OpenJDK 17

Using Shenandoah garbage collector with OpenJDK 17

Last Updated: 2021-11-25

Legal Notice

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

OpenJDK is a Red Hat offering on the Red Hat Enterprise Linux platform. The Using Shenandoah garbage collector with OpenJDK 17 guide provides an overview of Shenandoah garbage collector and explains how to configure it with OpenJDK 17.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE 3

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION 4

CHAPTER 1. SHENANDOAH GARBAGE COLLECTOR 5

CHAPTER 2. RUNNING JAVA APPLICATIONS WITH SHENANDOAH GARBAGE COLLECTOR 6

CHAPTER 3. SHENANDOAH GARBAGE COLLECTOR MODES 7

CHAPTER 4. BASIC CONFIGURATION OPTIONS OF SHENANDOAH GARBAGE COLLECTOR 8

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. To provide feedback, you can highlight the text in a document and add comments.

This section explains how to submit feedback.

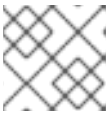
Prerequisites

- You are logged in to the Red Hat Customer Portal.
- In the Red Hat Customer Portal, view the document in **Multi-page HTML** format.

Procedure

To provide your feedback, perform the following steps:

1. Click the **Feedback** button in the top-right corner of the document to see existing feedback.



NOTE

The feedback feature is enabled only in the **Multi-page HTML** format.

2. Highlight the section of the document where you want to provide feedback.
3. Click the **Add Feedback** pop-up that appears near the highlighted text.
   A text box appears in the feedback section on the right side of the page.
4. Enter your feedback in the text box and click **Submit**.
   A documentation issue is created.
5. To view the issue, click the issue tracker link in the feedback view.

CHAPTER 1. SHENANDOAH GARBAGE COLLECTOR

Shenandoah is the low pause time garbage collector (GC) that reduces GC pause times by performing more garbage collection work concurrently with the running Java program. Concurrent Mark Sweep garbage collector (CMS) and G1, default garbage collector for OpenJDK 17 perform concurrent marking of live objects.

Shenandoah adds concurrent compaction. Shenandoah also reduces GC pause times by compacting objects concurrently with running Java threads. Pause times with Shenandoah are independent of the heap size, meaning you will have consistent pause time whether your heap is 200 MB or 200 GB. Shenandoah is an algorithm for applications which require responsiveness and predictable short pauses.

Additional resources

- For more information about the Shenandoah garbage collector, see Shenandoah GC in the Oracle OpenJDK documentation.

CHAPTER 2. RUNNING JAVA APPLICATIONS WITH SHENANDOAH GARBAGE COLLECTOR

You can run your Java application with the Shenandoah garbage collector (GC).

Prerequisites

- Installed OpenJDK. See Installing OpenJDK 17 on Red Hat Enterprise Linux in the *Installing and using OpenJDK 17 on RHEL* guide.

Procedure

- Run your Java application with Shenandoah GC by using **-XX:+UseShenandoahGC** JVM option.

```
$ java <PATH_TO_YOUR_APPLICATION> -XX:+UseShenandoahGC
```

CHAPTER 3. SHENANDOAH GARBAGE COLLECTOR MODES

You can run Shenandoah in three different modes. Select a specific mode with the **-XX:ShenandoahGCMode=<name>**. The following list describes each Shenandoah mode:

normal/satb (product, default)

This mode runs a concurrent garbage collector (GC) with Snapshot-At-The-Beginning (SATB) marking. This marking mode does the similar work as G1, the default garbage collector for OpenJDK 17.

iu (experimental)

This mode runs a concurrent GC with Incremental Update (IU) marking. It can reclaim unreachably memory more aggressively. This marking mode mirrors the SATB mode. This may make marking less conservative, especially around accessing weak references.

passive (diagnostic)

This mode runs Stop the World Event GCs. This mode is used for functional testing, but sometimes it is useful for bisecting performance anomalies with GC barriers, or to ascertain the actual live data size in the application.

CHAPTER 4. BASIC CONFIGURATION OPTIONS OF SHENANDOAH GARBAGE COLLECTOR

Shenandoah garbage collector (GC) has the following basic configuration options:

-Xlog:gc

Print the individual GC timing.

-Xlog:gc+ergo

Print the heuristics decisions, which might shed light on outliers, if any.

-Xlog:gc+stats

Print the summary table on Shenandoah internal timings at the end of the run.
It is best to run this with logging enabled. This summary table conveys important information about GC performance. Heuristics logs are useful to figure out GC outliers.

-XX:+AlwaysPreTouch

Commit heap pages into memory and helps to reduce latency hiccups.

-Xms and -Xmx

Making the heap non-resizeable with **-Xms = -Xmx** reduces difficulties with heap management. Along with **AlwaysPreTouch**, the **-Xms = -Xmx** commit all memory on startup, which avoids difficulties when memory is finally used. **-Xms** also defines the low boundary for memory uncommit, so with **-Xms = -Xmx** all memory stays committed. If you want to configure Shenandoah for a lower footprint, then setting lower **-Xms** is recommended. You need to decide how low to set it to balance the commit/uncommit overhead versus memory footprint. In many cases, you can set **-Xms** arbitrarily low.

-XX:+UseLargePages

Enables **hugetlbfs** Linux support.

-XX:+UseTransparentHugePages

Enables huge pages transparently. With transparent huge pages, it is recommended to set **/sys/kernel/mm/transparent_hugepage/enabled** and **/sys/kernel/mm/transparent_hugepage/defrag** to **madvise**. When running with **AlwaysPreTouch**, it will also pay the **defrag** tool costs upfront at startup.

-XX:+UseNUMA

While Shenandoah does not support NUMA explicitly yet, it is a good idea to enable NUMA interleaving on multi-socket hosts. Coupled with **AlwaysPreTouch**, it provides better performance than the default out-of-the-box configuration.

-XX:-UseBiasedLocking

There is a tradeoff between uncontended (biased) locking throughput, and the safepoints JVM does to enable and disable them. For latency-oriented workloads, turn biased locking off.

-XX:+DisableExplicitGC

Invoking System.gc() from user code forces Shenandoah to perform additional GC cycle. It usually does not harm, as **-XX:+ExplicitGCInvokesConcurrent** gets enabled by default, which means the concurrent GC cycle would be invoked, not the STW Full GC.

Revised on 2021-11-25 16:29:11 UTC