# Garbage Collection

This page covers how to monitor Garbage Collection for Java applications.

AppDynamics gathers Garbage Collection metrics and lets you analyze how periodic Garbage Collections affect the performance of your application. It is important to identify the impact of excessive Garbage Collection or memory-caused instability on the application. A typical Java application which runs on the Java Virtual Machine (JVM) creates objects such as strings, files, and arrays of primitives on the heap. The Java Garbage Collection is an automatic memory management process which finds and gets rid of the objects which are no longer used by the application.

The JVM periodically performs Garbage Collection to maximize available memory and the programmer need not explicitly mark the objects to be deleted. Garbage Collection requires a stop-the-world suspension of all application threads. This process affects the performance, especially for applications with large amounts of data, multiple threads, and high transaction rates.

See How to Master Your Java Memory for a review of how generational Garbage Collection works.

## Before You Begin

If your application runs under JDK version 1.6.0_26-b03 and you have configured the monitored application to use G1GC, the Java Agent cannot capture memory statistics. To capture memory statistics, you can do any one of the following:

- Remove G1GC (`-XX:+UseG1GC`) from the application startup options, or
- Upgrade the JDK to version 1.6.0_32 or higher.

In JVM version 1.7 or later, the agent attaches listeners to the Java event notification service to generate Garbage Collection metrics. In JVM versions prior to 1.7, the agent parses certain JVM log files to generate metrics, so you must verify that your JVM is generating the required logs. See Enable Log-based Garbage Collection.

## Monitor Garbage Collection

The Java Agent reports certain Garbage Collection metrics at one-minute intervals. You can view these metrics on the Heap & Garbage Collection sub-tab of the Memory tab on the Node dashboard:

- Heap utilization: free, used, committed, and available.
  This is the most coarse-grained view of memory use.
- Garbage Collection: minor, major, and total on one timeline.
  This should give you some notion of the ratio of minor to major collections.
- Minor Garbage Collection on a timeline.
- Major Garbage Collection on a timeline.
- Memory pool use, including the use of all memory spaces: Young Generation, Old Generation, and PermGen.

For a finer-grained view of the impact of Garbage Collection on application performance, you can view Garbage Collection metrics in the Metric Browser. Navigate to **Application Infrastructure Performance > *tier* > JVM > Garbage Collection**.

In addition to the periodically-collected metrics, the Metric Browser shows the following metrics triggered by minor or major Garbage Collection events:

| Metric | Triggering Event | Description |
|---|---|---|
| Allocated-Objects | Minor collection | The amount of memory allocated to the Young Generation.<br><br>A high growth rate might indicate memory thrash. The allocation rate affects the frequency of minor collection events, which can impact application performance over time.<br><br>The value for *Count* indicates how many collections were made. |

| | | |
|---|---|---|
| Promoted-Objects | Major collection | The amount of memory in use for objects promoted from Young Gen space to Old Gen space.<br><br>A high promotion rate is related to major collection events, which can have a significant impact on application performance due to the duration of a full major collection cycle.<br><br>If the promotion rate is close to the allocation rate, this might indicate premature promotion. In this case, you might want to allocate more memory to Young Gen space.<br><br>The value for *Count* indicates how many collections were made. |
| Freed Objects | Minor or Major collection | The amount of memory in use for objects being reclaimed in Young and Old space.<br><br>In a normal system, the number of objects allocated and the free rate metrics should be roughly equal.<br><br>The value for *Count* indicates how many collections were made. |
| LiveData | Major collection | The amount of memory in use that persists after major Garbage Collections.<br><br>An upward slope in the size of live data indicates a possible memory leak. |

# Tune Garbage Collection in the JVM

After reviewing Garbage Collection diagnostic metrics, you can use the following JVM arguments to tune space allocation in the JVM Garbage Collection memory pool. For example, you might want to increase the size of tenured space if your application needs to store many objects long term.

| Option | Meaning |
|---|---|
| `-Xms` | The amount of total memory allocated to young and old generation space. |
| `-XX:NewSize` | The amount of memory allocated to the young generation space. |
| `-XX:PermSize` | The amount of memory allocated to the permanent generation |

# Enable Log-based Garbage Collection

For applications running in JVMs prior to version 1.7, Garbage Collection monitoring is based on periodically parsing certain Garbage Collection logs.

To set up Garbage Collection logging for your application, use the following arguments:

```
-Xloggc:log-file-path
-XX:+Usecollector-type
-XX:+PrintGCDetails
```

The `log-file-path` option specifies where the log file is located. The table below describes the possible values for collector types for the `-XX:+Use`*`collector-type`* option:

| Collector Type | Effect |
|---|---|
| CMS Collector<br><br>`-XX:+UseConcMarkSweepGC` | Good for applications that require low pause times and that can share resources with the garbage collector.<br><br>Use the `-XX:ParallelCMSThreads=`<*n*> to set the number of threads to use. |

| | |
|---|---|
| Throughput or Parallel Collector<br><br>`-XX:+UseParallelGC`<br>`-XX:+UseParallelOldGC` | Can use multiple CPUs to speed up application throughput; good to use for work-intensive apps that can accept long pauses. |
| G1 Collector<br><br>`-XX:+UseG1GC` | Available in Java 7 and designed to be a long term replacement for the CMS collector. This is a parallel, concurrent, and incrementally compacting low-pause collector. |

To direct the Controller to display logged information, register the following node properties in the Controller UI:

```
enable-jmx-visibility=true
enable-log-based-gc=visiblity=true
```

To change the default log check interval, register the following node property:

```
logbased-visibility-log-check-interval-in-mills=1000
```

# Specify Regular Expressions for Parsing Garbage Collection Logs

When you enable log-based Garbage Collection metrics, the agent uses built-in regular expressions to accommodate different JDK and Garbage Collection type combinations. If the built-in regular expressions don't return the Garbage Collection metrics for your system, you can register the following node properties to specify custom regular expressions for parsing the logs:

- `young-gc-custom-regex-1`
- `young-gc-custom-regex-2`
- `young-gc-custom-regex-3`
- `full-gc-custom-regex-1`
- `full-gc-custom-regex-2`
- `full-gc-custom-regex-3`

After you set a custom regular expression using a node property, the agent no longer uses any of the built-in regular expressions to parse the logs.