

# Elastic Memory for Java

VMware vFabric Cloud Application Platform 5.0

Elastic Memory for Java 1.0

VMware vFabric tc Server 2.6

This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see <http://www.vmware.com/support/pubs>.

EN-000748-00

**vmware**<sup>®</sup>

---

You can find the most up-to-date technical documentation on the VMware Web site at: <https://www.vmware.com/support>.

The VMware Web site also provides the latest product updates.

If you have comments about this documentation, submit your feedback to: [docfeedback@vmware.com](mailto:docfeedback@vmware.com)

Copyright © 2012 VMware, Inc. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at <http://www.vmware.com/download/patents.html>.

VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

VMware, Inc., 3401 Hillview Avenue, Palo Alto, CA 94304

[www.vmware.com](http://www.vmware.com)

---

---

# Table of Contents

1. Elastic Memory for Java .....	1
Intended Audience .....	1
2. What is Elastic Memory for Java? .....	3
How ESXi and EM4J Improve Memory Utilization .....	3
How EM4J Affects Memory and Application Performance .....	4
3. Setting Up Elastic Memory for Java .....	7
Platform Support .....	7
EM4J Considerations and Limitations .....	7
Enable EM4J in the Virtual Machine .....	7
Create and Start an EM4J-Enabled tc Runtime Instance .....	10
Configure an EM4J-Enabled tc Runtime Instance .....	11
4. Using Elastic Memory for Java with Apache Tomcat .....	13
5. Configuring Java Applications for VMware .....	15
Configuring and Sizing a Virtual Machine for Java Applications .....	15
Over-Committing Memory with EM4J .....	15
Distributing Multiple tc Runtime Instances Among Virtual Machines .....	16
6. Monitoring and Managing EM4J .....	17
Monitoring Memory with vSphere Client .....	17
Monitoring Memory with vSphere Web Client .....	18
Monitoring Memory with the EM4J MBeans .....	28
EM4J and DRS or vMotion .....	37
Virtual Machine Suspend and Resume .....	38



# 1. Elastic Memory for Java

*Elastic Memory for Java* describes how to set up, monitor, and manage Elastic Memory for Java, the VMware® memory management technology that improves memory utilization when executing Java workloads on VMware vSphere® ESXi™ virtual machines.

## Intended Audience

*Elastic Memory for Java* is intended for anyone who wants to configure and administer Elastic Memory for Java with vFabric tc Server™ runtime instances executing on VMware ESXi™ virtual machines.



## 2. What is Elastic Memory for Java?

The Java virtual machine (JVM) manages memory automatically for applications, storing objects in heap memory pre-allocated to it by the operating system, and freeing the memory when objects are no longer needed. However, it takes a lazy approach, allowing unreferenced objects to remain in heap memory until it is necessary to free up space to store additional objects. This lazy approach to memory management works fine when Java is running directly on physical hardware. However, it can lead to performance problems if you are trying to leverage certain key benefits of virtualization. For Java applications running on ESXi that require consistent, fast response times, the traditional best practice advice has been to reserve 100% of the virtual machine memory and to make the Java heap size as small as possible to avoid wasting memory. This limitation made Java unable to benefit as effectively as other workloads from ESXi's industry-leading memory reclamation technology.

Elastic Memory for Java (EM4J) manages a memory balloon that sits directly in the Java heap and works with new memory reclamation capabilities introduced in ESXi 5.0. EM4J works with the hypervisor to communicate system-wide memory pressure directly into the Java heap, forcing Java to clean up proactively and return memory at the most appropriate times—when it is least active. You no longer have to be so conservative with your heap sizing because unused heap memory is no longer wasted on uncollected garbage objects. And you no longer have to give Java 100% of the memory that it needs; EM4J ensures that memory is used more efficiently, without risking sudden and unpredictable performance problems.

EM4J is an add-on to vFabric tc Server. With EM4J and tc Server, you can run more Java applications on your ESXi servers with predictable performance, squeezing the most value out of your hardware investment.

### How ESXi and EM4J Improve Memory Utilization

A chief benefit of virtualization is the ability to maximize physical host resources by adding virtual machines to consume unused computing capacity. Recouping unused CPU cycles lowers the costs of hardware, service, management, maintenance, and energy. Available physical memory can limit the number of virtual machines you can deploy on a host, even when CPU is still underutilized. To get the most out of your hardware, you can either add more physical memory or use the existing memory more efficiently.

ESXi ensures the efficient use of physical memory by employing shared memory pages, ballooning, memory compression and, as a last resort, disk swapping. These techniques allow ESXi to reduce the amount of physical memory consumed by the virtual machines. Disk swapping, where memory pages are written to and from disk, is least efficient because disk I/O is much slower than memory I/O. With Java, disk swapping is especially disruptive because when garbage collection occurs, page swapping activity can spike, causing performance degradations and failures.

#### Transparent Page Sharing

ESXi constantly scans memory pages in virtual machines for duplicate contents and can reduce memory overhead by eliminating the duplication. The indirection between virtual memory pages in the virtual machines and physical memory pages on the host make this possible. Duplicate pages are mapped to the same physical page, and if a virtual page is modified, a unique new copy is created transparently. For example, initialized memory pages can be shared if they are written with zeros and static code in virtual machines running the same OS and applications. When a virtual machine writes to a shared page, it is given a unique copy, which breaks the sharing for that page. The pages reclaimed by transparent page sharing can significantly improve your consolidation ratio.

#### Memory Compression

ESXi employs memory compression to avoid swapping when memory becomes tight. To free memory, ESXi attempts to compress memory pages and save them in a compression cache. Compressing and decompressing uses CPU cycles, but is still far more efficient than disk I/O. Memory compression is the last opportunity to reclaim memory before disk swapping.

#### VMware Tools Balloon

There is an important separation of concerns between ESXi and the virtual machines running on it; the memory management techniques employed within a virtual machine are completely opaque to the hypervisor. The hypervisor therefore has no knowledge of how memory allocated to a virtual machine is being used and when it swaps memory to disk or compresses memory, it could swap out memory that is still in use. A far more efficient mechanism of reclaiming memory from a virtual

machine is the VMware Tools balloon driver, which runs as a process within the virtual machine, allocates and pins unused memory, and communicates the pages back to the hypervisor. The memory owned by the balloon driver can then be temporarily de-coupled from the virtual machine and used elsewhere.

ESXi gives the balloon driver a target size, and the balloon driver attempts to fulfill the request. Under the control of the ESXi hypervisor, balloons in each host virtual machine expand or shrink depending on the shifting requirements of the virtual machines. ESXi calculates balloon targets based on virtual machine activity; a less active virtual machine gets a higher balloon target and the reclaimed memory moves to the more active virtual machines.

A balloon in the guest operating system forces the OS to use memory more conservatively. Operating systems use every bit of available memory to improve system performance. However, caching inactive data in one virtual machine while another virtual machine has scarce memory for immediate needs is poor use of memory in a virtualization environment. Expanding the balloon in a virtual machine constrains the OS from using memory for low priority storage and allows the hypervisor to distribute physical memory resources where they are most needed.

## EM4J Balloon

When a Java application starts, the operating system allocates memory to the JVM. The allocated memory is then managed by the JVM. On a virtual machine deployed to serve an enterprise Java application, the Java heap typically occupies the greatest portion of allocated memory. The separation of concerns between the operating system and the JVM is very similar to that between the ESXi hypervisor and virtual machines. The JVM manages its object heap as a single block of memory which is entirely opaque to the OS. When Java objects become garbage, there is no way that the OS can reclaim this memory, and if the OS cannot reclaim the memory, then neither can the VMware tools balloon. For this reason, in order for the hypervisor to reclaim memory from a virtual machine in which the Java heap occupies the majority of the available memory, a balloon operating within the Java heap can be much more efficient.

When EM4J is enabled in the virtual machine and a JVM executing in the virtual machine, it serves ESXi hypervisor ballooning requests from the JVM heap instead of from the OS memory pool. Just as the guest balloon allocates memory from the OS memory pool, EM4J allocates memory from the Java heap by creating and storing a few large Java objects. To avoid fragmentation, EM4J does not pin memory in the Java heap. The ESXi hypervisor sets the target size of the balloon, and EM4J attempts to satisfy the request.

Like the VMware tools guest balloon, the EM4J balloon encourages the JVM to use the available memory more conservatively. When the EM4J balloon inflates, it can force the JVM to clean up garbage and return memory to the hypervisor, typically during periods of low activity. When the JVM needs the memory, typically during an increase in activity, the next garbage collection clears out some of the balloon and that memory is once again available to the JVM. This is the "elastic" nature of EM4J memory management; the amount of ballooned heap memory grows or shrinks according to the relative requirements of all of the virtual machines on the host.

A welcome side effect of EM4J is that you can size the Java heap less conservatively without wasting memory. Traditionally, to determine the optimum size of the Java heap, you determined the application's peak requirement. Allocating less could mean unacceptable performance, errors, or crashes; allocating more potentially wastes memory. With EM4J you can size the heap to comfortably accommodate the peak requirement, and excess memory is ballooned away and redistributed where it is needed.

## How EM4J Affects Memory and Application Performance

EM4J is tuned to work with long-running Web applications, where the application serves client requests and response times are the critical performance metric. If EM4J is enabled and the host is not over-committed, there is no cost to running EM4J. When you enable EM4J and begin to over-commit memory, client response times should be similar to running your application with fully reserved memory, the difference imperceptible to your users. As the host memory pressure increases, response times may increase gradually due to an increase in GC frequency. Balloons inflate first in the least active virtual machines where the increased GC is less likely to be disruptive.

The effect on performance varies proportionally with how much medium to long-lived data is created by the workload. If your application just responds to Web requests and avoids storing long-lived state in the heap, there should be little effect on performance from EM4J ballooning. The effect increases when you save long-lived state in the heap and the tenured generation grows, reducing the portion of the heap available for ballooning and forcing more frequent GC.

In contrast, the inability of the VMware Tools balloon to reclaim memory from Java has the potential to cause sudden, unpredictable, and significant drops in performance. EM4J helps the system behave gracefully and predictably when memory becomes scarce. It helps you to more easily determine the over-commit ratio that provides acceptable performance at peak loads.



## 3. Setting Up Elastic Memory for Java

Elastic Memory for Java (EM4J) is included with vFabric tc Server Standard Edition. You enable EM4J in the ESXi virtual machine by adding a parameter to the virtual machine configuration and then use the tc Server `elastic-memory` template to create an EM4J-enabled tc Runtime instance.

EM4J works with all existing garbage collection policies and heap configurations and requires no special JVM tuning or configuration.

### Subtopics

[EM4J Platform Support](#)

[EM4J Considerations and Limitations](#)

[Enable EM4J in the Virtual Machine](#)

[Create and Start an EM4J-Enabled tc Runtime Instance](#)

[Configure an EM4J-Enabled tc Runtime Instance](#)

### Platform Support

The following table lists supported platforms for this release of EM4J.

Table 3.1. EM4J Platform Support

Operating System	Version	Architecture	JVM
Red Hat Enterprise Linux/CentOS	v5	x32	HotSpot 1.6
Red Hat Enterprise Linux/CentOS	v5	x64	HotSpot 1.6

### EM4J Considerations and Limitations

When you enable EM4J in a virtual machine, memory outside of the Java heap cannot be reclaimed. Therefore, EM4J should only be used in configurations where the majority of memory is used by Java heap.

EM4J does not function in heap sizes smaller than 512MB.

EM4J currently can reclaim a maximum of 2GB from a Java heap.

EM4J works equally well in virtual machines with large pages enabled as with small pages.

You can run more than one EM4J-enabled tc Runtime instance in a virtual machine. Each instance contributes to the overall balloon for the virtual machine. However, the individual instances do not coordinate ballooning with each other, so running large numbers of instances is not currently recommended. A pragmatic limit is four EM4J-enabled tc Runtime instances.

### Enable EM4J in the Virtual Machine

After you create a VM on ESXi, you enable EM4J in the VM by adding a parameter to the VM's configuration. You can add a second parameter to limit the size of the EM4J balloon, but it is unnecessary. The following table describes the parameters.

Table 3.2. EM4J VM Configuration Parameters

Name	Description
<code>sched.mem.pshare.guestHintsSyncEnable</code>	When set to <code>true</code> , EM4J ballooning is enabled and guest ballooning is disabled. When set to <code>false</code> , guest ballooning is enabled and EM4J ballooning is disabled. This parameter is <i>required</i> to enable EM4J.

Name	Description
sched.mem.maxmemctl	Sets a limit for the total balloon size for the VM, in megabytes. If set to -1, the default, no limit, is set and ESXi determines the balloon size. If set to 0, ballooning is effectively disabled. It is generally unnecessary to set this parameter. The parameter affects both EM4J and VMware Tools balloons. If you set this parameter and later change the VM's configuration, this parameter will not change dynamically with the other changes.

When EM4J ballooning is enabled, the guest balloon is disabled. If you enable EM4J ballooning, but do not run an EM4J-enabled JVM, there will be no ballooning for the VM, which could lead to the ESXi host swapping pages from the VM. This situation is undesirable because it can be expensive to swap the pages back in again. If you shut down a JVM with an EM4J balloon, the ballooned memory remains until it is written over. This means that the risk of swapping occurring after a JVM shutdown is at least initially very low.

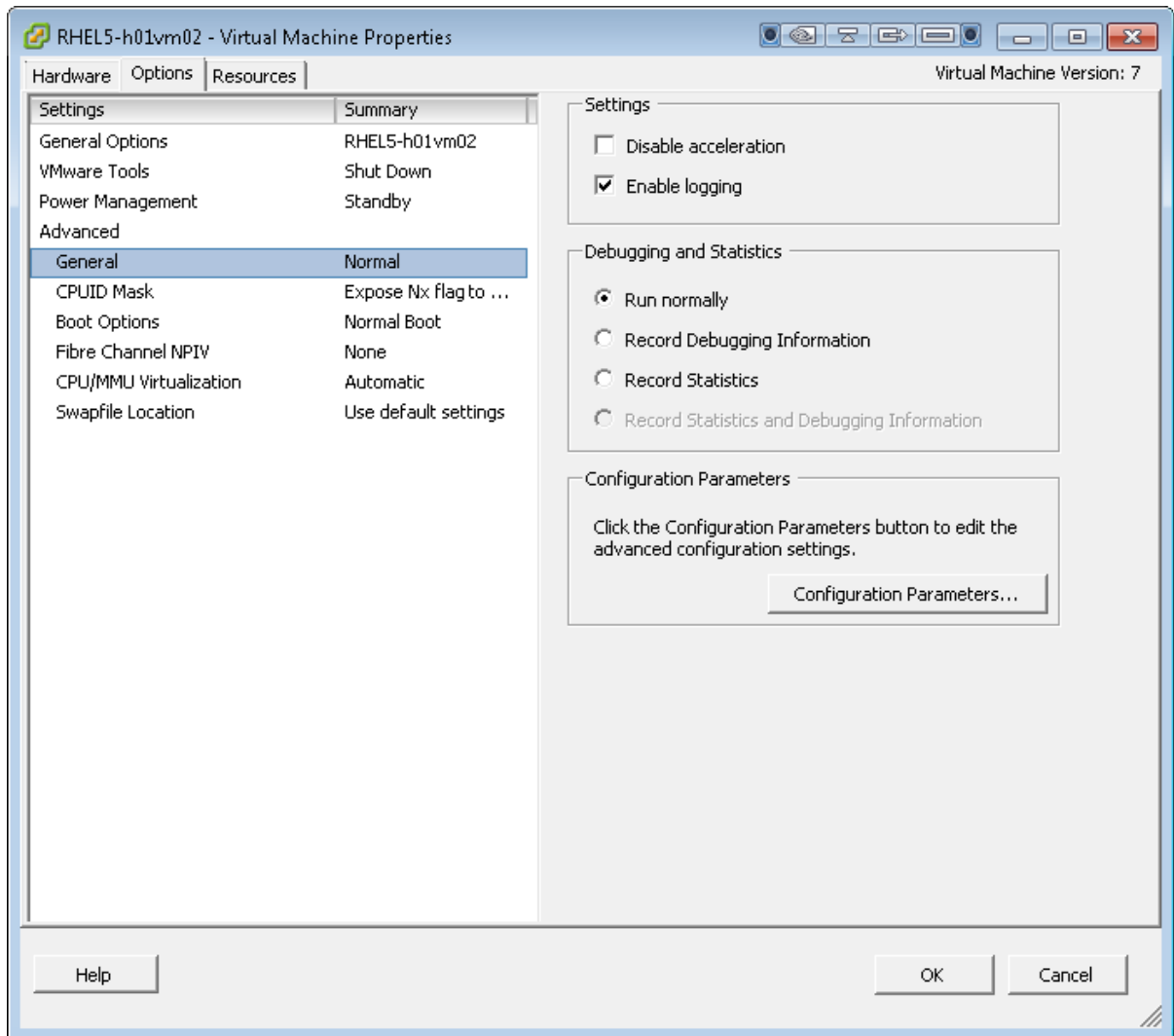
The procedure describes how to add the EM4J configuration parameters using the vSphere Client. You can also add the parameters by editing the VM's VMX text file. After you enable EM4J, cloning the VM will duplicate the setup.

## Prerequisites

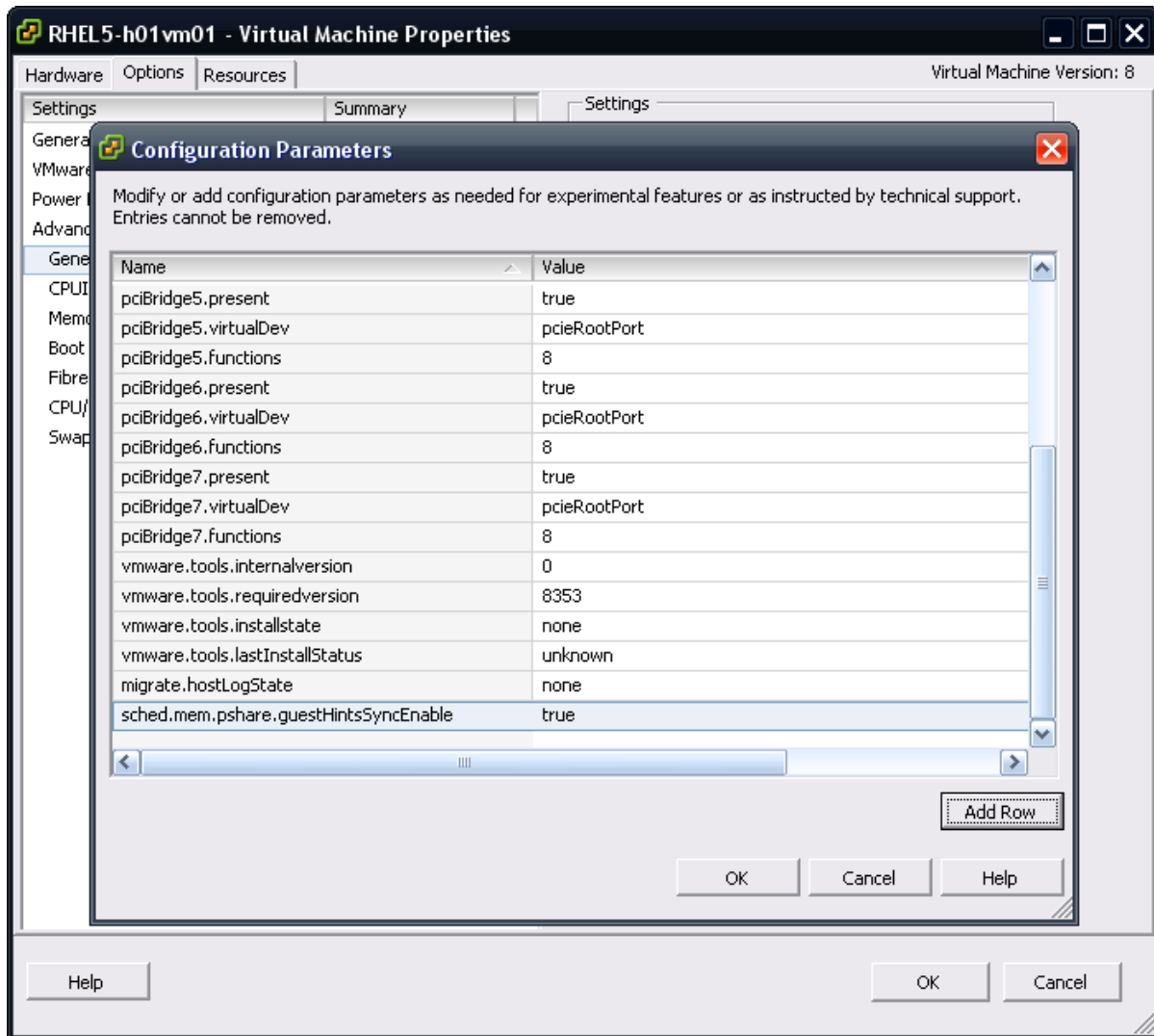
- Create at least one virtual machine. EM4J requires VMware ESXi 5.0 or higher. See [Creating a Virtual Machine](#).

## Procedure

1. Shut down the guest OS and, using vSphere Client, power off the VM, if necessary. It is not possible to edit the configuration parameters while the VM is running.
2. In the left panel of the vSphere Client, expand the ESXi host so that the VM you want to configure is visible.
3. Right-click the VM and select **Edit Settings...** from the pop-up menu.
4. Click the **Options** tab.
5. In the **Settings** column, under **Advanced**, select **General**. See the following figure.



6. In the **Configuration Parameters** box on the right, click **Configuration Parameters...**
7. Click **Add Row** and enter the following parameter, as shown in the following figure.
  - **Name:** `sched.mem.pshare.guestHintsSyncEnable`
  - **Value:** `true`



8. (Optional) Click **Add Row** and enter the following parameter.
  - **Name:** `sched.mem.maxmemctl`
  - **Value:** `-1` (or enter the maximum number of megabytes to limit the size of the EM4J balloon)
9. Click **OK** twice.
10. Repeat this procedure for each VM that will run an EM4J-enabled Java application.

## Create and Start an EM4J-Enabled tc Runtime Instance

You create an EM4J-enabled tc Runtime instance with the `tcruntime-instance` command, specifying the `elastic-memory` template. This is no different than using any other tc Server template, but note these caveats:

- The elastic-memory template is only useful in an ESXi virtual machine with a [supported guest OS and JVM](#), where it is part of the memory management strategy for the virtualization environment.
- The path to the tc Runtime instance (`CATALINA_BASE`) must not contain spaces.

## Prerequisites

- Enable EM4J. See [Enable EM4J in the VM](#).
- Install supported guest operating systems and a JDK or JVM on the VMs. See [Platform Support](#).
- Install VMware Tools on the guest operating systems. See [Installing and Upgrading VMware Tools](#).
- Install tc Server Standard Edition 2.6. This tc Server release includes the `elastic-memory` template, which you use to create EM4J-enabled tc Server instances. See "Installing vFabric tc Server" in *Getting Started with vFabric tc Server*.

## Procedure

1. Change to the tc Server installation directory and create an EM4J-enabled tc Runtime instance.

For example:

```
prompt$ ./tcruntime-instance.sh create instanceName -t elastic-memory
```

Replace `instanceName` with the name of the runtime instance. You can use additional templates (`-t` option) or use any of the other features of the `tcruntime-instance` command described in "tcruntime-instance.sh Reference" in *Getting Started with vFabric tc Server*.

2. Start the instance using the `tcruntime-ctl.sh` command.

For example:

```
prompt$ ./tcruntime-ctl.sh instanceName start
```

3. To verify that EM4J started successfully, look in `CATALINA_HOME/logs/catalina.out` for the message `EM4J 1.0.x agent initialized`.

## What to do next

- Configure the tc Runtime instance. See [Configure an EM4J-Enabled tc Runtime Instance](#).
- For more information about running tc Server, see "Starting and Stopping tc Runtime Instances" in *Getting Started with vFabric tc Server*.
- To get recommendations for using EM4J with ESXi and tc Server, see [Configuring Java Applications for VMware](#).

## Configure an EM4J-Enabled tc Runtime Instance

The tc Server `elastic-memory` template installs a Java library (`balloon.jar`) and a native library (`libballoon.so`). It sets up logging for EM4J and modifies the Java command line in the following ways:

- Sets maximum heap size to 1GB (`-Xmx1024M`)
- Sets the `java.agent` system property to `CATALINA_BASE/lib/balloon.jar`
- Sets the `agent.path.unix.x86` system property to `CATALINA_BASE/bin/x86-linux/libballoon.so`
- Sets the `agent.path.unix.x64` system property to `CATALINA_BASE/bin/amd64-linux/libballoon.so`

To modify these values, edit the `CATALINA_BASE/bin/setenv.sh` file, which constructs the `JAVA_OPTS` environment variable passed to the Java command line when the tc Runtime instance starts up.

The `elastic-memory` template sets the logging level for EM4J to the INFO level by default. You can edit `CATALINA_BASE/conf/login.properties` to set a different logging level.



## 4. Using Elastic Memory for Java with Apache Tomcat

In addition to vFabric tc Server 2.6.x, EM4J can be used when running Apache Tomcat 6.0 or Apache Tomcat 7.0 on ESXi.

For information on the requirements for using EM4J with Apache Tomcat, see the VMware knowledge base article at <http://kb.vmware.com/kb/2011243>.

For instructions on setting up EM4J with Apache Tomcat, see the VMware knowledge base article at <http://kb.vmware.com/kb/2011244>.



## 5. Configuring Java Applications for VMware

The [Enterprise Java Applications on VMware Best Practices Guide](#) is an essential guide for anyone deploying enterprise Java applications on VMware. The guide covers architecture, vCPU, memory, timekeeping, scalability, and other configuration and sizing issues. It contains best practices recommendations and offers help with troubleshooting Java memory and thread contention issues. The *Best Practices Guide* should be your primary resource for virtualizing Java applications. Using EM4J affects only recommendations concerning JVM memory sizing and memory reservations. This topic describes specific recommendations for using EM4J with ESXi and tc Server.

For more background on ESXi memory management, see [Memory Virtualization Basics](#) and [Administering Memory Resources](#) in the vSphere 5 Documentation Center.

### Subtopics

[Configuring and Sizing a Virtual Machine for Java Applications](#)

[Overcommitting Memory With EM4J](#)

[Distributing Multiple tc Runtime Instances Among Virtual Machines](#)

### Configuring and Sizing a Virtual Machine for Java Applications

To prevent ESXi or the OS from swapping, the *Best Practices Guide* recommends that you size VM memory to accommodate the OS and JVM. The formula presented is:

$$VM\text{Memory (needed)} = \text{GuestOSMemory} + \text{JVMMemory}$$

$$JVMMemory = \text{MaxHeap} + \text{PermGen} + \text{NumberOfConcurrentThreads} * \text{StackSize}$$

The amount consumed by the guest OS memory in this formula includes memory used by the OS and all applications in the VM except the JVM. You can also include in *GuestOSMemory* the non-heap memory used by the JVM, such as the JIT code cache and byte code interpreter.

*JVMMemory* includes the following separate memory areas:

- *Heap* memory, which contains the EM4J balloon, specified with the `-Xmx` option.
- *PermGen*, which contains class level code, specified with the `-XX:MaxPermSize` option (HotSpot only)
- *Stack memory*, which contains a stack for each concurrent thread. The size of a single stack is set with the `-Xss` option, so you need to multiply the stack size by the expected number of concurrent threads to calculate the total stack memory.

The *Best Practices Guide* recommends that you set a reservation for the full amount of VM memory, because if JVM memory is not resident during garbage collection, the swapping can be a serious performance problem. With EM4J, setting a reservation is unnecessary. The recommendation to size VM memory according to the above formula remains valid, but you do not need to reserve the memory. You can also oversize the VM and heap without wasting memory, because excess is ballooned away.

### Over-Committing Memory with EM4J

When you run Java applications on ESXi *without* EM4J, it is important to completely reserve VM memory to avoid thrashing during garbage collection. When EM4J is enabled and ESXi needs to reclaim memory, the JVM is placed under pressure to manage memory conservatively. Memory in the Java heap not used for application objects can be reclaimed by ESXi for use by other VMs. This means that you do not have to reserve VM memory with EM4J. You can create more VMs, over-committing the physical memory, and improving your consolidation ratio.

The amount you can safely over-commit, of course, varies depending on application behavior, loads, and other factors. Experiment by gradually increasing the over-commit and measuring your applications' response times. Response times will only

increase when memory pressure becomes significant, increasing garbage collection. With lower levels of over-commit, response times may not be affected at all. EM4J has been tested with memory over-committed up to 40% with good results.

One way to evaluate the memory pressure on the VM is to monitor the difference (delta) between the balloon target and balloon size. If the balloon size is smaller than the target for a significant time, the memory pressure on the VM is too high.

Ultimately, your physical memory must accommodate the maximum active data set for all VMs to avoid performance degradation. Ballooning can only reclaim unused memory. If your applications have large amounts of long-lived live data, your over-commit ratio will be smaller. As an optimization, the JVM moves objects in heap that survive multiple GCs to the tenured generation. As the size of the tenured area of the heap grows, the potential EM4J balloon size shrinks. You can observe this happening by monitoring the difference between the Balloon Target and Balloon size in the vSphere Client, or by monitoring the EM4J MBeans via JMX. In particular, the `com.springsource.balloon.jmx.JvmBalloonState` MBean shows the size of the tenured generation in the JVM. If the tenured area of the JVM grows to consume a large portion of the heap, you will have to reduce your over-commit.

## Distributing Multiple tc Runtime Instances Among Virtual Machines

You can run multiple EM4J-enabled JVMs in a virtual machine. ESXi sets one balloon target for the VM, and that target is distributed among the JVMs running on the VM. There is no coordination of balloons in the separate JVM heaps, so a pragmatic recommendation is to limit the number of EM4J-enabled JVMs running on a VM to approximately four. JVMs that are not using EM4J, for example agents or transient utilities, are not an issue.

When you co-locate tc Runtime instances on VMs, consider how the size and load of the various servers will influence the ESXi balloon targets for the VMs. The algorithm ESXi uses to set balloon targets tends to provide more memory to more active VMs than to less active VMs. It increases balloon targets on less active VMs so it can shift the reclaimed memory to more active VMs. When you assign application servers to VMs, look for opportunities to take advantage of shifting workloads. It is better to co-locate servers that become active at similar times in a VM than to group servers with contrary activity levels in a VM. When all of the servers in a VM become more active simultaneously, ESXi will shift more memory to that VM relative to less active VMs. If instead you distribute servers among VMs so that the activity level in each VM remains relatively constant, the memory allocation will be more static and less able to take advantage of shifting loads.

## 6. Monitoring and Managing EM4J

EM4J works with the ESXi hypervisor to make memory in the Java heap space available to other VMs that need it. Once EM4J is enabled, as described in [Enable EM4J in the VM](#), no additional configuration is required. When memory pressure increases on the host and it has recovered as much as it can from techniques such as page sharing, ESXi initiates ballooning in VMs that are configured for it. You can monitor memory utilization, including ballooning, using vSphere Client or other tools such as `esxtop`. In addition, more specific information about EM4J ballooning is available by querying EM4J MBeans through JMX.

### Subtopics

[Monitoring Memory with vSphere Client](#)

[Monitoring Memory with vSphere Web Client](#)

[Monitoring Memory with the EM4J MBeans](#)

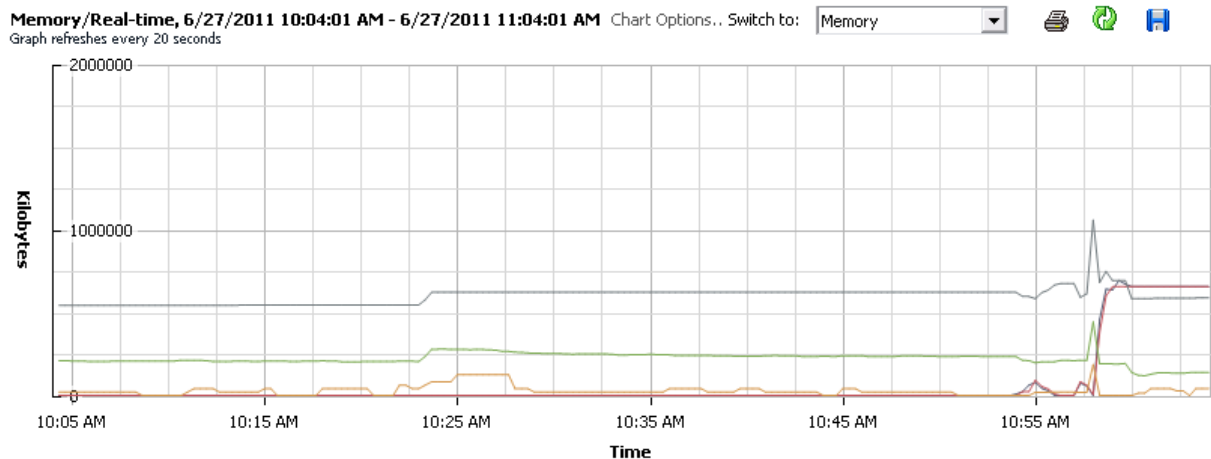
[EM4J and DRS or vMotion](#)

[Virtual Machine Suspend and Resume](#)






### Monitoring Memory with vSphere Client

Ballooning is one of several strategies ESXi employs to use host memory efficiently. Using vSphere Client, you can observe ESXi memory management as the different techniques are applied and memory is recovered from the VMs. Charts on the **Performance** tab let you view the memory metrics over time. You can see how memory utilization responds to different loads and events and how you might arrange and configure VMs to more fully utilize the physical resources and maintain desired performance.

You can select an individual VM, as illustrated by the following example, a Resource Pool, or the entire host. Choose the **Performance** tab and select **Memory** from the **Switch to** drop-down list. If necessary, click **Chart Options...** and select **Balloon** and **Balloon target** in the **Counters** section.



#### Performance Chart Legend

Key	Object	Measurement	Rollup	Units	Latest	Maximum	Minimum	Average
	appvm08	Balloon target	Average	Kilobytes	659844	697840	0	67130.489
	appvm08	Balloon	Average	Kilobytes	659844	659844	0	66370.356
	appvm08	Active	Average	Kilobytes	43116	188204	0	29161.4
	appvm08	Granted	Average	Kilobytes	591556	1064000	546752	603356.84
	appvm08	Consumed	Average	Kilobytes	140248	446544	120340	224679.42

The balloon metrics in the vSphere Client do not distinguish between the VMware tools balloon and the EM4J balloon, but the balloon in any given VM must be one type or the other. When EM4J is enabled in a VM, the Balloon and Balloon target metrics

for that VM describe the current and target sizes of the EM4J balloon. When EM4J is not enabled, these metrics instead describe the VMware tools balloon driver. When you view a group of VMs, the numbers from both types of balloons are summed.

## Monitoring Memory with vSphere Web Client

An EM4J plug-in is available for the vSphere Web Client. The plug-in adds a Workloads tab that displays statistics for Java workloads on the virtual machine selected in the vSphere Web Client inventory tree. You can quickly verify that the virtual machine and JVM are configured correctly for EM4J and see detailed information about the JVM process and memory usage. Alerts warn of EM4J configuration problems and suggest best practices.

### Setting Up the vSphere Web Client for EM4J

To monitor EM4J-enabled JVMs in the vSphere Web Client, the following components must be present and properly configured:

- vCenter Server. See [Installing vCenter Server](#).
- vSphere Web Client. See [Install and Start the vSphere Web Client](#).
- vSphere Web Client EM4J plug-in. See [Download and Install the vSphere Web Client EM4J Plug-in](#).
- Virtual machines, VM Version 7 or higher, with VMware Tools installed and `sched.mem.pshare.guestHintsSyncEnable` configuration property set to `true`. See [Enable EM4J in the Virtual Machine](#).
- tc Runtime instances started with the required EM4J and JMX properties specified on the `java` command line. See [Setting the Required JVM Properties](#).
- CGC set up in `cron`. See [Running the Console Guest Collector](#).

### Download and Install the vSphere Web Client EM4J Plug-in

#### Procedure

1. Browse to the VMware Downloads Web page at <http://downloads.vmware.com>.
2. Under Application Platform, click VMware vFabric tc Server.
3. Click the Driver & Tools tab.
4. Click Download next to "VMware vFabric EM4J Console UI Plug-in 1.0".
5. When prompted, log in with your VMware account and accept the license agreement.
6. Save the file to your computer.
7. Change to the vSphere Web Client `plugin-packages` directory. If you installed vSphere Web Client in the default location, use the following command:

```
prompt> cd C:\Program Files\VMware\Infrastructure\vSphere Web Client\plugin-packages
```

8. Enter the `dir` command to see if the `em4j-client` directory exists. If it exists, delete it and all of its contents:

```
prompt> del /S/Q em4j-client
```

9. Create the `em4j-client` directory:

```
prompt> mkdir em4j-client
```

10. In Windows Explorer, double-click the EM4J Console UI plug-in you downloaded and extract it into the `em4j-client` directory. You should have the following directory structure:

```
vSphere Web Client/
  plugin-packages/
    em4j-client/
      help/
      legal/
      pluginPackage.xml
      plugins/
```

11. Choose Start > Control Panel > Administrative Tools > Services.

12. Right-click vSphere Web Client and choose Restart.

### Setting the Required JVM System Properties

If you create a tc Runtime instance using the `elastic-memory` template, the system properties that enable EM4J are added to the `java` command line for you. To monitor the tc Runtime instances with the vSphere Web Client, an additional set of JMX properties is required. The CGC connects to the JVM using JMX, and it looks for these properties on the command line to establish the JMX connection. If the JMX properties are missing, the vSphere Web Client displays an alert to make you aware that the CGC could not establish the JMX connection.

The following table lists the JVM properties required to enable EM4J for the JVM and to allow the CGC to connect via JMX.

Table 6.1. EM4J Java System Properties

Property	Description
<code>-javaagent=/path/to/balloon.jar</code>	Points to the <code>balloon.jar</code> EM4J Java library, which can be found in the <code>lib</code> directory of the tc Runtime instance. This property is added for you when you create a tc Runtime instance using the <code>elastic-memory</code> template.
<code>-agentpath=/path/to/native/library</code>	Points to the native EM4J library, which is in an architecture-specific subdirectory of the <code>lib</code> directory in the tc Runtime instance directory. This property is added for you when you create a tc Runtime instance using the <code>elastic-memory</code> template.
<code>-Dcom.sun.management.jmxremote=true</code>	Enables JMX remote agent and local monitoring. You must manually add this property to the tc Runtime instance.
<code>-Dcom.sun.management.jmxremote.port=portno</code>	Creates a JMX connector listening on the specified port. You must manually add this property to the tc Runtime instance.
<code>-Dcom.sun.management.jmxremote.authenticate=false</code>	Disables authentication for JMX. You must manually add this property to the tc Runtime instance.
<code>-Dcom.sun.management.jmxremote.ssl=false</code>	Disables JMX monitoring via SSL. You must manually add this property to the tc Runtime instance.

If you use the `tcruntime-instance` command with the `elastic-memory` template to create the tc Runtime instance, the `-javaagent` and `-agentpath` system properties are set up and the required Java and native libraries are copied into the tc Runtime instance.

For example, this command, executed in the tc Server installation directory, creates an EM4J-enabled tc Runtime instance named `myInstance`:

```
prompt$ ./tcruntime-instance.sh create myInstance -t elastic-memory
```

Once the tc Runtime instance is created, edit the `/$CATALINA_HOME/bin/setenv.sh` file and add the JMX system properties to the `JVM_OPTS` variable:



Lines are wrapped for readability.

```
JVM_OPTS="-Dcom.sun.management.jmxremote=true
-Dcom.sun.management.jmxremote.port=6969
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false"
```

## Running the Console Guest Collector

To view real, current data in the vSphere Web Client, you must configure **cron** to run the Console Guest Collector (CGC) periodically on each virtual machine with Java workloads. The CGC gathers information about the workloads and saves it in an area of the virtual machine where the vSphere Web Client can retrieve it.

JMX properties must be specified on the `java` command line for each JVM to allow the CGC to find the information it requires to connect to the JVM. The CGC reads the process table to find Java processes and then extracts the JMX connection properties from the command line. See [Setting the Required JVM System Properties](#).



A `cgc.sh` script exists in the `templates/elastic-memory/bin` directory of your tc Server installation and in the `$CATALINA_HOME/bin` directory of each tc Server instance you create using the `elastic-memory` template. You can run `cgc.sh` from any of these locations, but you should only run one instance per virtual machine.

You must have OS-level permission to edit the `crontab`.

## Procedure

1. Log in to the guest operating system as `root` or as a user with `sudo` permission.
2. Edit the `crontab` table for the user who executes tc Runtime instances, for example, the `tc-server` user.

```
sudo crontab -u tc-server -e
```

3. Add a line to execute `cgc.sh`. This example uses the `cgc.sh` located in the `templates/elastic-memory/bin` directory in the tc Server installation directory.



Lines are wrapped for readability. The entire `crontab` entry is a single line.

```
* /5 * * * * /opt/vmware/vfabric-tc-server-standard-2.6.0.RELEASE/templates/elastic-memory/bin/cgc.sh > /dev/null 2>&1
```

This `crontab` entry executes `cgc.sh` once every 5 minutes. See the man page for `cron` to modify the `crontab` entry to run the CGC at different time intervals.

4. Save the `crontab` file.

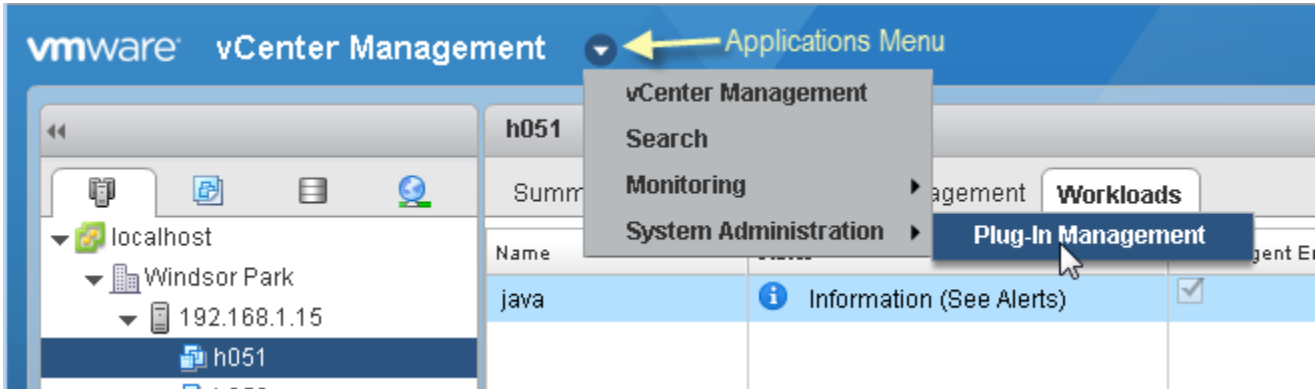
The `cgc.sh` script executes a Java class that collects information about each of the JVMs running on the virtual machine and saves it in an area of the virtual machine where the EM4J plug-in for the vSphere Web Client can retrieve it. The vSphere Web Client displays information collected by the most recent CGC execution.

## Disabling the vSphere Web Client EM4J Plug-in

You can disable the EM4J plug-in in the vSphere Web Client Plug-in Management console. The plug-in remains installed, but is inoperative. You can re-enable the plug-in later using a similar procedure.

## Procedure

1. From the Applications menu, choose System Administration > Plug-in Management.



2. Right-click Elastic Memory for Java Client and choose Other > Disable from the context menu.
3. Click Yes. A Reload vSphere Web Client dialog box appears.
4. Click Yes.

## Viewing Java Workloads in the vSphere Web Client

When the vSphere Web Client EM4J plug-in is installed, select any virtual machine in the inventory tree and then click the Workloads tab.

If the virtual machine has no JVMs, or if EM4J is not properly configured in the virtual machine, the list of workloads is empty and alert messages report why no workloads were discovered.

 A screenshot of the vSphere Web Client interface showing the "Workloads" tab for a virtual machine named "vmc-ssrc-ub1004". The interface has tabs for "Summary", "Monitor", "Resource Management", and "Workloads". Below the tabs is a table with the following columns: "Name", "Status", "EM4J Agent Enabled", "Main Class", "Virtual Machine", and "Host". The table content is "This list is empty." Below the table is an "Alerts" section with a table containing two alert messages:
 

Date Last Reported	Severity	Details
Friday, October 21, 2011 1:36:38 PM	Information	The 'sched.mem.pshare.guestHintsSyncEnable' VM option must be set to 'true' to allow JVM ballooning.
Friday, October 21, 2011 1:36:38 PM	Information	Unable to communicate with the EM4J Console Guest Collector. Is EM4J installed?

When JVMs are discovered on the virtual machine, they appear in the workloads list.

Name	Status	EM4J Agent Enabled	Main Class	Virtual Machine	Host
java	✓ Normal	<input checked="" type="checkbox"/>	org.apache.catalina.startup.Bootstrap	vmc-ssrc-rh185	vmc-ssrc-c7k

The following table describes the data presented in the workloads list.

Table 6.2. Workloads List

Column	Description
Name	The name of the workload, normally <code>java</code> .
Status	The workload status: Normal, Information, or Warning. It is a rollup value, indicating the highest severity of any alerts that exist for the workload.
EM4J Agent Enabled	A checkbox indicating if the EM4J agent is enabled. The EM4J agent is enabled when the ESXi version is supported for EM4J, VMware tools are installed, and the <code>sched.mem.pshare.guestHintsSyncEnable</code> configuration parameter is set to <code>true</code> .
Main Class	The name of the main Java class the JVM is executing.
Virtual Machine	The name of the virtual machine where the JVM is executing.
Host	The name of the ESXi host executing the virtual machine.

When a JVM is selected in the workloads list, you can select from three tabs to display information about the JVM:

- [Summary](#)
- [Alerts](#)
- [Resource Management](#)



### Summary Tab

The Workload Summary tab shows information about the workload selected in the Workloads list.

### Status

The Status section shows information about the state of EM4J for workload.

Status	
Status	✓ Normal
EM4J Agent Enabled	<input checked="" type="checkbox"/>

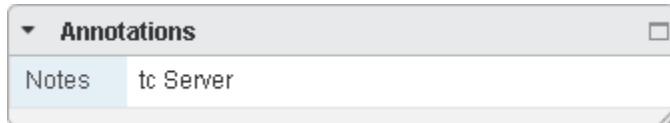
Table 6.3. Attributes in the Status Section

Attribute	Description
Overall	The overall status for EM4J, either Normal, Information, or Warning.

Attribute	Description
EM4J Agent Enabled	A checkbox indicating if the EM4J agent is enabled. The EM4J agent is enabled when the ESXi version is supported for EM4J, VMware tools are installed, and the <code>sched.mem.pshare.guestHintsSyncEnable</code> configuration parameter is set to <code>true</code> .

## Annotations

The Annotations section displays descriptive information associated with the workload. If EM4J is able to determine the name of the application the JVM is running, it is displayed here.



## JVM Details

The JVM Details section displays information about the Java virtual machine.

JVM Details	
PID	23510
Main Class	org.apache.catalina.startup.Boots...
Large Pages Supported by Guest OS	<input checked="" type="checkbox"/>
Number of Large Pages	562
JVM Using Large Pages	<input checked="" type="checkbox"/>

Table 6.4. Attributes in the JVM Details Section

Attribute	Description
PID	The operating system process ID for the executing JVM process.
Main Class	The name of the main Java class the JVM executed on startup.
Large Pages Supported by Guest OS	A checkbox indicating if <code>/proc/sys/vm/nr_hugepages</code> exists with non-zero contents, which signals that the guest OS is configured to use large memory pages. Using large memory pages is a best practice for running Java on a virtual machine, but is not a requirement to use EM4J.
Number of Large Pages	The number of large pages configured in the Guest OS; the contents of <code>/proc/sys/vm/nr_hugepages</code> .
JVM Using Large Pages	A checkbox indicating if the JVM is using large memory pages. JVM large page support is enabled with the <code>-XX:+UseLargePages</code> Java command line option.

## Related Items

The Related Items section lists information about the location of the virtual machine in the vSphere Web Client inventory.

Related Items	
Virtual Machine	vmc-ssrc-rh185
Host	vmc-ssrc-c7k2-esx-12.eng.vmware.com
Resource Pool	RP4-molshan-LargePages

Table 6.5. Attributes in the Related Items Section

Attribute	Description
Virtual Machine	The name of the virtual machine where the workload is executing.
Host	The host name or IP number of the ESXi host.
Resource Pool	The name of the resource pool the virtual machine belongs to.

### Alerts Tab

The Alerts tab displays a list of alerts that have been activated in response to the state of a Java workload. Alerts are messages that report real or potential configuration problems that prevent EM4J from operating normally.

EM4J alerts have a severity of Information, Warning, or Java Best Practice. An alert with a severity of Information reports a non-error condition that prevents EM4J from running. For example, if there are no JVMs running on the virtual machine, an alert with Information severity reports the fact.

An alert with Warning severity reports an error condition that prevents EM4J from operating normally. For example, if the Console Guest Collector on a virtual machine is unable to connect to the JMX port of a JVM, an alert with Warning severity is issued.

Java Best Practice alerts are suggestions to help improve performance when running Java on a virtual machine, for example, using large memory pages when the OS supports them.

The Status displayed on the Workloads list is the most severe alert level for the workload, or Normal if there are no alerts. When determining the severity, Java Best Practice alerts are treated as Information alerts.

A list of the potential alerts appears in the Elastic Memory for Java Client Plug-in Online Help topic for the Alerts tab.

### Resource Management Tab

The Resource Management tab displays information about memory usage in JVMs and the virtual machine, and JVM garbage collections. The EM4J memory statistics displayed on the Resource Management tab are a snapshot of memory utilization and garbage collection when the Console Guest Collector last ran on the guest OS.



The statistics shown in the Guest Memory section are updated more frequently than the JVM Heap Memory statistics and provide the most up-to-date and accurate measure of ballooned memory size for the virtual machine.

The JVM Heap memory statistics include details reported by all JVMs running when the CGC executed last. When a JVM exits, its balloon memory can persist in the ESXi-managed balloon for the virtual machine, although it is no longer associated with a JVM. For this reason, the total balloon size reported for all EM4J-enabled JVMs may be less than the actual balloon size.

### Heap Memory: Selected JVM

This section includes a stacked bar graph depicting the composition of heap memory in the JVM selected in the Workloads list.

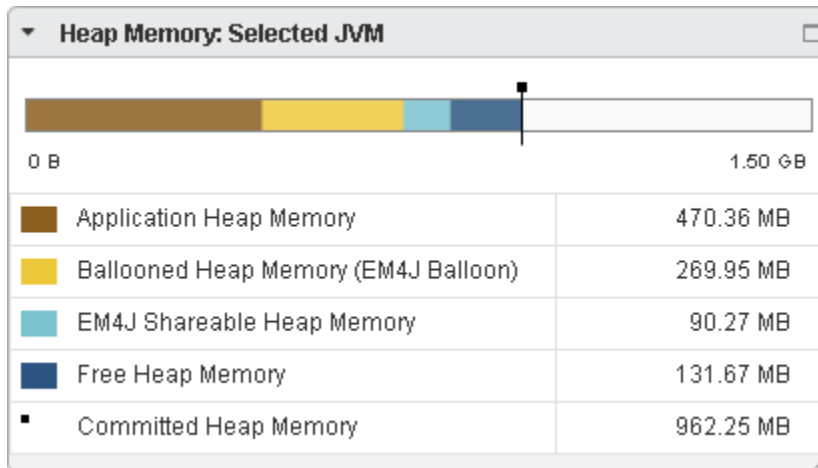


Table 6.6. JVM Heap Memory Statistics

Attribute	Description
Application Heap Memory	Amount of heap memory currently used, excluding EM4J ballooned and shareable memory.
Ballooned Heap Memory (EM4J Balloon)	Amount of the EM4J memory reclaimed by ESXi.
EM4J Shareable Heap Memory	Amount of heap memory objects owned by EM4J that are not currently ballooned, but can still be reclaimed by ESXi as zero-shared memory.
Free Heap Memory	Amount of free heap memory.
Committed Heap Memory	Amount of heap memory currently allocated to the JVM. It varies as the JVM requests additional memory or releases memory to the operating system. Committed Heap Memory is the sum of Application Heap Memory, Ballooned Heap Memory, EM4J Shareable Heap Memory, and Free Heap Memory.

The portion of heap memory that is in use contains objects stored by applications running on the JVM (Application Heap Memory) as well as objects created by EM4J. Of the memory controlled by EM4J, some may be reclaimed by ESXi to satisfy the balloon target for the virtual machine (Ballooned Heap Memory) and the remainder is zero-filled, making it available to ESXi to reclaim with transparent page sharing (EM4J Shareable Memory).

## Heap Memory: All EM4J-Enabled JVMs

This section includes a stacked bar graph depicting the composition of heap memory for all EM4J-enabled JVMs running on the virtual machine.

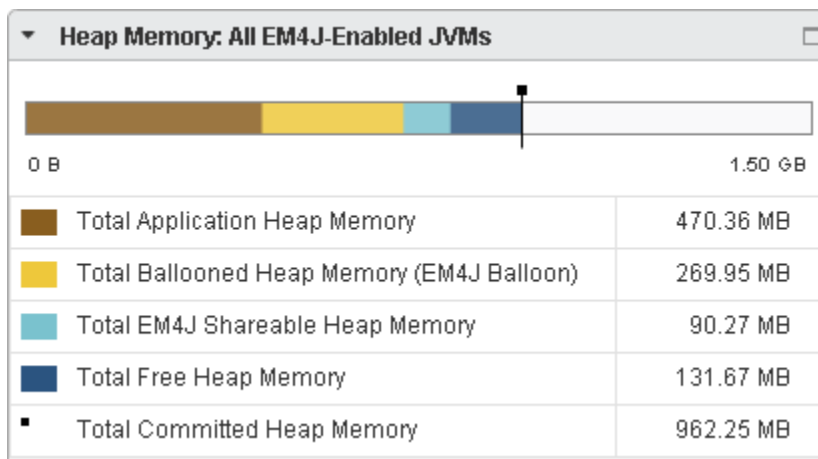


Table 6.7. JVM Heap Memory Statistics

Attribute	Description
Total Application Heap Memory	Total amount of heap memory used by all JVMs on the virtual machine, excluding EM4J ballooned and shareable memory.
Total Ballooned Heap Memory (EM4J Balloon)	Total amount of heap memory reclaimed by all EM4J balloons on the virtual machine.
Total EM4J Shareable Heap Memory	Total amount of heap memory objects owned by EM4J that are not currently ballooned, but that could be reclaimed by ESXi as zero-shared memory.
Total Free Heap Memory	Total amount of free heap memory for all JVMs on the virtual machine.
Total Committed Heap Memory	Total amount of committed heap memory for all JVMs on the virtual machine.

## Guest Memory

The Guest Memory section displays information about guest memory usage. The statistics in this section are live, unlike other sections displayed on this tab, which are from the most recent CGC execution.

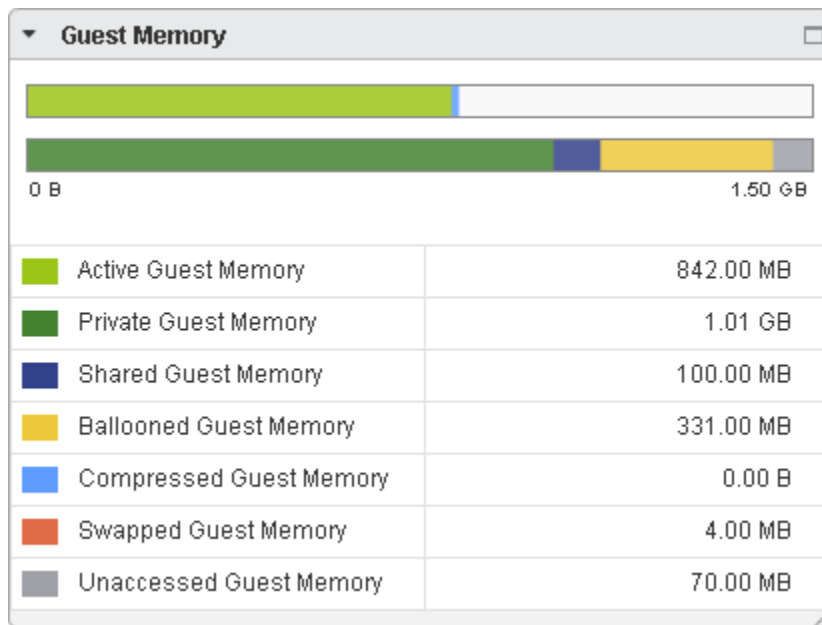


Table 6.8. Guest Memory Statistics

Attribute	Description
Active Guest Memory	Amount of memory recently accessed.
Private Guest Memory	Amount of memory backed by host memory and not being shared.
Shared Guest Memory	Amount of the virtual machine's memory being shared.
Ballooned Guest Memory	Amount of the virtual machine's memory reclaimed by ballooning.
Compressed Guest Memory	Amount of the virtual machine's memory in compression cache.
Swapped Guest Memory	Amount of the virtual machine's memory reclaimed by swapping.
Unaccessed Guest Memory	Amount of the virtual machine's memory never referenced by the guest.

## Garbage Collectors

The Garbage Collectors section displays a list of the JVM's garbage collectors.

Garbage Collectors		
Name	Collection Count	Collection Time (ms)
PS MarkSweep	4	5164
PS Scavenge	109	10251

Table 6.9. Garbage Collectors Statistics

Attribute	Description
Name	Name of the garbage collector.
Collection Count	Cumulative number of garbage collections.
Collection Time (ms)	Cumulative execution time of the garbage collector, in milliseconds.

## Initial Memory Configuration

The Initial Memory Configuration section describes the initial JVM memory layout as configured on the java command line.

Initial Memory Configuration	
Maximum Heap Size	1.00 GB
Minimum Heap Size	1.00 GB
Maximum PermGen Size	96.00 MB
Thread Stack Size	192.00 KB
Using Large Pages	<input checked="" type="checkbox"/>

Table 6.10. Initial Memory Configuration Statistics

Attribute	Description
Minimum Heap Size	Minimum heap size, specified with the <code>-Xms</code> system property.
Maximum Heap Size	Maximum heap size, specified with the <code>-Xmx</code> JVM option.
Maximum PermGen Size	Maximum size of the permanent generation, specified with the <code>-XX:MaxPermSizeJVM</code> option.
Thread Stack Size	Size of the stack for each thread, specified with the <code>-Xss</code> JVM option.
Using Large Pages	Checked if the JVM is using large memory pages. Specified with the <code>-XX:+UseLargePages</code> JVM option. Large memory support must be configured in the OS to enable this option.

## Monitoring Memory with the EM4J MBeans

The EM4J Java library, `Balloon.jar`, includes four MBeans you can access through JMX to view the state of EM4J in a JVM.

- [com.springsource.em4j.BalloonConfig](#) reports whether the prerequisites are met to enable EM4J in the VM. Each of the MBean's attributes must be true for the other three EM4J MBeans to be present.
- [com.springsource.balloon.jmx.VmBalloonState](#) provides balloon target and size information for the entire virtual machine.
- [com.springsource.balloon.jmx.JvmBalloonState](#) provides balloon information for the JVM in which the MBeans reside.
- [com.springsource.balloon.jmx.BalloonInternals](#) provides information describing internal implementation details of the heap objects owned by EM4J.

You can use vFabric Hyperic Server or JConsole, a JMX client included with the Java HotSpot JDK, to view the MBeans.

### Viewing EM4J MBeans with Hyperic Server

To view MBeans in the Hyperic user interface, you first need to add the target JVM as a server. Follow this procedure for each JVM with EM4J MBeans you want to view.

#### Procedure

1. In the Hyperic user interface, click the **Resources** tab.
2. Click **Platforms**.
3. Click the name of the platform (VM) where the target JVM is running.
4. Select **Tools Menu > New Server**.
5. In the **Name** field, enter a name for the JVM.
6. Under **Type & Host Properties**, choose **Sun JVM 1.5** for the **Server Type**.

**New Server**

**General Properties**

\* Name:

Description:

**Type & Host Properties**

\* Server Type:

\* Install Path:   
Enter the full path of the directory where the server software is installed.

Although EM4J requires Java 1.6, you can use the Hyperic Sun JVM 1.5 server type to define the server, since the JMX interface is compatible.

7. In the **Install Path** field, enter the path to the Java installation directory.
8. Click **OK**. The server is added and a message appears:

This resource has not been configured. Please set it's Configuration Properties.

9. Click **Configuration Properties**.
10. In the **jmx.url** field, enter the JMX URL:

```
service:jmx:rmi:///jndi/rmi://localhost:6969/jmxrmi
```

The JMX port, 6969 by default, is configured in the `server.xml` file for the tc Server.

**Configuration Properties**

**Shared**

\* **jmx.url**  
JMX URL to MBean Server

**jmx.username**  
JMX username

**jmx.password**  
JMX password

**Monitoring**

**server.log\_track.enable**  
Enable Log Tracking

**server.log\_track.level**  
Track event log level

**server.log\_track.include**  
Log Pattern Match

**server.log\_track.exclude**  
Log Pattern Exclude

**server.log\_track.files**  
Log Files

**server.config\_track.files**  
Configuration Files

**server.config\_track.enable**  
Enable Config Tracking

Auto-Discover Garbage Collectors and Memory Pools?

**Ok** **Reset** **Cancel**

11. In the **jmx.username** and **jmx.password** fields enter the JMX username and password. The defaults are `admin` and `springsource`.
12. Click **OK**.

The JVMs you add appear in the **Servers** list on the **Resources** tab. Follow this procedure to view MBeans in the Hyperic user interface.

### Procedure

1. In the Hyperic user interface, click **Resources**.
2. Click the **Servers** link.
3. Click the name of the JVM you added.
4. Click the **Views** tab.
5. Click the **JMX MBean Query** button.
6. In the Object Name Pattern field, enter a query string like the following to select the EM4J MBeans:

```
com.springsource.balloon:type=com.springsource.balloon.jmx.*
```

The screenshot shows the JMX MBean Query interface with the following details:

- Navigation:** Monitor, Inventory, Alert, Control, Views (selected).
- Query:** JMX MBean Query
- Object Name Pattern:** `com.springsource.balloon:type=com.springsource.balloon.jmx.*`
- Attribute Regex Filter:** `.*`
- Operation Regex Filter:** `.*`
- Query MBeans:** 3 matches
- Results:**
  - com.springsource.balloon:type=com.springsource.balloon.jmx.JvmBalloonState**

Attributes:

Name	Value
MaxPotentialBalloonSize	680057215
PotentialBalloonSizeTenured	314371763
PotentialBalloonSizeTotal	439164595
  - com.springsource.balloon:type=com.springsource.balloon.jmx.BalloonInternals**

Attributes:

Name	Value
LockedBalloonBytes	0
PotentialBalloonSizeHard	136011443
PotentialBalloonSizeSoft	303153152
PotentialBalloonSizeWeak	0
TenuredBalloonBytesPerSec	3334144

7. Click **Query MBeans**.

## Viewing EM4J MBeans with JConsole

The Java HotSpot JDK includes JConsole, a GUI JMX client, you can use to view MBeans in tc Runtime. JConsole is not included in the JRE; you need the JDK.

JMX must be configured in the tc Server instance. The base template configures JMX for local access by default. To enable remote JMX access, you must edit `CATALINA_HOME/conf/server.xml` and change the `bind` attribute of the `JMX Listener` element from `127.0.0.1` to the computer's IP number or host name so that JMX is addressable from the network. If you have disabled JMX, you can re-enable it by adding a `Listener` to your `tc Runtime Server.xml` configuration file as shown in "Simple tc Runtime Configuration" in *VMware vFabric tc Server Administration*.

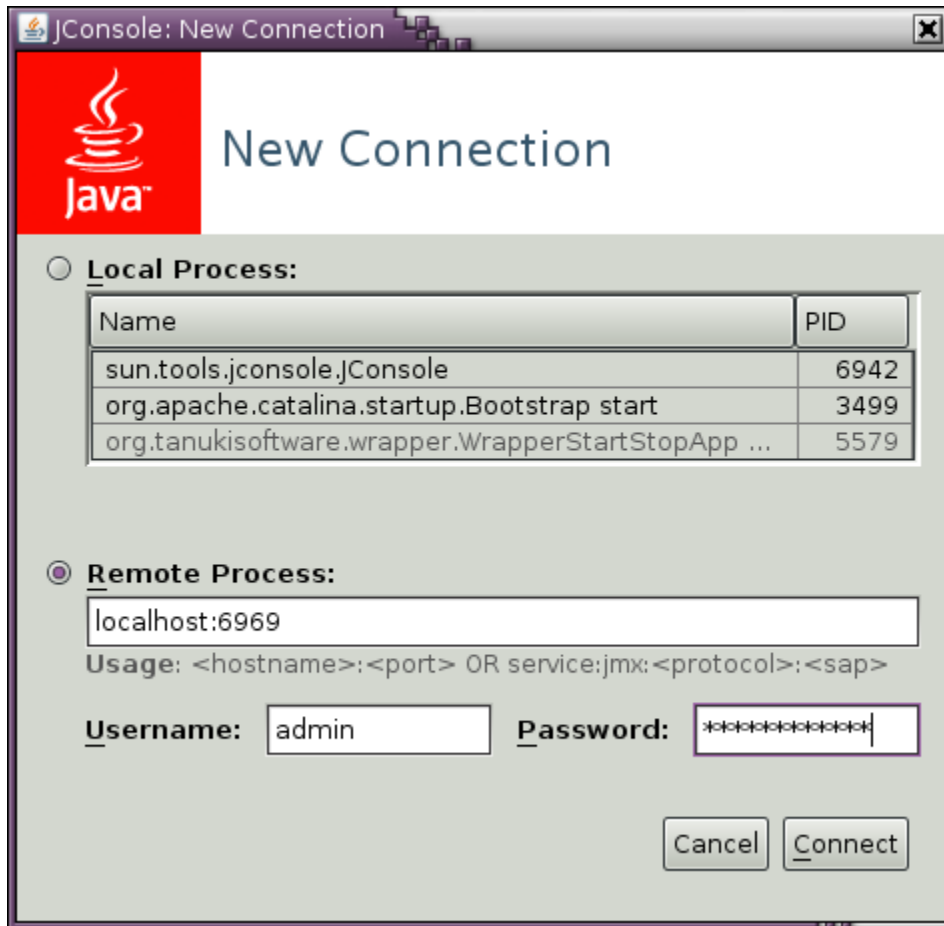
To connect to an EM4J-enabled JVM, you need to know the JMX port and, if authentication is enabled for the JMX listener, the username and password. By default, JMX listens on port 6969, the username is `admin`, and the password is `springsource`.

### Procedure

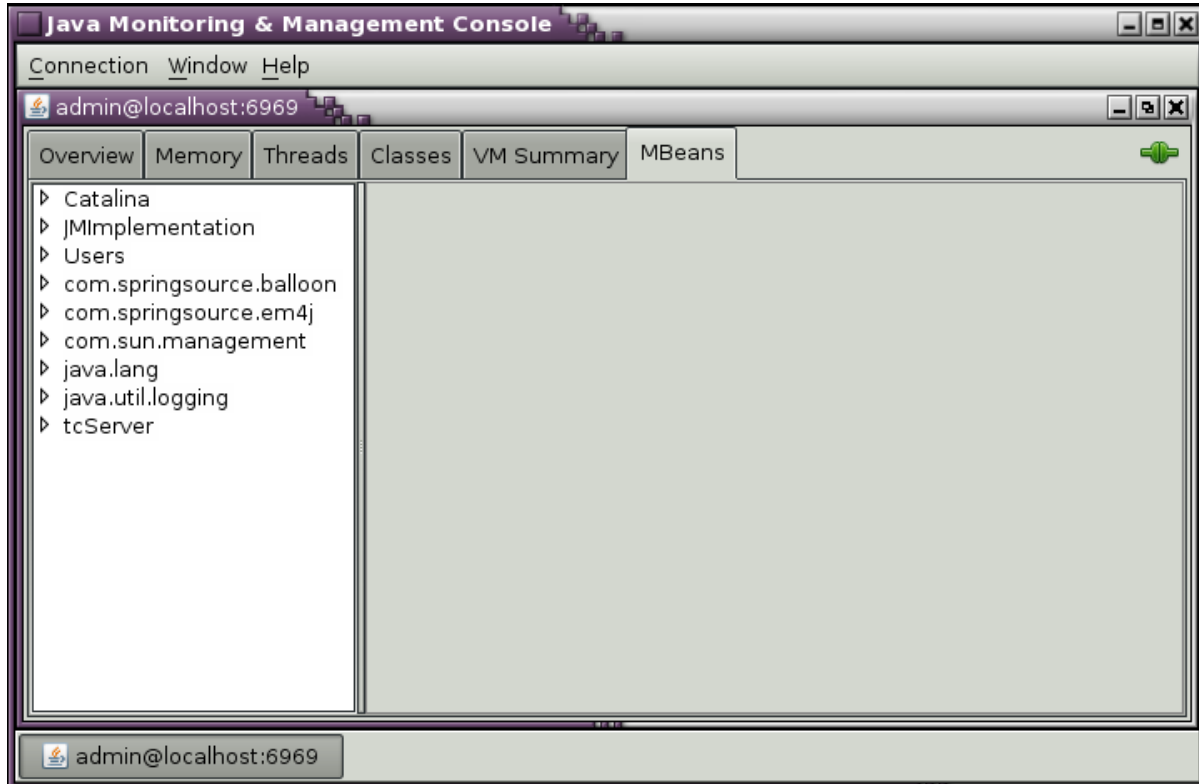
1. Open a terminal window on the VM running the JVM you want to connect to.
2. Start `jconsole` with a command like the following:

```
prompt$ $JAVA_HOME/bin/jconsole
```

3. In the **New Connection** window select **Remote Process** and enter `localhost:port`, where `port` is the JMX management port, 6969 by default:



4. Enter the JMX **Username** and **Password**. The defaults are admin and springsource, respectively.
5. Click **Connect**.
6. Click the **MBeans** tab to display the MBeans.



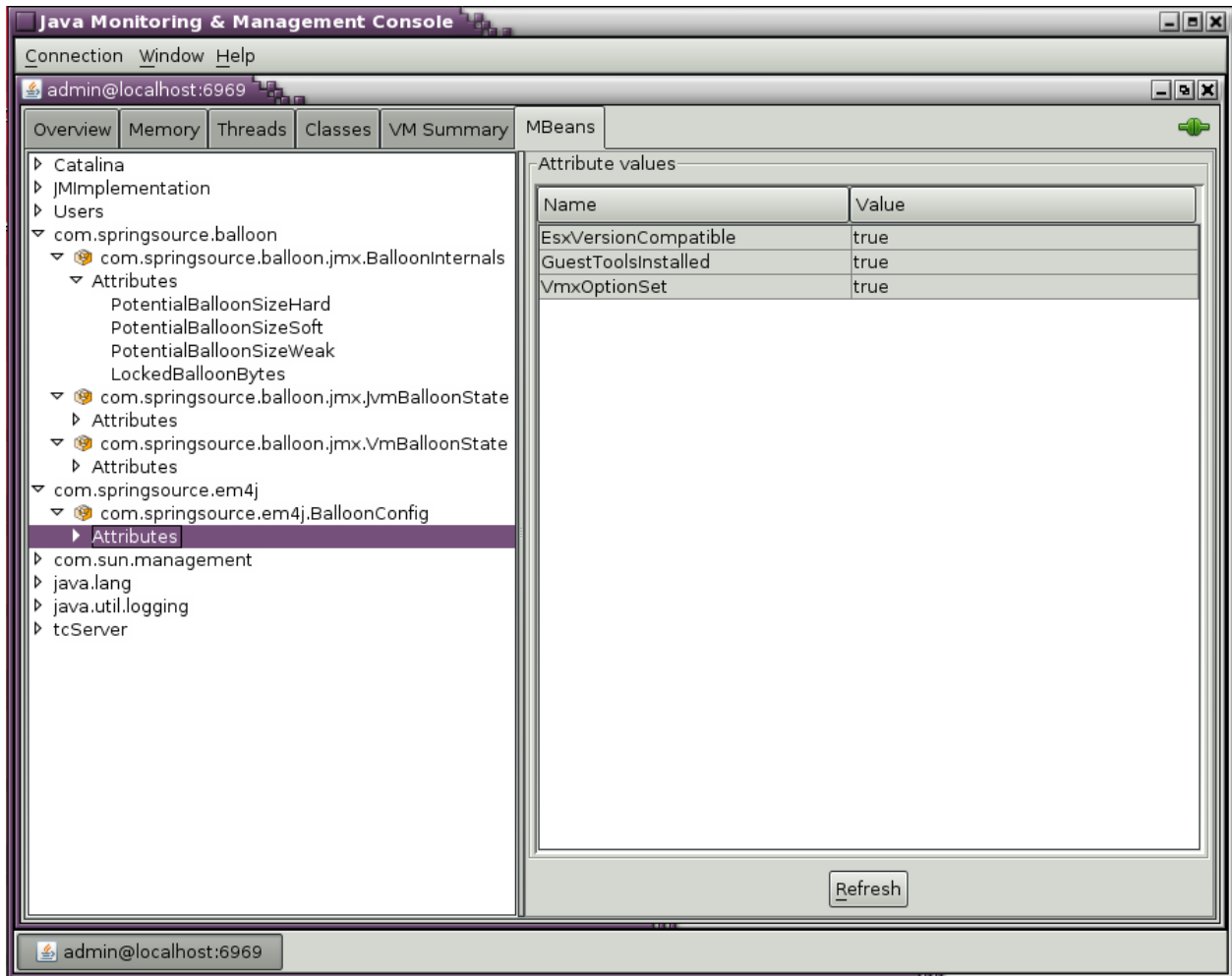
The next sections describe and illustrate the EM4J MBeans.

## BalloonConfig

The attributes of the BalloonConfig MBean report whether the prerequisites for EM4J are met. If all three attributes are `true`, EM4J is enabled in the VM. If any of the three is `false`, EM4J is disabled. The other EM4J MBeans do not exist unless these attributes are `true`. If ballooning does not occur in a VM when expected, you can check this MBean first to see if misconfiguration is the reason.

Table 6.11. Attributes of the `com.springsource.em4j.BalloonConfig` MBean

Attribute	Description
GuestToolsInstalled	This value is <code>true</code> if the VMware Tools are installed in the VM.
EsxVersionCompatible	This value is <code>true</code> if the VM is running on an EM4J-capable ESXi host. Hosts running ESXi 5.0 are compatible with EM4J.
VmxOptionSet	This value is <code>true</code> if the <code>sched.mem.pshare.guestHintsSyncEnable</code> parameter is set to <code>true</code> in the VM's <code>.vmx</code> file.

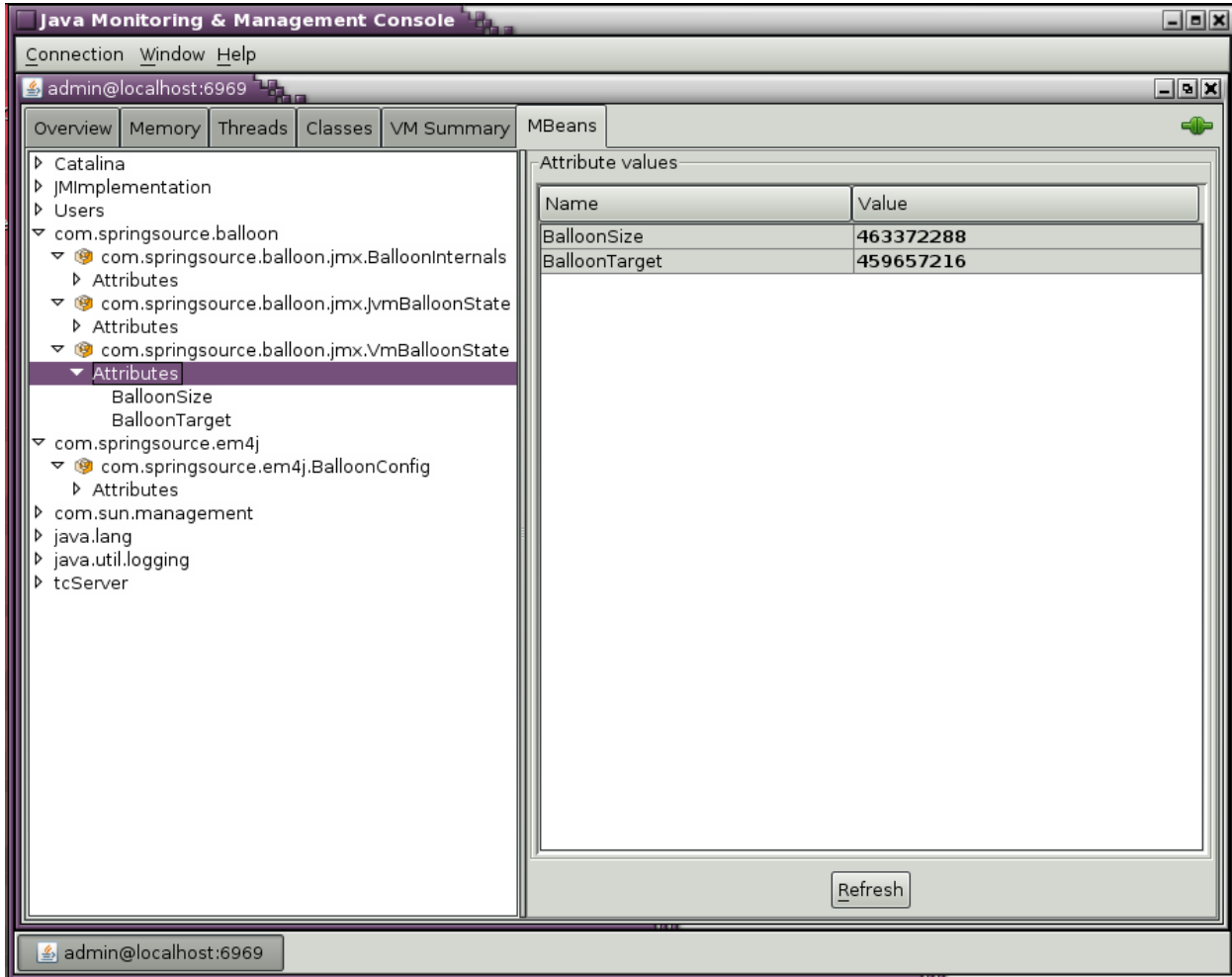


## VmBalloonState

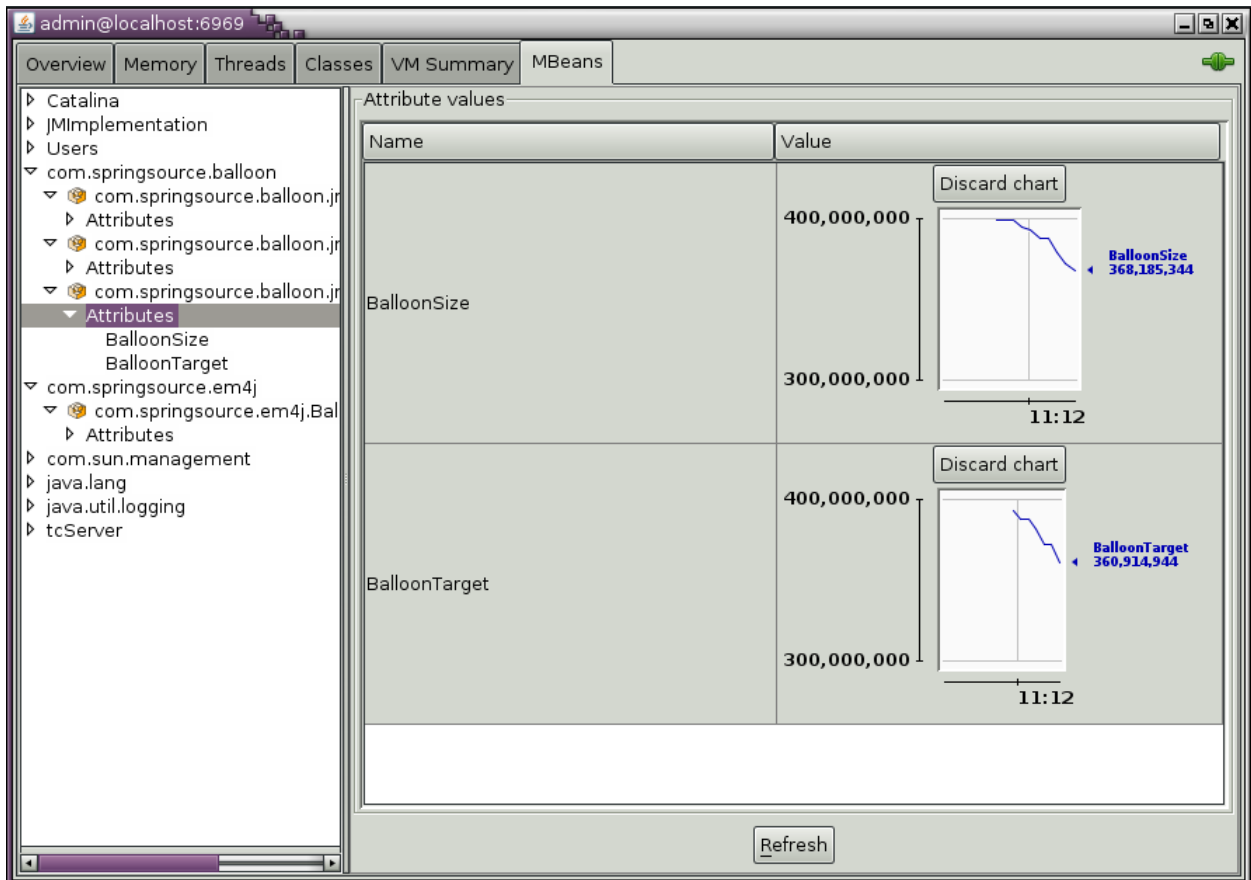
The VmBalloonState MBean reports the actual and target balloon sizes for the entire VM. This is the same balloon data reported in the vSphere Client.

Table 6.12. Attributes of the *com.springsource.jmx.VmBalloonState* MBean

Attribute	Description
BalloonSize	The composite size of all EM4J balloons in the VM.
BalloonTarget	The ESXi balloon target for this VM.



When an attribute has a numerical value, in JConsole you can double-click the value to begin graphing the value over time, as shown in the following illustration.

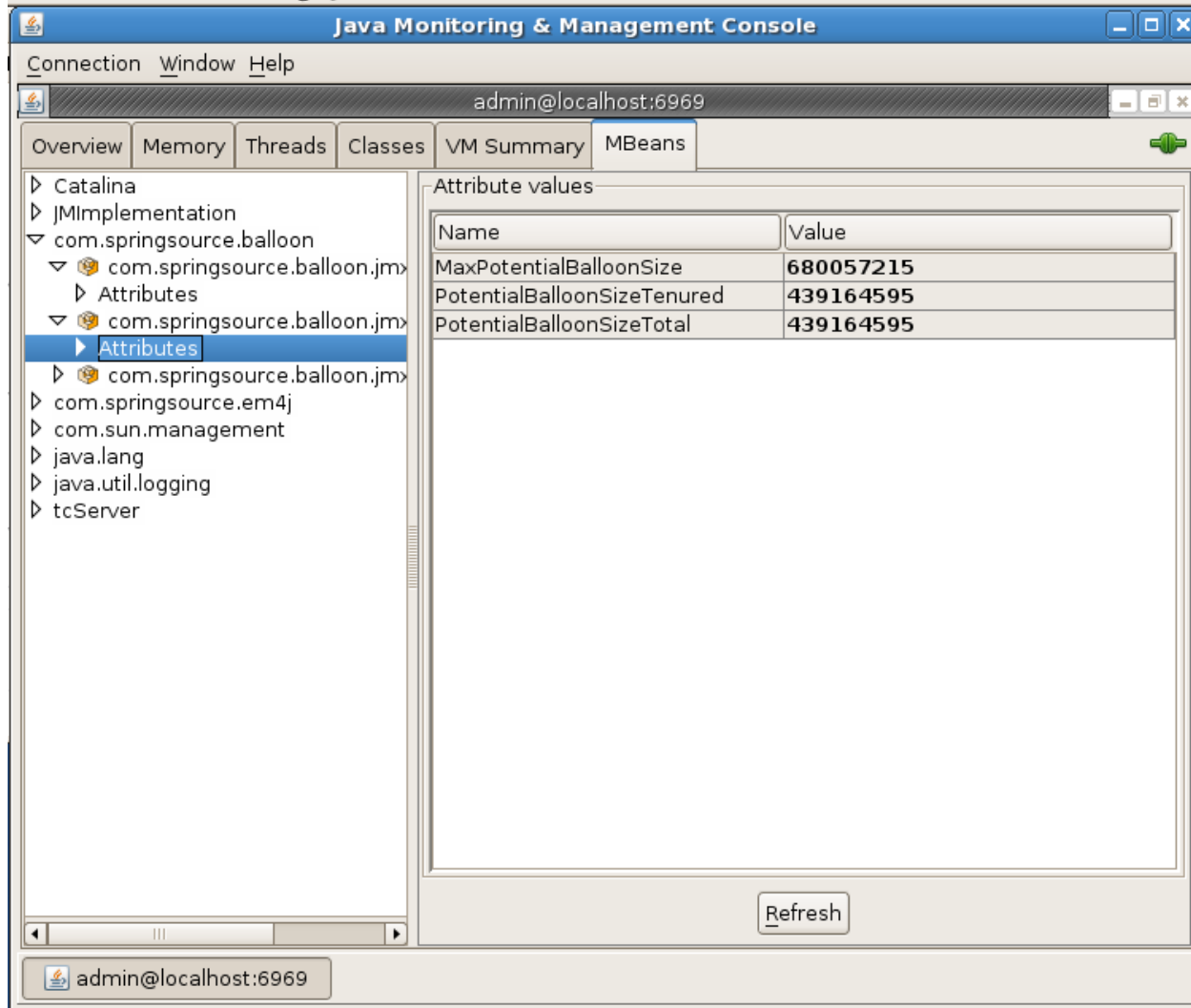


## JvmBalloonState

The `JvmBalloonState` MBean reports information about the potential size of the balloon in the JVM. The potential size is the total size of all of the objects in the heap owned by EM4J. Each of these objects can form part of the balloon in the JVM. The potential size can be larger than the balloon target, in which case EM4J maintains objects that are not part of the balloon. When the balloon target drops, the excess objects are not immediately released because recreating them is unnecessarily expensive.

Table 6.13. Attributes of the `com.springsource.jmx.JvmBalloonState` MBean

Attribute	Description
MaxPotentialBalloonSize	The maximum theoretically possible balloon size for the JVM. It is a percentage of the tenured part of the Java heap.
PotentialBalloonSizeTotal	The total size of all objects in the heap owned by EM4J.
PotentialBalloonSizeTenured	The total size of all objects owned by EM4J in the tenured portion of the heap. Balloon objects are created in the Eden generation and become tenured soon after. The objects are not useful for ballooning until they have been tenured.



## BalloonInternals

The attributes of the BalloonInternals MBean report the composition of heap memory allocated for ballooning by reference type, as well as the number of bytes locked. Potential sizes are the numbers of bytes of heap memory allocated and potentially available to satisfy the ESX balloon target for the VM. Objects allocated with weak or soft references allow the Java application to reclaim memory from the pool allocated for the balloon, if needed.

Table 6.14. Attributes of the *com.springsource.balloon.jmx.BalloonInternals* MBean

Attribute	Description
PotentialBalloonSizeHard	The total of the sizes of strongly referenced balloon objects.
PotentialBalloonSizeSoft	The total of the sizes of softly referenced balloon objects in the heap.
PotentialBalloonSizeWeak	The total of the sizes of weakly referenced balloon objects.
LockedBalloonBytes	The number of locked bytes in balloon; this is memory reclaimed by ESX.
TenuredBalloonBytesPerSec	The rate at which balloon bytes are tenured.

The screenshot shows the Java Monitoring & Management Console interface. The left pane displays a tree view of MBeans, with the path `com.sun.management.OperatingSystem` selected. The right pane shows the 'Attribute values' table for this MBean.

Name	Value
LockedBalloonBytes	206536704
PotentialBalloonSizeHard	0
PotentialBalloonSizeSoft	606269440
PotentialBalloonSizeWeak	73793536
TenuredBalloonBytesPerSec	42620108

## EM4J and DRS or vMotion

VMware Distributed Resource Scheduler (DRS) and VMware vMotion can migrate VMs from one host to another. When you migrate a VM with EM4J enabled, the result depends on whether the host to which the VM is migrated supports EM4J.

EM4J ballooning works with ESXi 5.0 and higher, but an EM4J-enabled VM can migrate to earlier ESX releases. This allows you to use EM4J-enabled VMs in an environment with mixed host versions, where it may be necessary to migrate a VM to an ESX or ESXi 4.1 host.



To move a VM from ESXi 5 to ESXi 4, the VM must be created as a Version 7 Virtual Machine. Version 8 is the default for ESXi 5.0.

If a VM is moved to another EM4J-compatible host while an EM4J-enabled JVM is running, EM4J will decouple the ballooned memory from the ESXi host and may create a new balloon on the new host, depending on the memory pressure the new host is experiencing. The ballooned memory will momentarily drop to zero when the VM is moved and will quickly achieve the new size required by the new host after migration.

If you move an EM4J-enabled VM to an ESX/ESXi 4.1 host, the VMware tools balloon is enabled if the host requires it. If the VM later migrates back to an ESXi 5.0 host, the EM4J balloon is again enabled.

## Virtual Machine Suspend and Resume

When a VM is suspended, its memory is released to ESXi for use by other VMs. If the VM is resumed, ESXi reallocates memory to it and EM4J quickly re-establishes a balloon if one is still required.