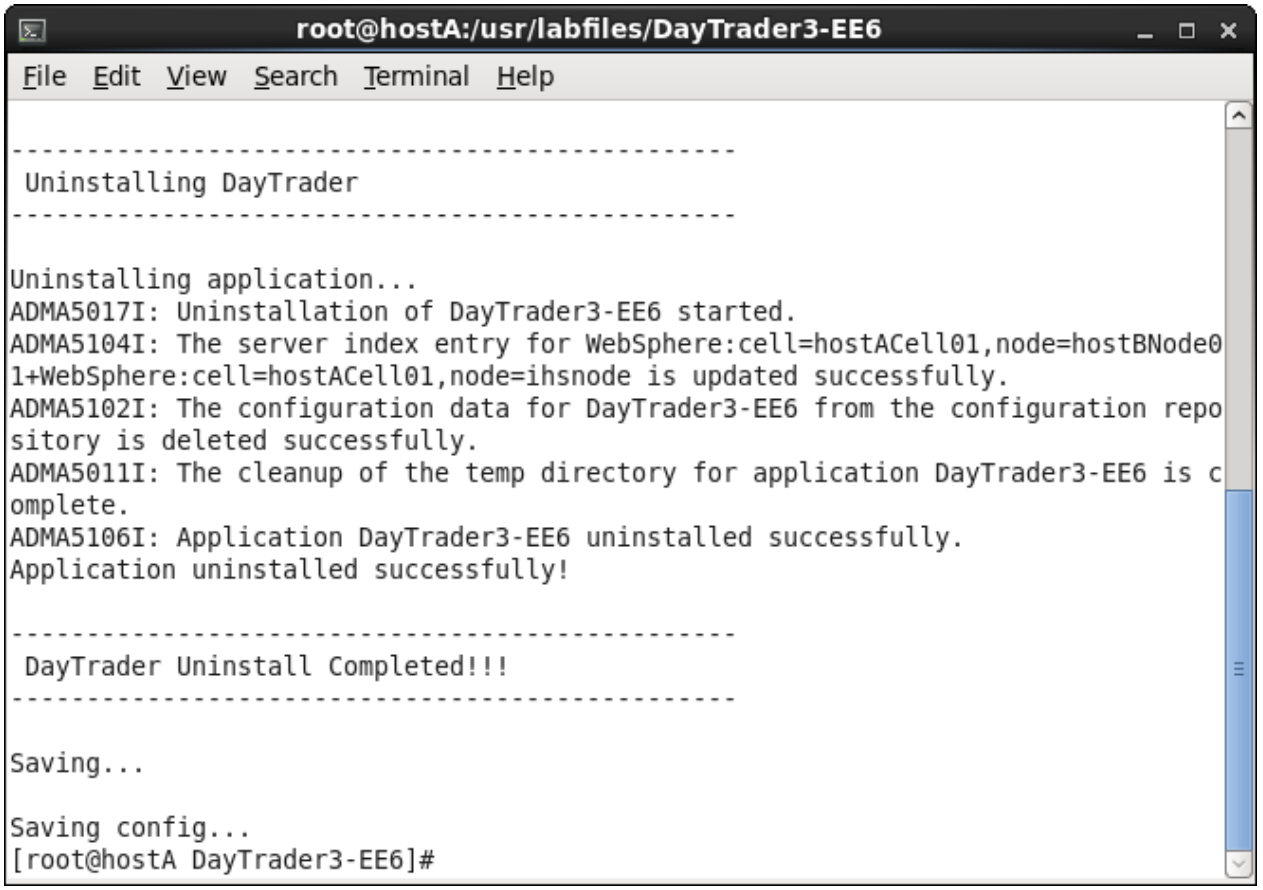
__ 3. Start the DayTrader uninstallation and unconfiguration scripts.

__ a. On **hostA**, start a new terminal window and enter the command:
cd /usr/labfiles/DayTrader3-EE6

__ b. Uninstall DayTrader by entering the following command on one line:

/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin/wsadmin.sh -f daytrader_singleServer.py uninstall -user wasadmin -password web1sphere

__ c. Wait until you see the following messages.

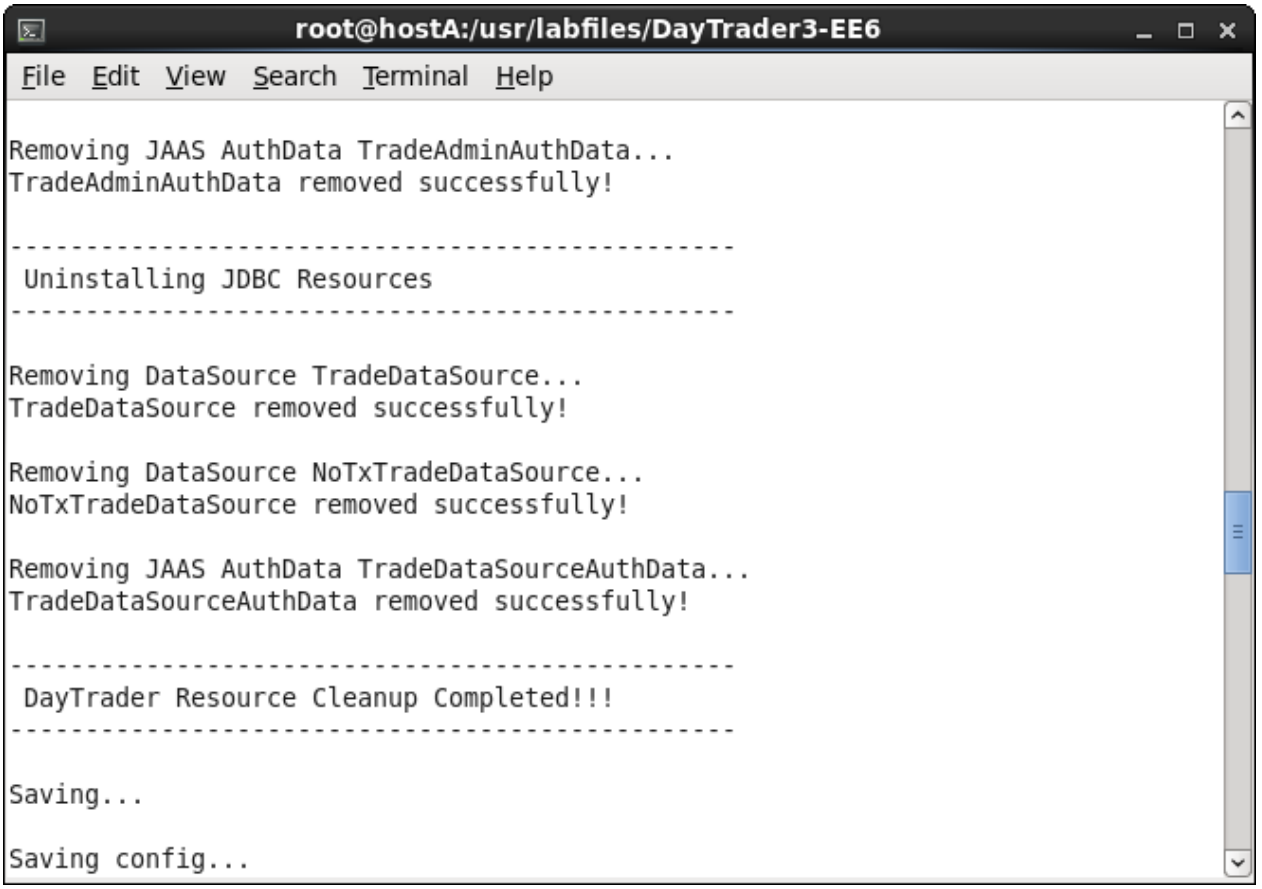


__ d. Remove the JDBC and JMS resources by entering the following command on one line:

/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin/wsadmin.sh -f daytrader_singleServer.py cleanup -user wasadmin -password web1sphere

__ e. When prompted for the node, enter: **hostBNode01**

__ f. Wait until you see the following messages:



__ 4. Use the administrative console to verify that the DayTrader application and its resources are removed.

__ 5. Delete TradeServer1.

__ a. From the administrative console, uninstall the Dynamic Cache Monitor application.

__ b. Then, delete **TradeServer1.**

Section 3: Reinstall DayTrader to the TradeCluster

In this section, you use a Jython script to create the TradeCluster with two members, one on hostB and one on hostC. The JDBC and JMS resources are created at the cell scope level.

__ 1. From the same terminal window on **hostA** that you used in the last section, enter the following command:

```
/opt/IBM/WebSphere/AppServer/profiles/Dmgr/bin/wsadmin.sh -f
daytrader_cluster.py -user wasadmin -password web1sphere
```

__ 2. Supply script information. The Jython script asks a series of questions that it uses to configure the cluster and the necessary JDBC and JMS resources. As you progress through the interactive steps, enter the following information and press the Enter key.

__ a. Global security is enabled: **`true`**

__ b. Have all nodes been federated and network connectivity verified? (yes|no) [yes]: **yes**

__ c. Enter the cluster name [TradeCluster]: **TradeCluster**

__ d. Select the desired node [hostACellManager01]: **hostBNode01**

__ e. Enter the cluster member name [TradeServer1]:**TradeServer1**

__ f. Add more cluster members (yes|no) [yes]: **yes**

__ g. Select the desired node [hostACellManager01]: **hostCNode01**

__ h. Enter the cluster member name [TradeServer2]: **TradeServer2**

__ i. Add more cluster members (yes|no) [yes]: **no**

__ j. Select the JDBC provider type [DB2 Universal]: **DB2 Universal**

__ k. Select the EJB deployment target [DB2UDB_V82]: **DB2UDB_V82**

__ l. Enter the location of JDBC driver (jar) files: **/home/db2inst1/sqllib/java/db2jcc.jar;/home/db2inst1/sqllib/java/db2jcc_license_cu.jar**

__ m. Enter the database name (location) [tradedb]: **tradedb**

__ n. Enter the database hostname [localhost]: **hostC**

__ o. Enter the database port number [50000]: **50000**

__ p. Enter the database username [userid]: **db2inst1**

__ q. Enter the database password [password]: **was1edu**

__ r. Enter the ME database name (location) [tradedb]: **tradedb**

__ s. Enter a valid administrative username [AdminUserID]: **wasadmin**

__ t. Enter a valid administrative password [password]: **web1sphere**

__ 3. The script begins creating the cluster and all of the application resources and installs the DayTrader application.

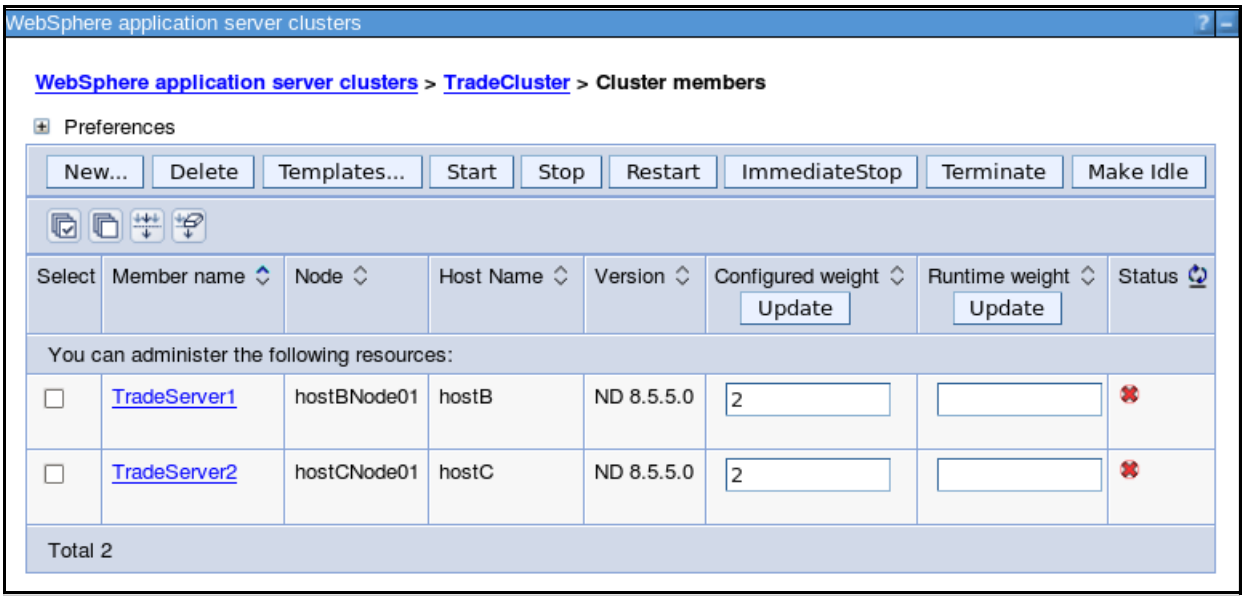
__ a. Scroll through the terminal window and verify that there are no error messages or warnings.

__ b. Verify that the script completes with the message:

```
DayTrader Installation Completed!!!
```

__ 4. Verify the **TradeCluster** from the administrative console.

__ a. Click **Servers > Clusters > WebSphere application server clusters > TradeCluster > Cluster members**.



__ b. Start **TradeServer1** and **TradeServer2** by selecting them and click **Start**.

__ 5. Verify that the WC_defaulthost ports are **9081** for both cluster members.

__ a. Click **TradeServer1 > Ports**.

__ b. Click **TradeServer2 > Ports**.

__ 6. After the DayTrader application is installed and started, you can access it at the following web address: **`http://hostb:9081/daytrader`** or **`http://hostc:9081/daytrader`** However, before accessing any of the trading application functions, the required database tables must be created and populated with an initial set of accounts, stock quotes, and holdings. Follow the simple steps that are outlined next to configure the database tables and populate them with data.

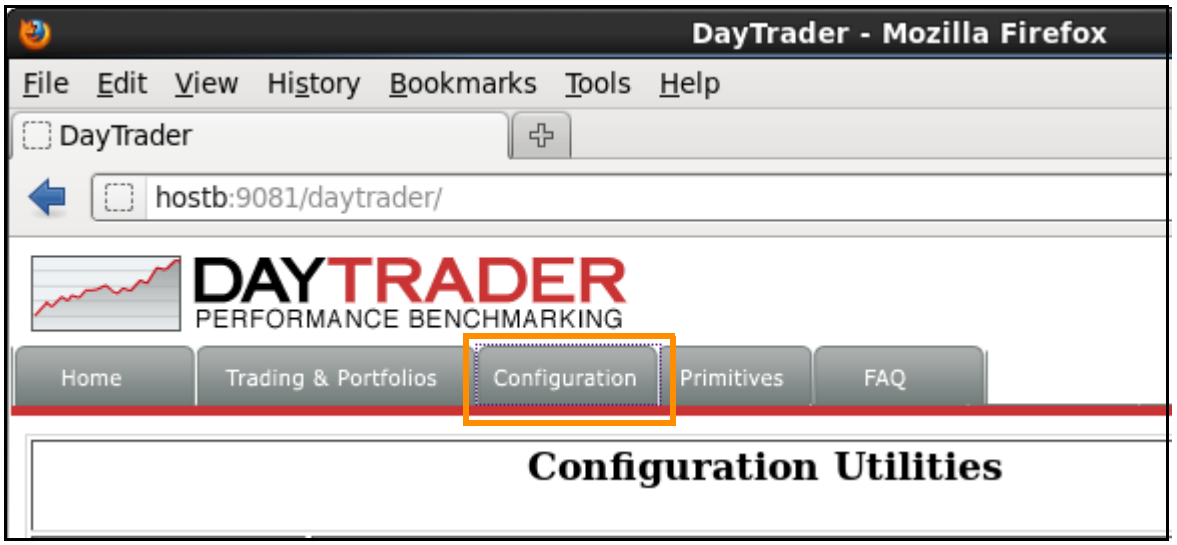
__ a. Open a web browser and enter the web address: **`http://hostb:9081/daytrader`**

Troubleshooting

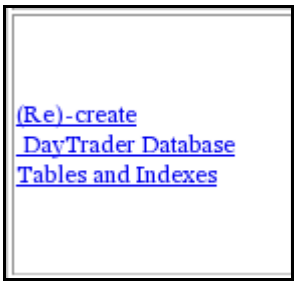
Virtual host error message

If for some reason you see an error message in the browser, "**`SRVE0255E: A WebGroup/Virtual Host to handle hostb:9081 has not been defined`**", make sure that entries exist in the default_host for hostb 9081 and hostc 9081. Then, restart **TradeServer1** and **TradeServer2**.

__ b. Click the **Configuration** tab.



__ c. Scroll down and click the **(Re)-create Day Trader Database Tables and Indexes** link.



Information

This link is used to (a) initially create or (b) drop and re-create the DayTrader tables. A DayTrader database should exist before doing this action. The existing DayTrader tables, if any, are dropped, and then new tables and indexes are created.

The initial population size consists of 15000 accounts and 10000 stock quotes. These values can be updated by using the **Configure DayTrader runtime parameters** link on the **Configuration** tab.

__ d. Wait for the database tables and indexes to be created. You should see the following messages when the process is complete.

```
TradeBuildDB: **** Dropping and Recreating the DayTrader tables... ****


TradeBuildDB: **** DayTrader tables successfully created! ****
```

__ e. Close the browser and open a new browser.

__ f. Log in to the administrative console.

__ g. Restart the application **DayTrader3-EE6**.

__ h. Open a web browser and enter the web address **`http://hostB:9081/daytrader`**

__ i. Click the **Configuration** tab.

__ j. Click the **(Re)-populate DayTrader Database** link.

__ k. Wait several minutes for the operation to complete. You should see messages similar to the following when the process is complete.

```
TradeBuildDB: **** Creating 10000 Quotes ****
.....s:0 -
.....s:10.....s:20.....s:30.....s:40.....s:50.....s:60.....s:70.....s:80.
....s:90.....s:100 -
.
.
**** Registering 15000 Users ****
Account# 1 userID=uid:0 has 8 holdings.
Account# 51 userID=uid:50 has 3 holdings.
.
.
Account# 451 userID=uid:450 has 10 holdings.
```

__ l. You also see a detailed description of the current DayTrader configuration. Read the description and close the browser when you are finished.

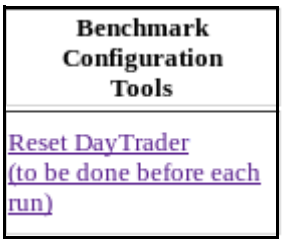
__ m. Close the DayTrader browser.

__ 7. Verify access to the DayTrader on both cluster members.

__ a. Open a web browser and enter the web address: **http://hostB:9081/daytrader**

__ b. On the DayTrader welcome page, click the **Configuration** tab.

__ c. Click the **Reset DayTrader** link.



__ d. Verify that DayTrader resets successfully.

DayTrader Scenario Runtime Statistics

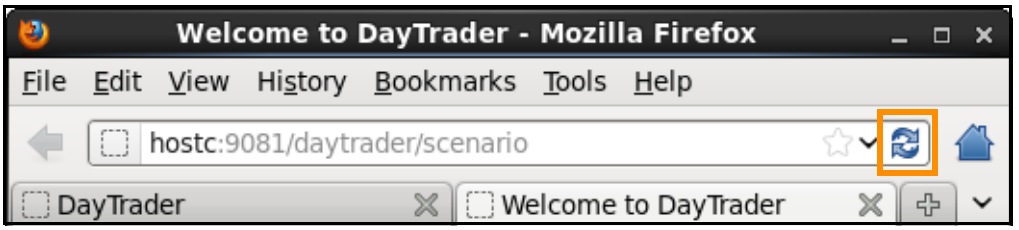
Trade Reset completed successfully

Modify runtime configuration

Benchmark runtime configuration summary	Value
Run-Time Mode	Full EJB3
Order-Processing Mode	Synchronous
Scenario Workload Mix	Standard
Web Interface	JSP
Active Traders / Trade User population	15000 / 15000
Active Stocks / Trade Stock population	10000 / 10000

__ e. Click the **Configuration** tab again, and click the **Test DayTrader Scenario** link.

__ f. Step through the scenario by pressing F5 or the reload icon on the browser.



__ g. Close the browser when your are satisfied that the DayTrader working properly.

__ h. Open a new web browser and enter the web address: **`http://hostC:9081/daytrader`**

__ i. Repeat **Steps b-g** to verify that DayTrader is working properly on **TradeServer2**.

Section 4: Map DayTrader to the TradeCluster and web server and test access through the HTTP server

In this section, you map the DayTrader application modules to TradeCluster and the web server. You also verify that you can access DayTrader through the web server and plug-in.

__ 1. Map the DayTrader's web module to the **web server**.

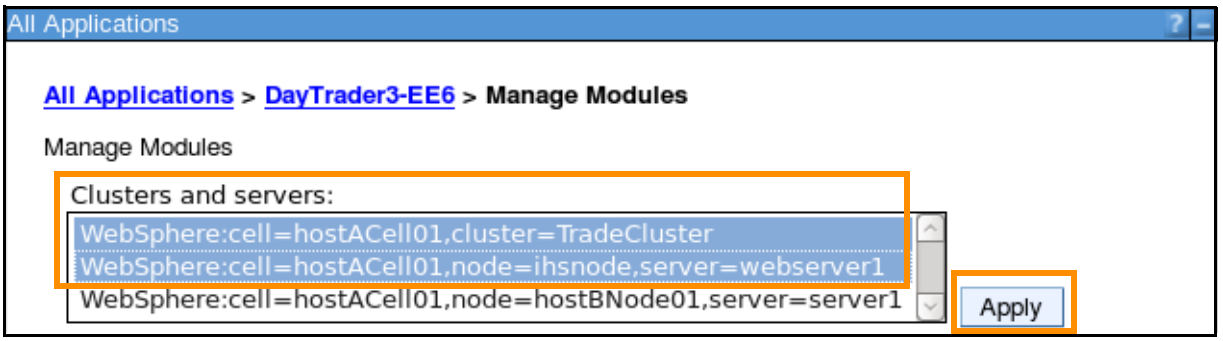
__ a. From the administrative console, click **Applications > Application Types > WebSphere enterprise applications**.

__ b. Click **DayTrader3-EE6**.

__ c. Click **Manage Modules**.

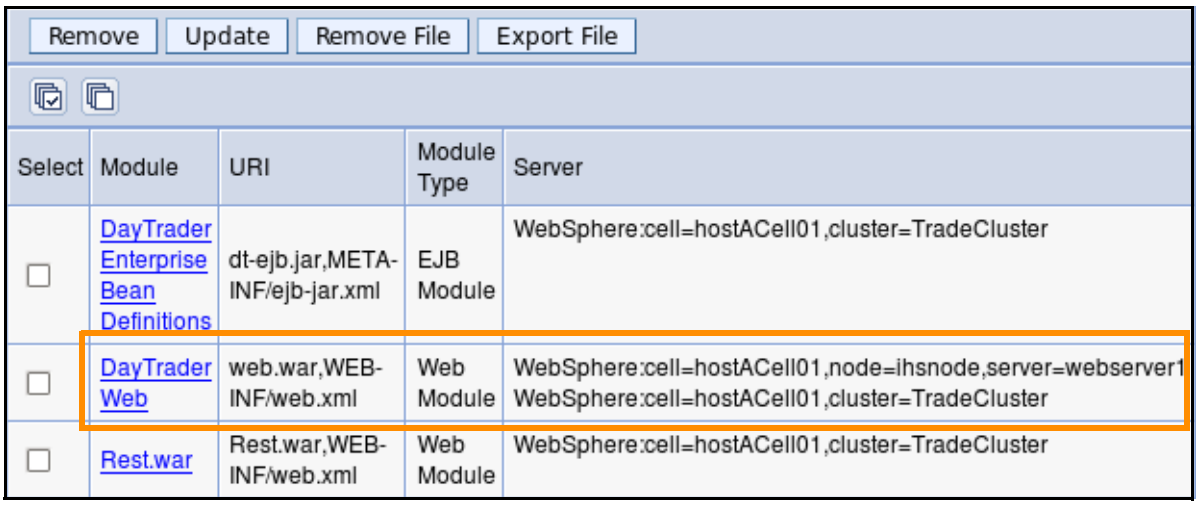
__ d. Select only the **DayTrader Web** check box.

__ e. In the Clusters and servers list, press Ctrl and select both the check boxes for **WebSphere:Cell=hostACell01,cluster=TradeCluster** and **WebSphere:Cell=hostACell01,node=ihsnode, server=webserver1**.



__ f. Click **Apply**.

__ g. Verify that the mapping is correct.



__ h. Click **OK**.

__ i. **Save** the changes.

__ j. Wait for the node to synchronize and click **OK**.

__ 2. If not already started, start the web server on **hostA**.

__ a. In a terminal window, enter: **`cd /opt/IBM/HTTPServer/bin`** and then enter the following commands:

```
./apachectl start
./adminctl start
```

__ 3. Verify that the DayTrader can be accessed by using the web server and web container port.

__ a. Open a new web browser and verify access by using the web server by entering the address: **`http://hostA/daytrader`**

__ b. Open a new web browser tab and verify access by using the web server by entering the address: **`http://hostB:9081/daytrader`**

__ c. Open a new web browser tab and verify access by using the web server by entering the address: **`http://hostC:9081/daytrader`**

__ d. Close the all DayTrader browser tabs.

Note

Recall that the web server is running on hostA, TradeServer1 is running on hostB, and TradeServer2 is running on hostC. The URL must point to the correct host for the request to get routed correctly.

Information

Initial TradeCluster configuration

The script that creates the TradeCluster, also applies some initial configuration to tune the environment. For example, the Trade data source properties are given the following values.

- Maximum connections = 50
- Minimum connections = 10
- Statement cache size = 60

Also, the scope of the Trade data source is now set to cell. If you have time, you can explore the configuration of some of the other TradeCluster properties by examining the **`daytrader_cluster.py`** script.

Section 5: Tune the tradedb database and back it up

This section shows you the basic DB2 database performance commands that you must run to maintain the tradedb database so that you can achieve consistent performance results. After the database is tuned, you back it up so that you can restore it to a consistent state for testing.

__ 1. Stop **TradeServer1** and **TradeServer2**. It is important to stop TradeServer1 and TradeServer2. If these servers are running, some of the DB2 commands (`db2 reorgchk` and `db2rbind`) in the following steps might hang.

__ 2. Double-click the **ssh hostC** icon on your desktop to access a terminal window on **hostC**.

__ 3. Enter the command: **`su - db2inst1`**

__ 4. Update statistics on all tables in the tradedb database by entering the db2 commands ***or*** running the script, do not do both.

__ a. Enter the following db2 commands.

```
db2 connect to tradedb

db2 reorgchk update statistics on table all

db2 connect reset

db2rbind tradedb -l rbind.log all
```

__ b. Or run the script: **`/usr/labfiles/db2scripts/db2updatestats.sh`**

__ c. Wait until you see the messages:

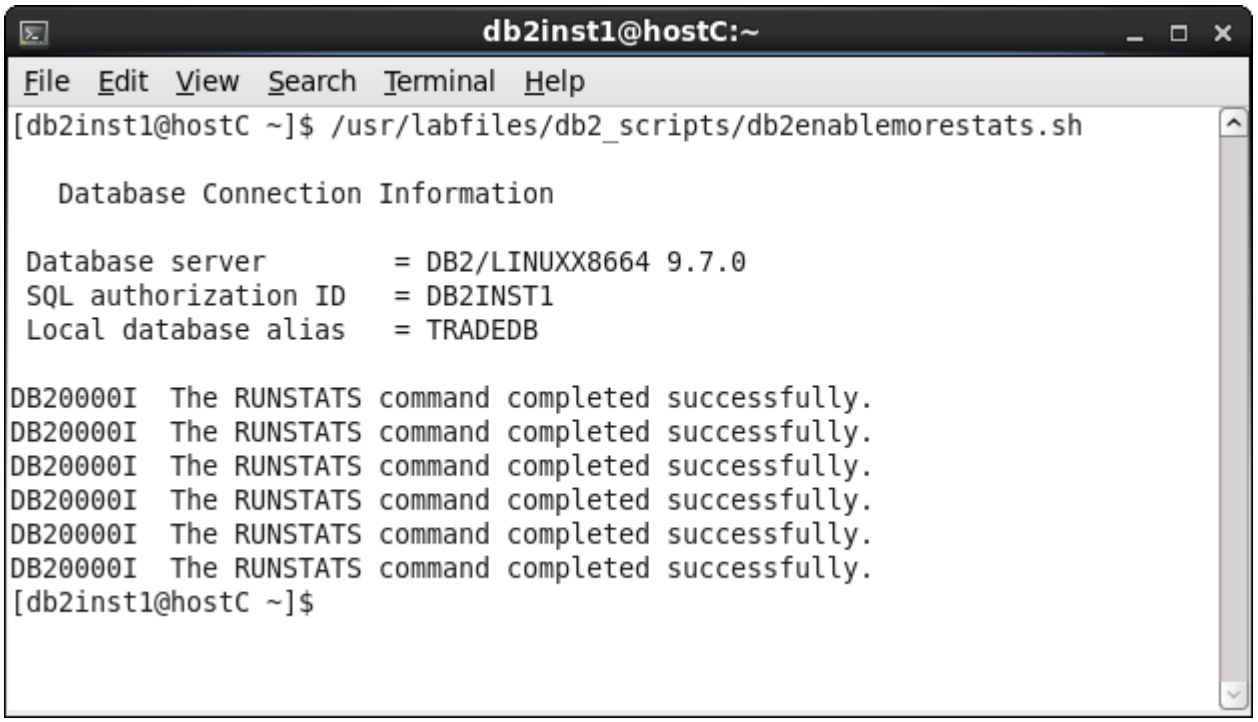
DB20000I The SQL command completed successfully.

Rebind done successfully for database 'TRADEDB'.

__ 5. Gather more statistics on specific database tables.

__ a. Run the script: **/usr/labfiles/db2scripts/db2enablemorestats.sh**

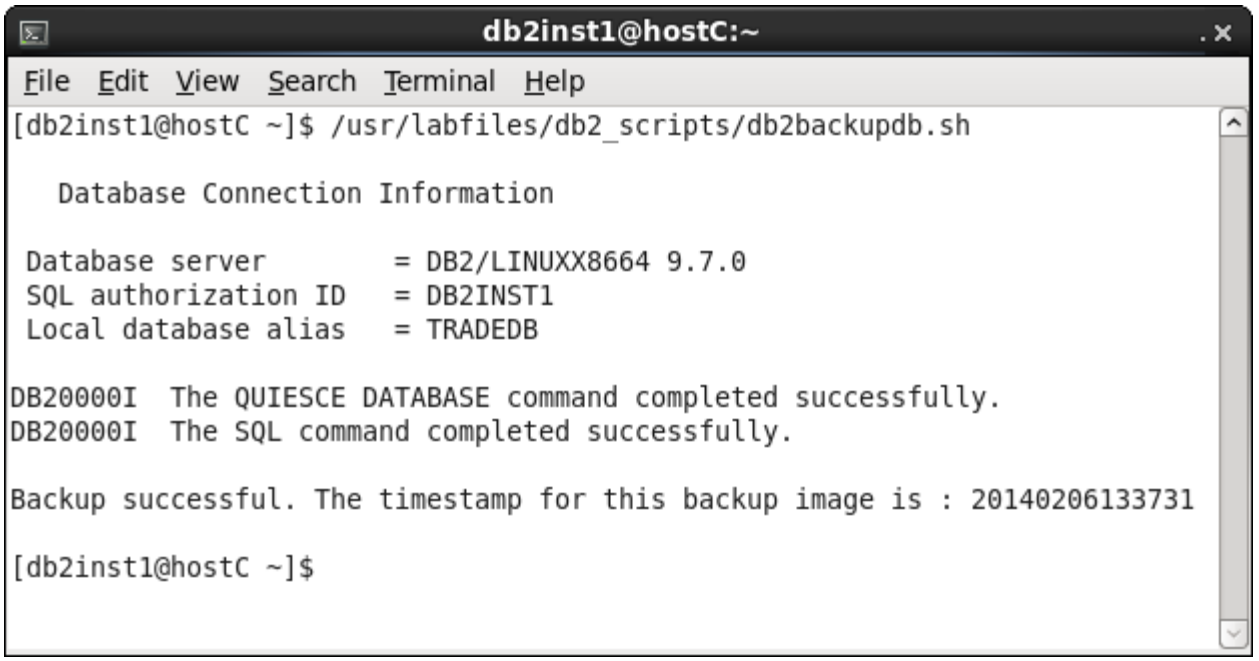
__ b. Wait until you see the messages:



__ 6. Back up the tradedb database.

__ a. Run the script: **/usr/labfiles/db2scripts/db2backupdb.sh**

__ b. Wait until you see similar messages:



__ 7. When the backup completes, you see the message:

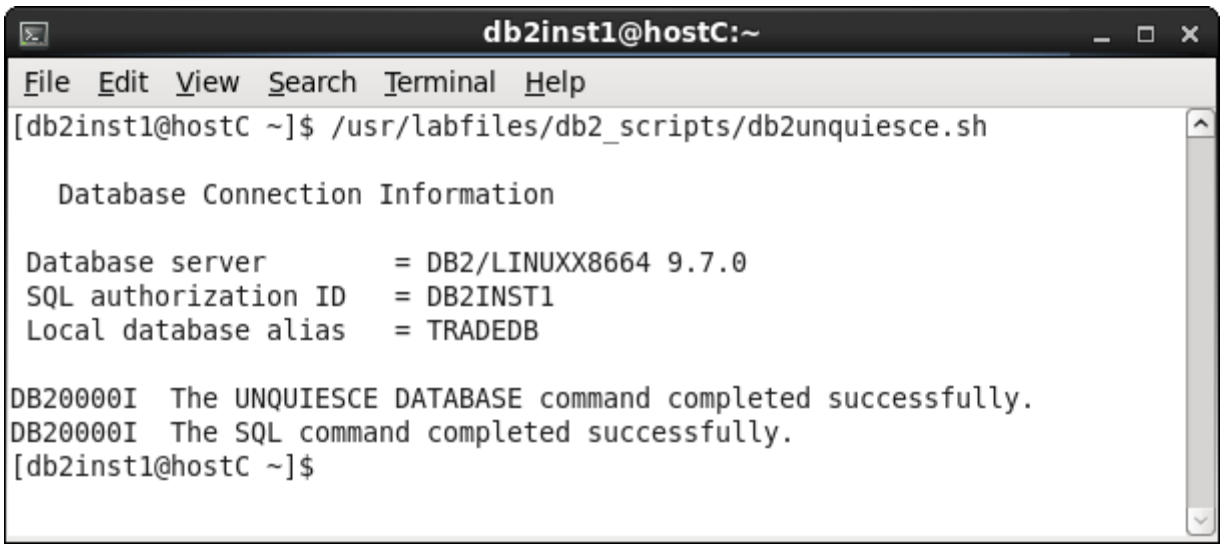
Backup successful. The timestamp for this backup image is: <time_stamp>

Record your time stamp here _____.

__ 8. Unquiesce the database.

__ a. Run the script: **/usr/labfiles/db2scripts/db2unquiesce.sh**

__ b. Wait until you see the messages:

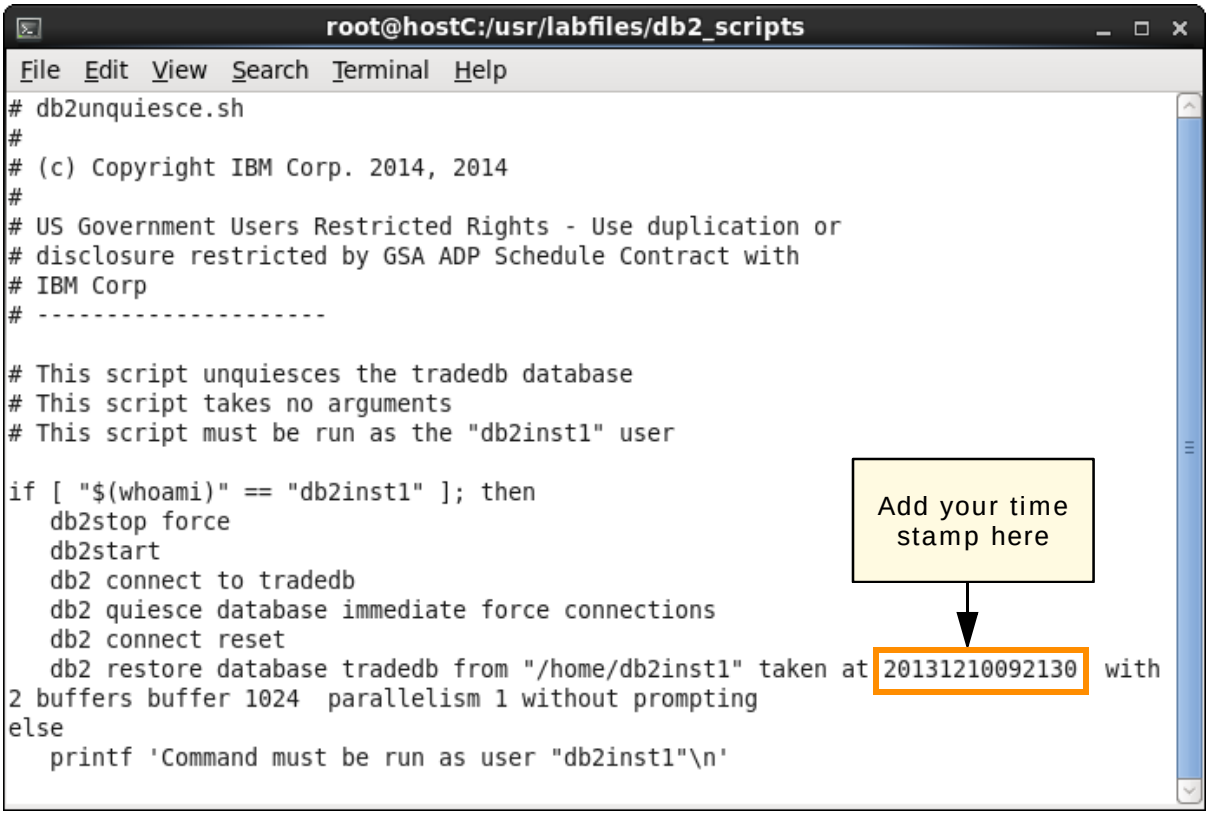


__ 9. Edit the Trade database restore script.

__ a. On **hostC**, use a text editor such as **vi** to open the script, **/usr/labfiles/db2scripts/db2restore.sh**

__ b. Replace the existing time stamp with the value that you recorded in the previous step.

__ c. For example:



__ d. Save the modified script.

__ 10. Start TradeServer1 and TradeServer2.

Section 6: Configure the web server server-status and Apache JMeter test plan

The Status module allows an administrator to find out how well the web server is working. An HTML page is presented that gives the current server statistics in an easily readable form. This page can be configured to automatically refresh.

__ 1. On **hostA**, edit the `httpd.conf` file for the web server to enable server-status.

__ a. Open a terminal window, and enter the command:
`cd /opt/IBM/HTTPServer/conf`

__ b. Open the `httpd.conf` file in an editor by typing the command: **`gedit httpd.conf`**

__ c. When the editor opens, press Ctrl + F, and enter **`server-status`** in the **Search** field.

__ d. Click **Find** until you see the following directive:

```
<IfModule mod_status.c>
<Location /server-status>
    SetHandler server-status
```

```
Order deny,allow
Deny from all
```

__ e. After the last line (**Deny from all**), add the following line: **Allow from 127.0.0.1**

__ f. Save and exit the editor.

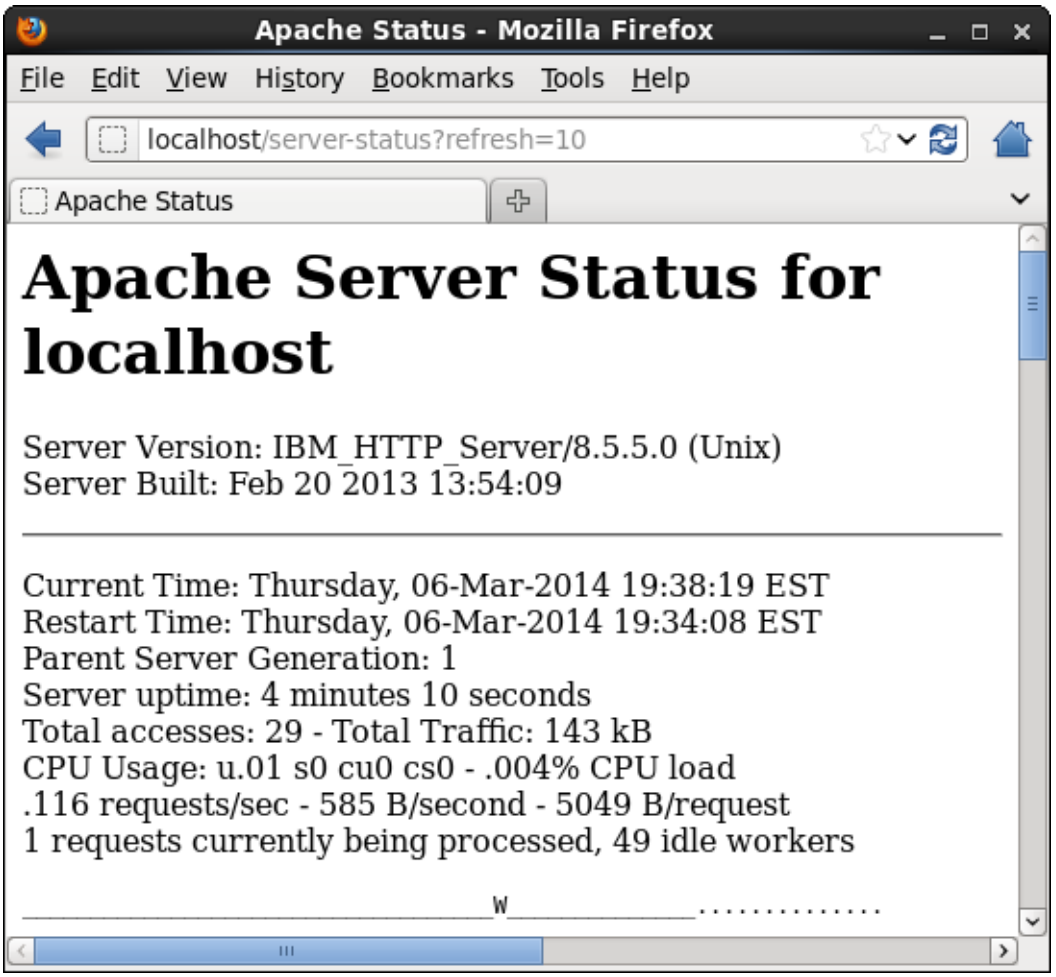
__ 2. Restart the web server.

__ a. In a terminal window on **hostA**, enter the command **cd /opt/IBM/HTTPServer/bin** and enter the following command:

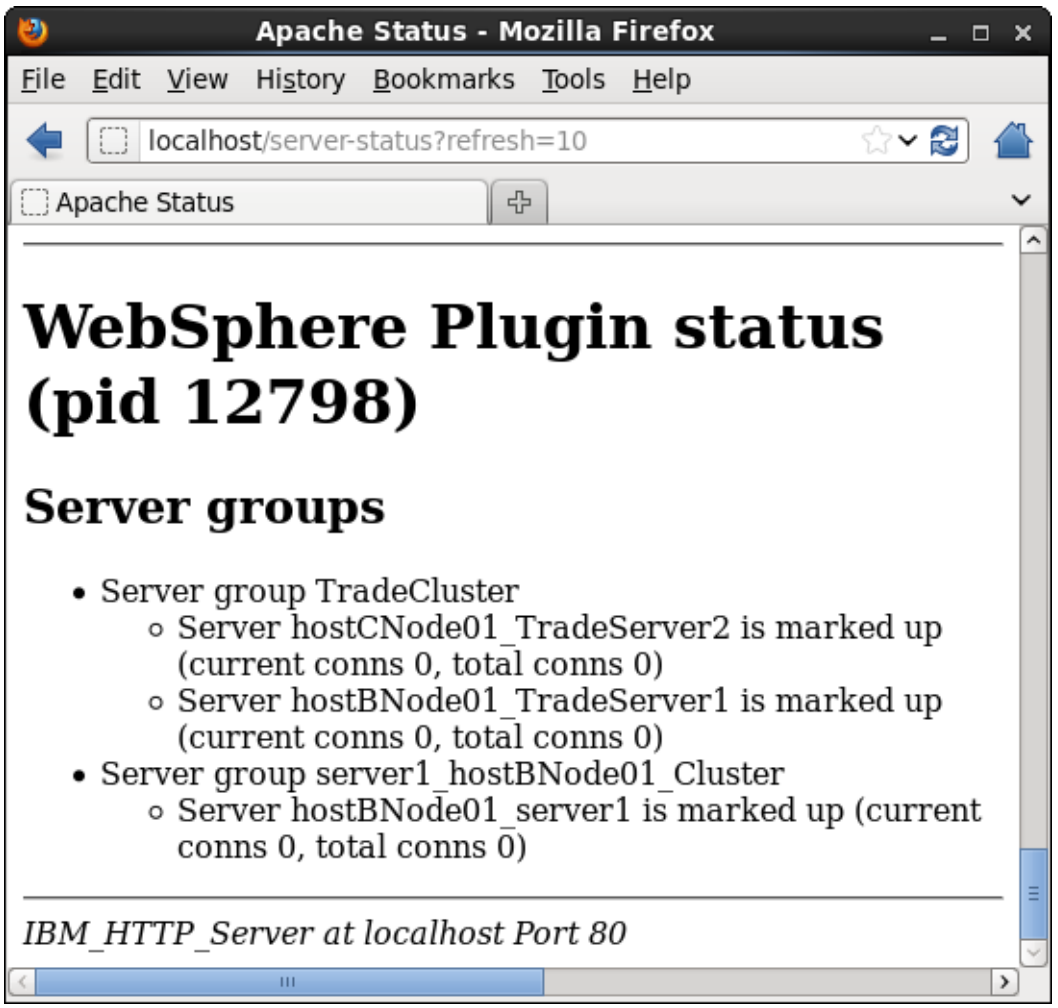
```
./apachectl restart
```

__ 3. Test the server-status tool.

__ a. Open a web browser and enter the web address:
http://localhost/server-status?refresh=10



__ b. Later in the exercise, you can use this tool to monitor requests from the web server. Scroll down to the bottom of the page, and you can see data about the TradeCluster.



__ c. Later in the exercise, you can use this tool to monitor connections to each TradeCluster member. The **`refresh=10`** option directs the web server to update the data every 10 seconds. Keep this browser open for the rest of the exercise, or bookmark this web address.

__ 4. Examine the test plan for the TradeCluster.

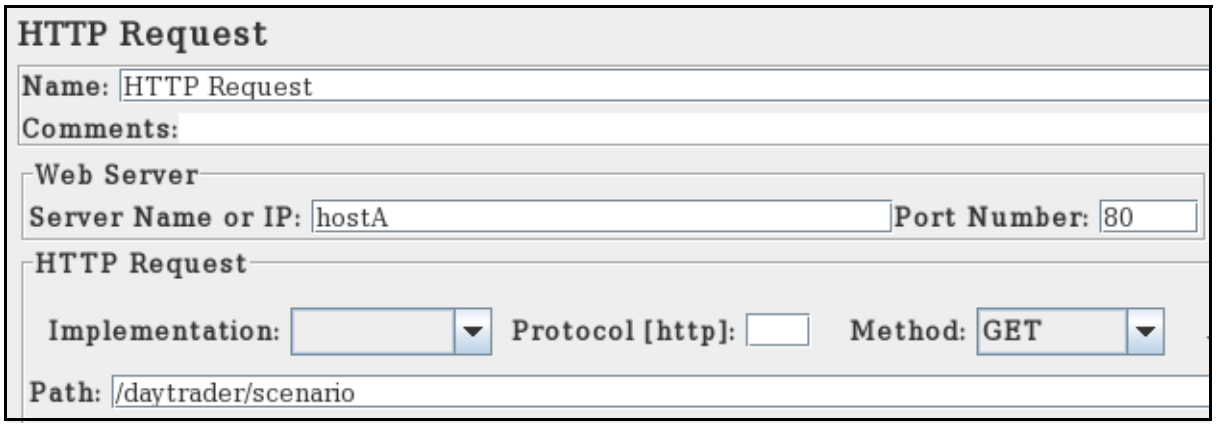
__ a. Start Apache JMeter on hostA by opening a terminal window, and enter the command: **`cd /opt/apache-jmeter/bin`**

__ b. Enter the command: **`./jmeter.sh &`**

__ c. In Apache JMeter, click **File > Open**, and browse for **`/usr/labfiles/Test_Plans`**

__ d. Select **`TradeCluster_test.jmx`**, and click **Open**.

__ e. In the tree, expand Thread Group, and click **HTTP Request**.



__ f. Notice that the web server name points to hostA where IBM HTTP Server is running, and the port is 80. The HTTP plug-in routes the requests to the TradeCluster members by using the information in **`plugin-cfg.xml`** file.

__ 5. View the contents of the **`plugin-cfg.xml`** file.

__ a. On hostA, open a terminal window and enter the command:
`cd /opt/IBM/WebSphere/Plugins/config/webserver1`

__ b. Open the plug-in configuration file with an editor by using the command: **`gedit plugin-cfg.xml`**

__ c. Locate the clone IDs for TradeServer1 and TradeServer2 by searching for “CloneID”. Record the clone IDs here for future reference.

TradeServer1 clone ID: _____

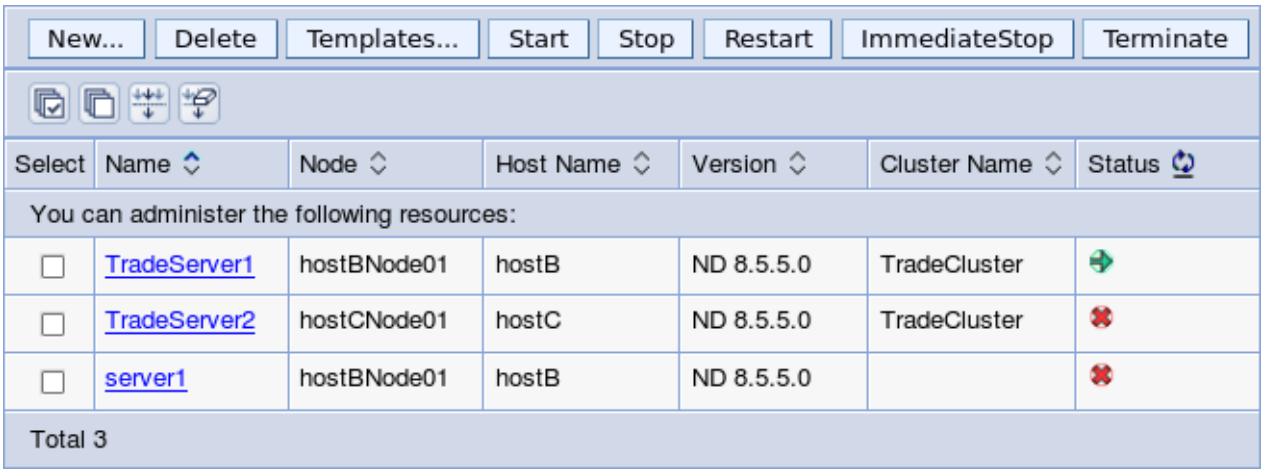
TradeServer2 clone ID: _____

Section 7: Run the load test against a single cluster member

In the section, you run the load test against only TradeServer1, with TraderServer2 stopped. In the next section, you are going to compare performance metrics when both cluster members are running.

__ 1. Stop TradeServer2.

__ a. Log in to the administrative console and stop TradeServer2.



__ 2. Prepare to run the load test.

__ a. In Apache JMeter, click **Summary Report**.

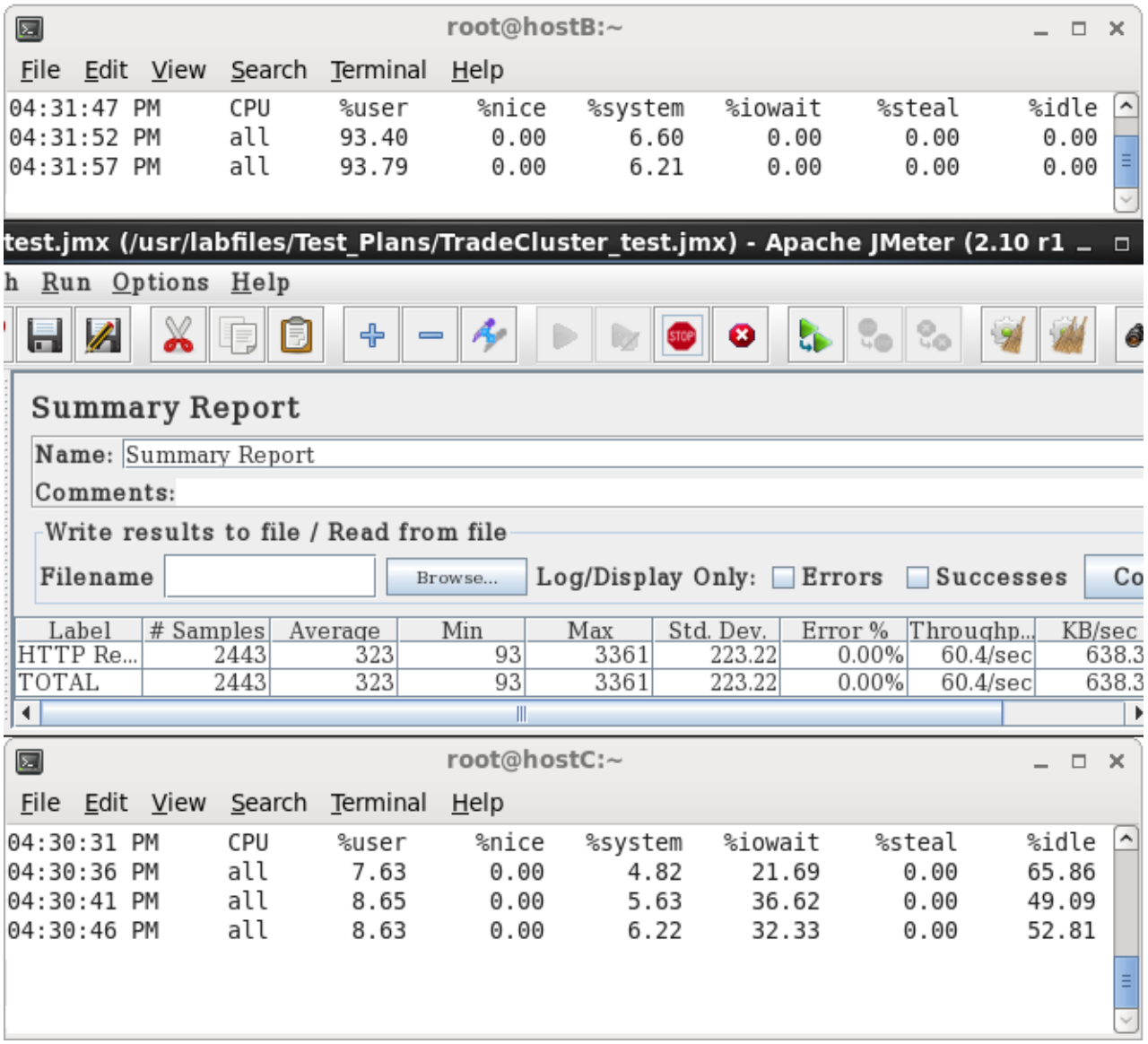
__ b. Open an ssh terminal window on **hostB** and **hostC**. Monitor CPU usage by using the command: **`sar 5`**

__ c. Open a new browser, and enter the web address: **`http://hostA/daytrader`**

__ d. Select the **Configuration** tab, and click the **Reset DayTrader** link.

__ 3. Run the load test.

__ a. Position the terminal windows so that you can see the sar output from hostB and hostC while the test is running and also the JMeter Summary Report view.



__ b. Clear the Summary Report view by clicking the “broom” icon.

__ c. Start the test by clicking **Run > Start** or the green arrow.

__ d. Notice the total CPU usage (**`%user`** + **`%system`**) on both hostB and hostC. HostC should show a lower CPU usage because only DB2 is running on that system. However, you might see a significant value for **`%iowait`** (Percentage of time that the CPU is idle while the system has an outstanding disk I/O request) which also contributes to the total CPU usage.

__ e. The first test run is for warming up the cluster members.

__ f. Remember to reset DayTrader after each test, and run the test 3 more times.

__ g. Run the test three more times and record the performance results (average response time, throughput, and CPU). Remember to reset DayTrader and clear the JMeter Summary Report before each test run.

Table 19: Test with only TradeServer1 running

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% hostB (us+sy)	CPU% hostC (us+sy)
warm-up					

__ 4. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

__ 5. Stop TradeServer1.

Section 8: Run the load test against the TradeCluster

In this section, you run the test load against both cluster members. Initially, the load balance weight is the same (2) for both cluster members, so they share the load equally. Effectively, there are now two CPUs sharing the same load, one on each host. How is the response time and throughput affected in this scenario?

__ 1. Restore the tradedb database by running the following scripts on **hostC**.

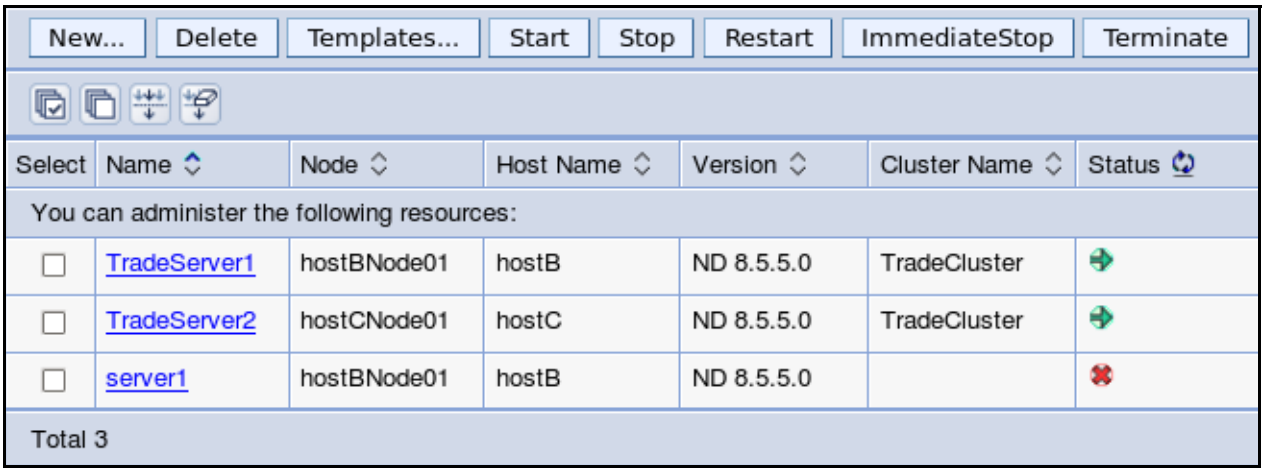
__ a. Run the following script to restore the tradedb database.

```
/usr/labfiles/db2scripts/db2restore.sh
```

__ b. Run the following script to unquiesce the database.

```
/usr/labfiles/db2scripts/db2unquiesce.sh
```

__ 2. Start TradeServer1 and TradeServer2.



__ 3. Prepare to run the load test.

__ a. In Apache JMeter, click **Summary Report**.

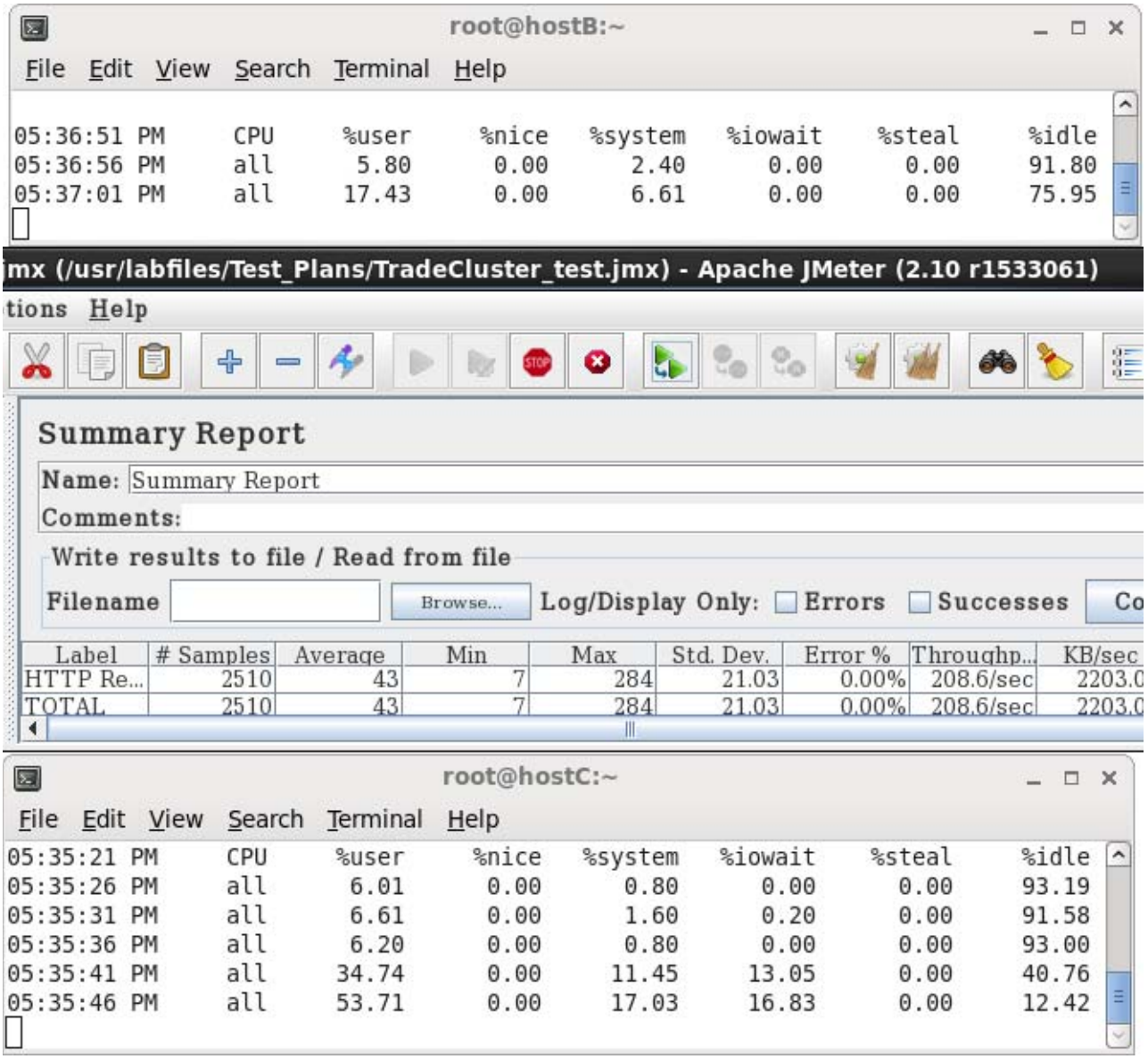
__ b. Open an ssh terminal window on **hostB** and **hostC**. Monitor CPU usage by using the command: **`sar 5`**

__ c. Open a new browser, and enter the web address: **`http://hostA/daytrader`**

__ d. Click the **Configuration** tab, and click the **Reset DayTrader** link.

__ 4. Run the load test.

__ a. Position the terminal windows so that you can see the sar output from hostB and hostC while the test is running and also the JMeter Summary Report view.



__ b. Clear the **Summary Report** view by clicking the “broom” icon.

__ c. Start the test by clicking **Run > Start** or the green arrow.

__ d. Notice the total CPU usage (%user + %system) on both hostB and hostC. HostC should show a higher CPU usage because DB2 is also running on that system.

__ e. The first test run is for warming up the cluster members.

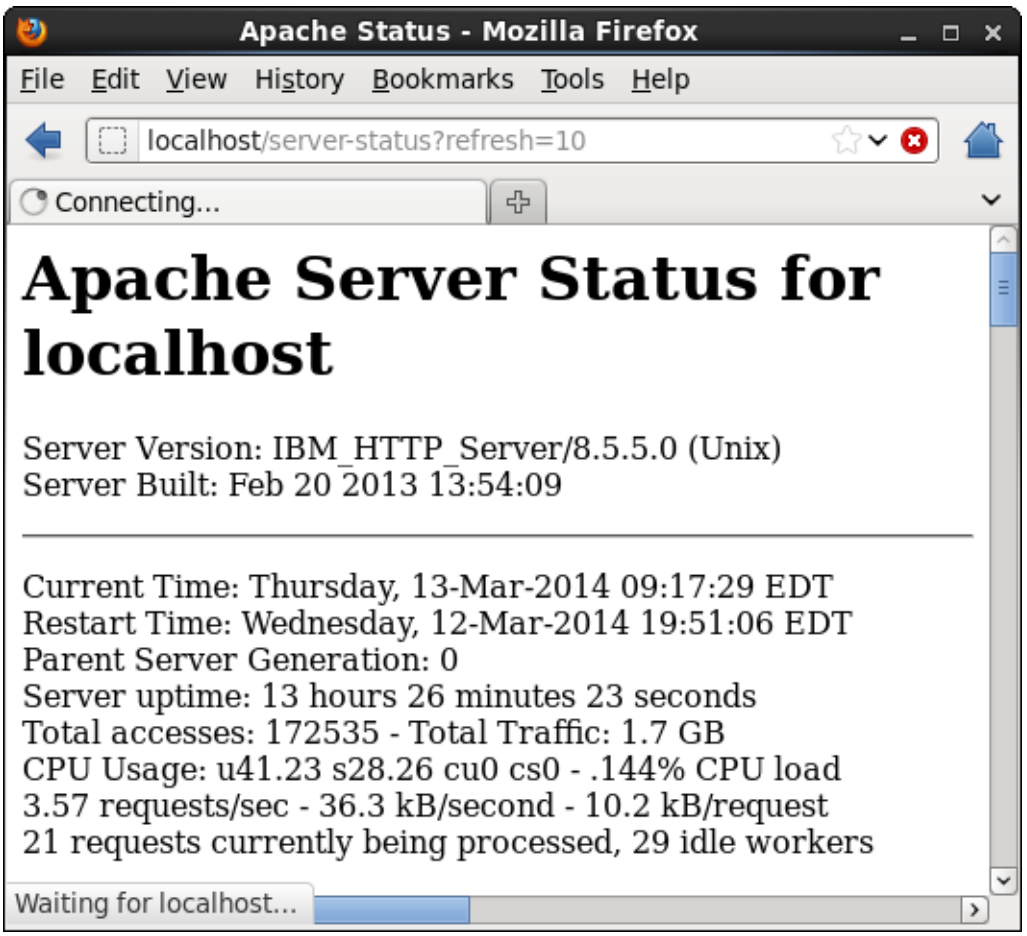
__ f. Remember to reset DayTrader after each test, and run the test 3 more times.

__ g. Run the test three more times and record the performance results (average response time, throughput, and CPU). Remember to reset DayTrader and clear the JMeter Summary Report before each test run.

Table 20: Test with both cluster members started

No. users	Samples	Average response time (ms)	Throughput(req/sec)	CPU% hostB (us+sy)	CPU% hostC (us+sy)
warm-up					

__ h. Examine the Apache server status browser to see the load on the web server.



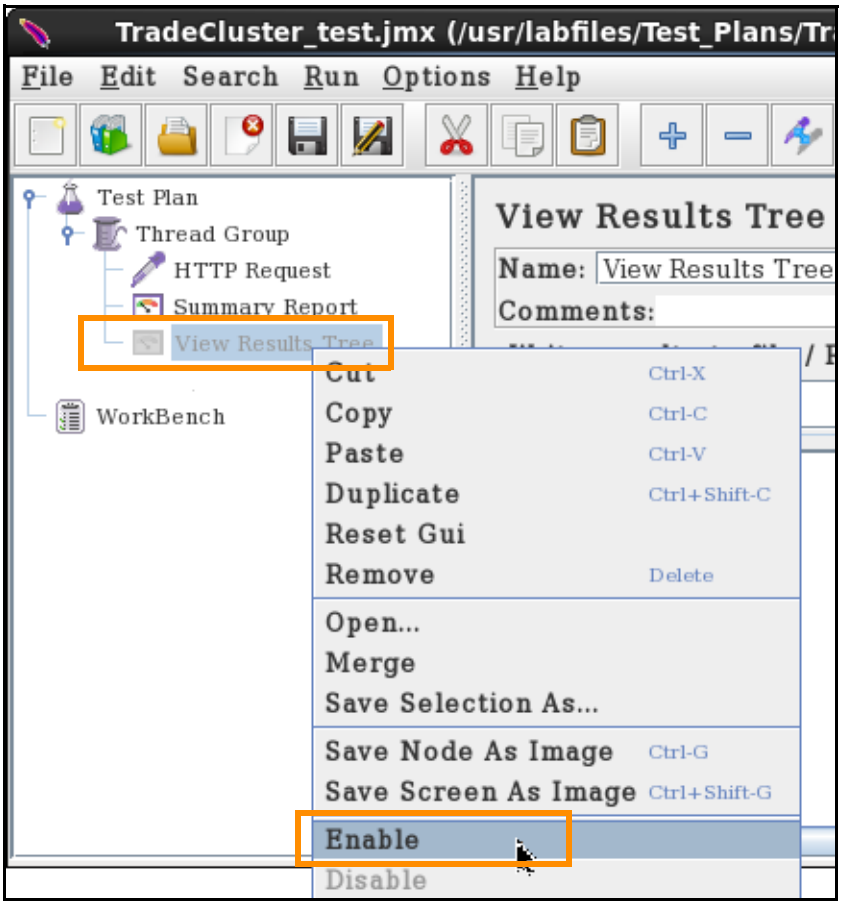
__ 5. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

__ 6. You are likely to see CPU usage on hostB at 28-30%, and CPU usage on hostC consistently at 100%. Because the CPU for hostB is not driven to 100%, try distributing the load so that hostB gets more requests than hostC. You can try this approach in the next section.

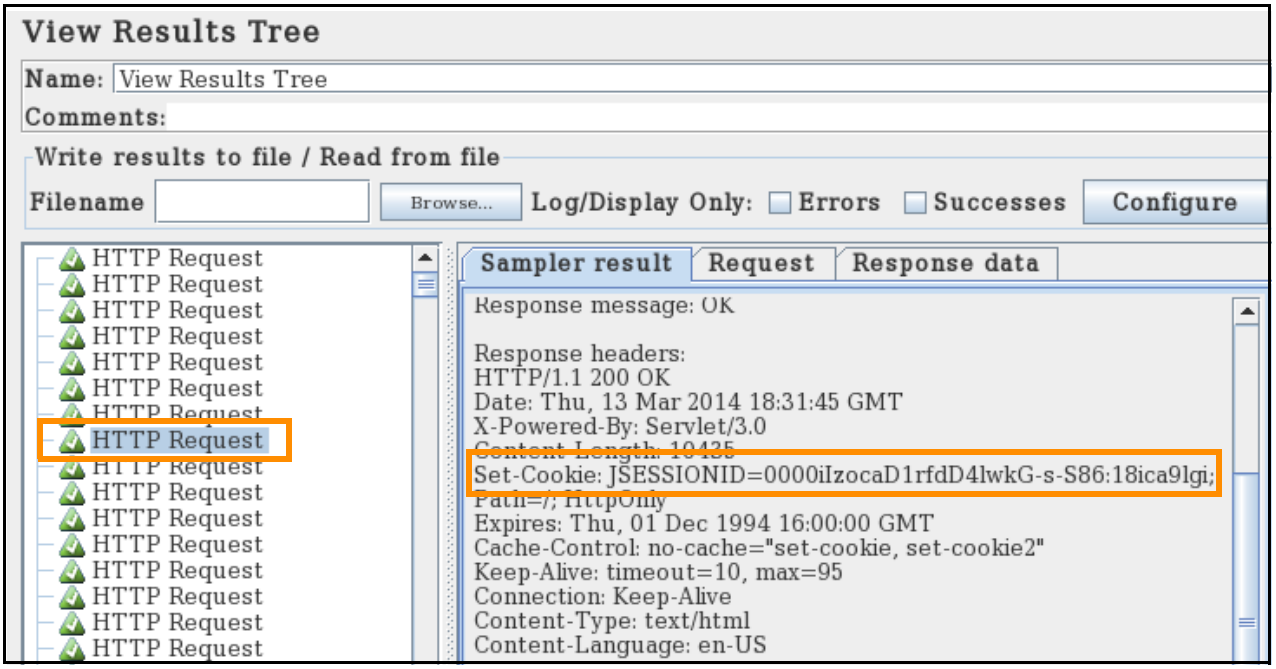
__ 7. Enable the Apache JMeter **View Result Tree** listener and verify plug-in load balancing.

__ a. In Apache JMeter, right-click **View Result Tree** and click **Enable**.



__ b. Click the green arrow to run the test, and look at the data in the View Results Tree pane.

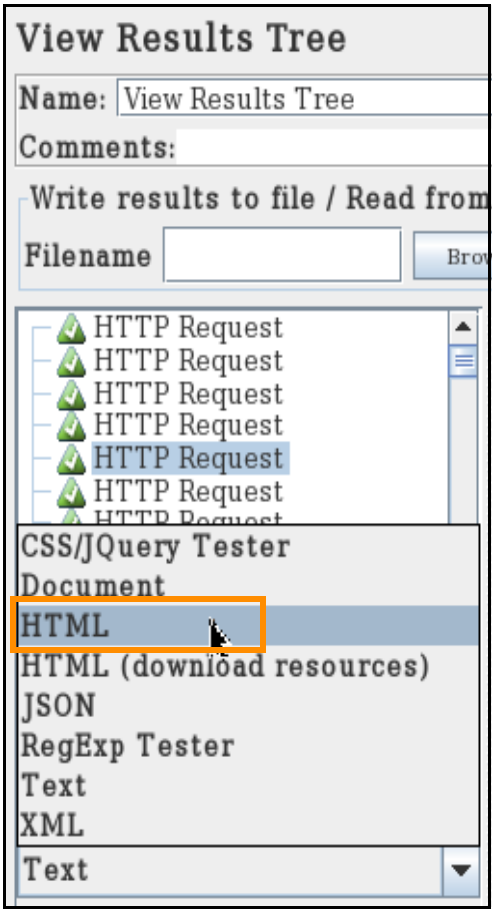
__ c. Click several of the **HTTP Request** entries and view the data in the **Sampler result** tab.



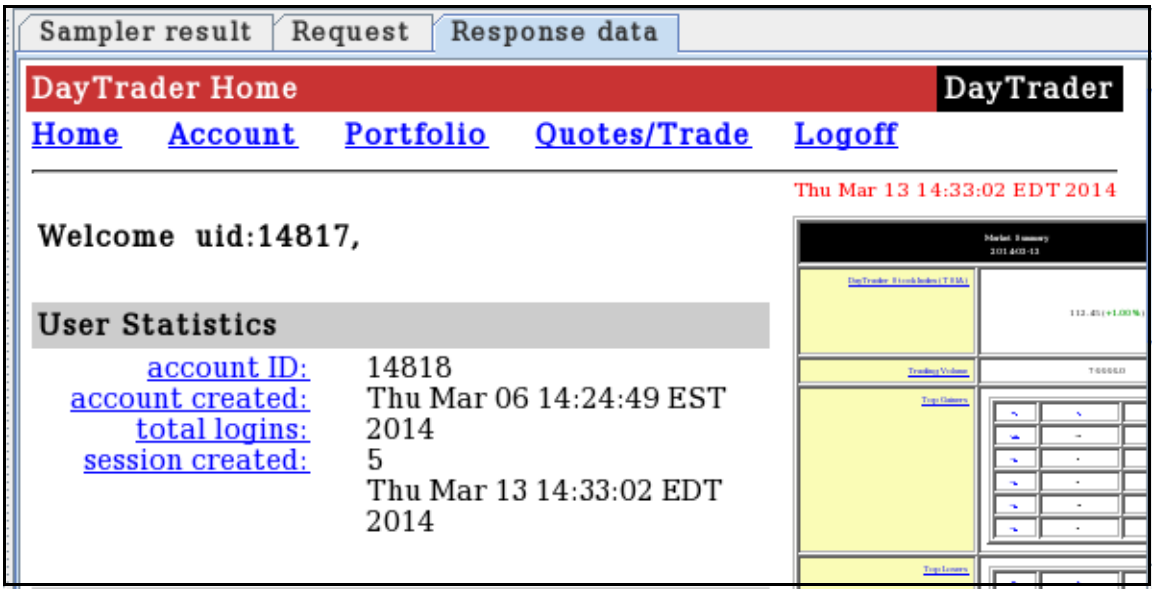
__ d. Notice the JSESSIONID cookie. The last part of the ID contains the clone ID of the server that the request was routed to. The clone ID in this example is **18ica9lgi**. As you look at other requests, you should see the clone ID alternate between the clone IDs for your two cluster members.

__ e. Try clicking the Request and Response tabs to see what data they provide.

__ f. Finally, select **HTML** from the menu at the bottom of the HTTP Request column.



__ g. Select the **Response data** tab, and you can view the HTML rendering of the response.



__ h. The View Results Tree listener is useful for verifying your test requests and responses. However, running the View Results Tree listener has a performance cost. So you should disable it or remove it from your test plan after you verify the test.

__ i. Disable the View Results Tree listener by right-clicking it in the tree, and clicking **Disable**.

Information

WTRN0074E message

`RegisteredSyn E WTRN0074E: Exception caught from before_completion synchronization operation:<openjpa-2.2.2-SNAPSHOT-r422266:1462076 fatal store error> org.apache.openjpa.persistence.OptimisticLockException: The transaction has been rolled back. See the nested exceptions for details on the errors that occurred.`

`FailedObject: org.apache.geronimo.samples.daytrader.AccountDataBean-10262....`

You might see this message in the SystemOut log for TradeServer1 and TradeServer2. This error is most often the result of how the application accesses the database.

Recall that the Test DayTrader Scenario (`/daytrader/scenario`), which you are running against the cluster, selects users at random. If the same user is trying to log in, this exception is thrown. Because it is random, and might occur only once during a test run, the performance testing results are not affected too much. In production, however, you must fix the application code to eliminate these exceptions.

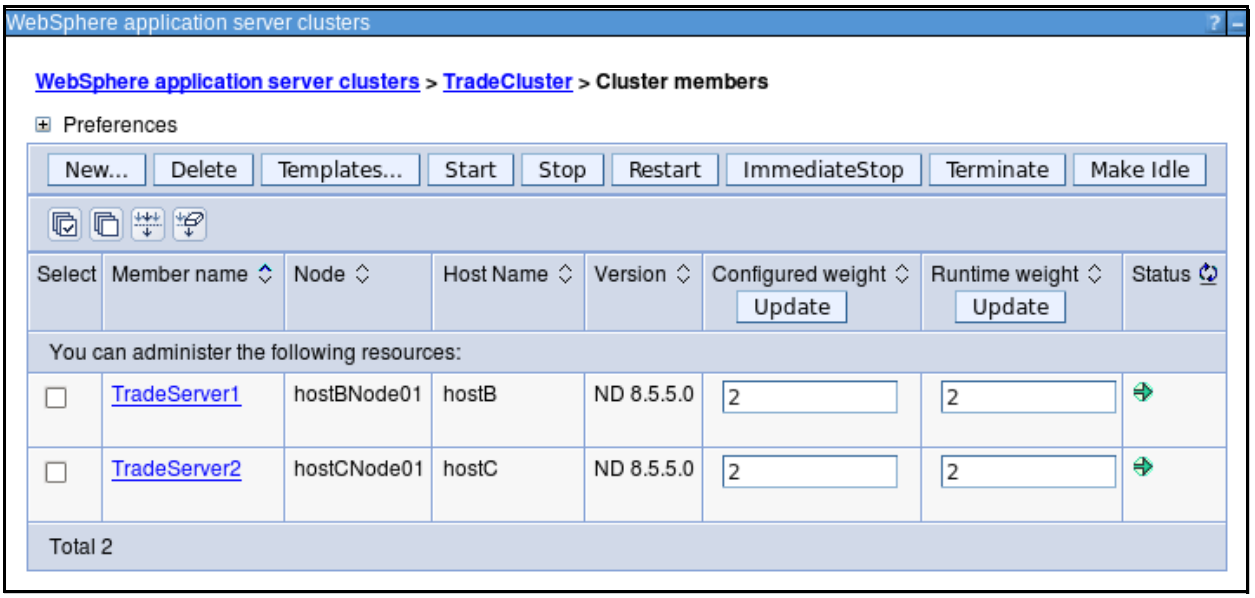
Section 9: Distribute the load between the TradeCluster members

In this section, you adjust the configured load balance weights for the cluster members so that TradeServer1 on hostB gets more requests than TradeServer2 on hostC.

__ 1. Examine the configured weights of the TradeCluster members.

__ a. Log in to the administrative console.

__ b. Click **Servers > Server Types > Clusters > WebSphere application server clusters > TradeCluster > Cluster members**.



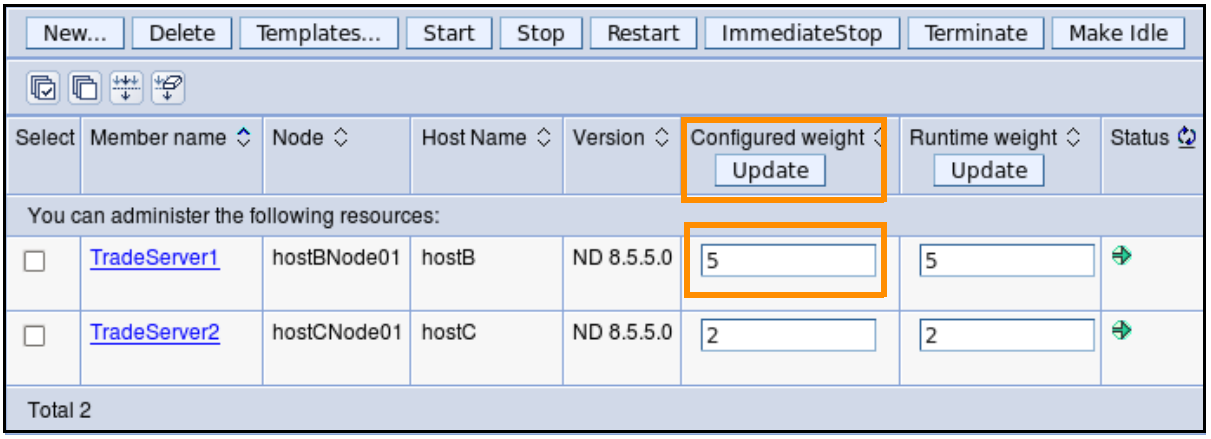
__ c. Notice that the weights for the two cluster members are both equal to 2. Therefore, the HTTP plug-in routes an equal number of requests to each cluster member.

__ 2. Increase the runtime weight for TradeServer1.

__ a. Change the **configured weight** for TradeServer1 to **5** and click **Update**.

__ b. Click **Save**.

__ c. Wait for the nodes to synchronize, and click **OK**.



__ d. With these new weights, TradeServer1 gets roughly 71% of the requests, and TradeServer2 gets 29%.

__ e. Select TradeServer1 and TradeServer2 and click **Stop**.

__ 3. Restore the tradedb database by running the following scripts on **hostC**.

__ a. Run the following script to restore the tradedb database.

```
/usr/labfiles/db2scripts/db2restore.sh
```

__ b. Run the following script to unquiesce the database.

```
/usr/labfiles/db2scripts/db2unquiesce.sh
```

__ 4. Start TradeServer1 and TradeServer2.

Important

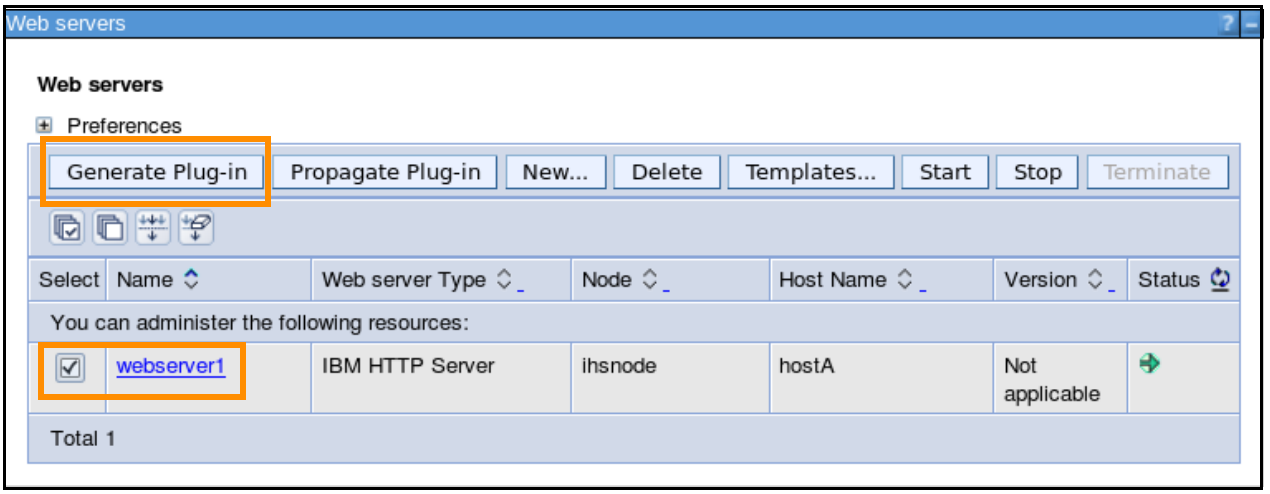
Regenerate and propagate the HTTP plug-in

The configured weight for TradeServer1 was changed. Therefore, it is necessary to generate a new plug-in configuration file and propagate it to the web server. The plug-in needs the new weight so that it can route requests correctly.

__ 5. Regenerate and propagate the HTTP plug-in. Verify that the weight for TradeServer1 is 5 and the weight for TradeServer2 is 2, by examining the **`plugin-cfg.xml`** file. Look for the following lines.

__ a. Log in to the administrative console, and click **Servers > Server Types > Web servers**.

__ b. Select **webserver1** and click **Generate Plug-in**.

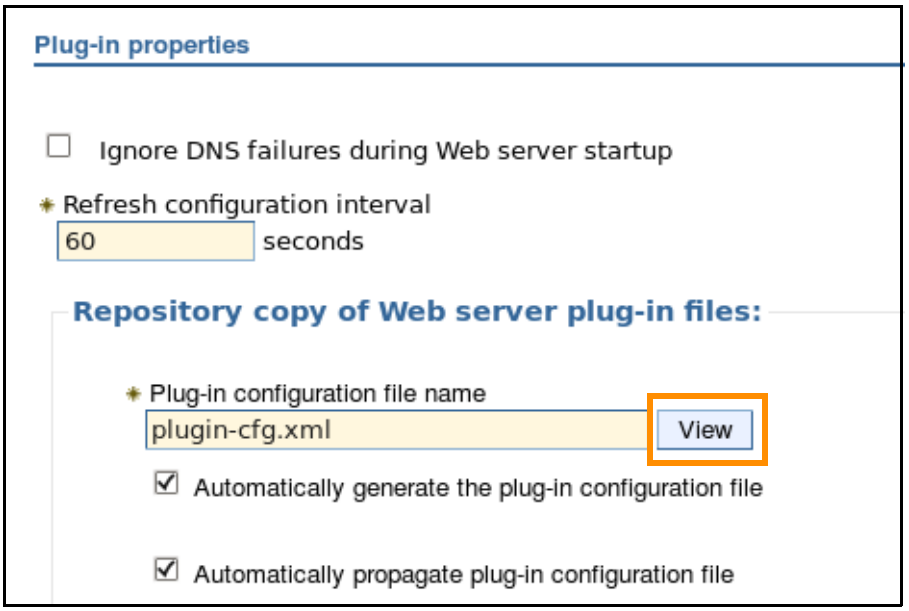


__ c. When the plug-in configuration file generation is complete, click **Propagate Plug-in**.

__ d. When the propagation of the plug-in configuration file is complete, click **webserver1**.

__ e. On the configuration tab, click the **Plug-in properties** link.

__ f. Click **View** to examine the plug-in configuration.



__ g. Look for the load balance weights for each cluster member.

<Server CloneID="18ica9lgi" ConnectTimeout="5" ExtendedHandshake="false" **LoadBalanceWeight="5"** MaxConnections="-1" **Name="hostBNode01_TradeServer1"** ServerIOTimeout="900" WaitForContinue="false">

<Server CloneID="18ica9q8g" ConnectTimeout="5" ExtendedHandshake="false" **LoadBalanceWeight="2"** MaxConnections="-1" **Name="hostCNode01_TradeServer2"** ServerIOTimeout="900" WaitForContinue="false">

__ h. Click **OK**, and log out of the administrative console.

__ 6. Run the load test again.

__ a. In Apache JMeter, clear the Summary Report view by clicking the “broom” icon.

__ b. Start the test by clicking the green arrow.

__ c. Notice the total CPU usage (%user + %system) on both hostB and hostC. HostB should show a higher CPU usage when compared to the first test because it is handling more requests.

__ d. The first test run is for warming up the cluster members.

__ e. Remember to reset DayTrader after each test, and run the test 3 more times.

__ f. Run the test three more times and record the performance results (average response time, throughput, and CPU). Remember to reset DayTrader and clear the JMeter Summary Report before each test run.

Table 21: Test with higher weight for TradeServer1

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% hostB (us+sy)	CPU% hostC (us+sy)
warm-up					

__ 7. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

__ 8. Compare your results for the two configurations.

__ a. Was more CPU used on hostB when TradeServer1 was configured with the higher weight?

__ b. Which configuration showed the faster average response time?

__ c. Which configuration showed the higher throughput?

Section 10:Optional: Increase the maximum heap size for the cluster members

Another property to tune is the maximum heap sizes for each cluster member. If you have time, do this next section.

__ 1. Change the maximum heap size for TradeServer1 to **512 MB**.

__ 2. Stop TradeServer1 and TradeServer2.

__ 3. Restore the tradedb database on **hostC**.

__ 4. Run the load tests again and record the performance results (average response time, throughput, and CPU).

Table 22: Test with -Xmx=512m for TradeServer1

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% hostB (us+sy)	CPU% hostC (us+sy)
warm-up					

__ 5. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

__ 6. Compare your results with the previous tests.

__ 7. Change the maximum heap size for both TradeServer1 and TradeServer2 to **512** MB.

__ 8. Stop TradeServer1 and TradeServer2.

__ 9. Restore the tradedb database on **hostC**.

__ 10. Run the load tests again and record the performance results (average response time, throughput, and CPU).

Table 23: Test with -Xmx=512m for TradeServer1 and TradeServer2

No. users	Samples	Average response time (ms)	Throughput (req/sec)	CPU% hostB (us+sy)	CPU% hostC (us+sy)
warm-up					

__ 11. Excluding the warm-up run, calculate the average response time and throughput for the three runs.

- Response time _____(ms)
- Throughput _____(req/sec)

__ 12. Compare your results with the previous tests.

End of exercise

Exercise review and wrap-up

In this exercise, you used a Jython script to install the DayTrader Application to a cluster. Then, you configured the IBM HTTP plug-in to route requests to the cluster. You used Apache JMeter to load test the cluster. While load-testing the cluster, you used the Apache server-status tool to monitor web server usage, and system tools to monitor CPU usage on cluster member hosts. Finally, you reconfigured cluster member weights to make better use of system resources, and verified the performance with Apache JMeter.

Appendix A. Load test results tables

This appendix contains extra tables if you need them.

Table 24: Ramp up test

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)	Little's Law resp x thru
warm-up						

.

Table 25: Verify baseline test

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)	Little's Law resp x thru
warm-up						

Test results for:______________________________

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

Test Results for:______________________________

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

Test Results for:______________________________

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

Test Results for:______________________________

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

Test Results for:____

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

Test Results for:____

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

Test Results for:____

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

Test Results for:____

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

Test Results for:___________________________________

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

Test Results for:___________________________________

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

Test Results for:___________________________________

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up					

Test Results for:___________________________________

No. users	Samples	Response time (ms)	Throughput (req/sec)	CPU% application server (us+sy)	CPU% DB2 server (us+sy)
warm-up