Red Hat build of OpenJDK 11

Using Shenandoah garbage collector with Red Hat build of OpenJDK 11

Last Updated: 2024-05-09

Legal Notice

Abstract

Red Hat build of OpenJDK is a Red Hat offering on the Red Hat Enterprise Linux platform. The Using Shenandoah garbage collector with Red Hat build of OpenJDK 11 guide provides an overview of Shenandoah garbage collector and explains how to configure it with Red Hat build of OpenJDK 11.

Table of Contents

PROVIDING FEEDBACK ON RED HAT BUILD OF OPENJDK DOCUMENTATION

To report an error or to improve our documentation, log in to your Red Hat Jira account and submit an issue. If you do not have a Red Hat Jira account, then you will be prompted to create an account.

Procedure

1. Click the following link to **create a ticket**.
2. Enter a brief description of the issue in the **Summary**.
3. Provide a detailed description of the issue or enhancement in the **Description**. Include a URL to where the issue occurs in the documentation.
4. Clicking **Submit** creates and routes the issue to the appropriate documentation team.

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.

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 Red Hat build of OpenJDK 11 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 Red Hat build of OpenJDK. See Installing Red Hat build of OpenJDK 11 on Red Hat Enterprise Linux in the *Installing and using Red Hat build of OpenJDK 11 on RHEL* guide.

Procedure

- Run your Java application with Shenandoah GC by using the **-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 Red Hat build of OpenJDK 11.

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 2024-05-09 16:49:03 UTC