JVM GC Flags for Logging and Monitoring

---

1. General JVM Options for GC

- **-XX:+UseG1GC**: Enables the G1 Garbage Collector (default in most modern JVMs).
- **-XX:+UseParallelGC**: Enables the Parallel (Throughput) Garbage Collector.
- **-XX:+UseConcMarkSweepGC**: Enables CMS Garbage Collector (deprecated in newer JVMs).
- **-XX:+UseZGC**: Enables Z Garbage Collector (low-latency collector for large heaps).
- **-XX:+UseShenandoahGC**: Enables Shenandoah Garbage Collector (low-pause collector).
- **-XX:+UseEpsilonGC**: Disables garbage collection entirely (useful for testing memory behavior).

---

2. GC Logging Flags

These flags help in generating detailed logs for understanding GC behavior and tuning performance.

Unified JVM Logging (Java 9+)

- **-Xlog:gc***: Enables logging of all garbage collection-related events.
- **-Xlog:gc=info**: Logs informational GC events (e.g., GC start/end, pause times).
- **-Xlog:gc+heap=debug**: Detailed logs about heap usage during GC.
- **-Xlog:gc+cpu=debug**: Logs CPU usage during GC.
- **-Xlog:gc*:file=gc.log:time,uptime,level,tags**: Directs GC logs to a file with a specified format.
- **-Xlog:gc+phases=trace**: Logs detailed phases of GC execution.

Pre-Java 9 Logging (Deprecated in Java 11+)

- **-XX:+PrintGC**: Prints a basic GC log line for each collection.
- **-XX:+PrintGCDetails**: Detailed logs of GC events, including heap regions.
- **-XX:+PrintGCDateStamps**: Includes timestamps in GC logs.
- **-XX:+PrintTenuringDistribution**: Outputs survivor space age information.
- **-XX:+PrintHeapAtGC**: Logs heap usage before and after each GC.

---

3. Performance Tuning and Monitoring Flags

- **-XX:MaxGCPauseMillis=<ms>**: Targets maximum GC pause time (useful for low-latency applications).
- **-XX:GCTimeRatio=<value>**: Configures the ratio of GC time to application time.
- **-XX:+AlwaysPreTouch**: Pre-touches memory pages during startup to avoid delays later.
- **-XX:+UnlockDiagnosticVMOptions -XX:+G1SummarizeConcMark**: Provides a summary of concurrent marking phases in G1GC.
- **-XX:+G1PrintRegionLivenessInfo**: Outputs detailed G1 region liveness information during GC.
- **-XX:+ParallelRefProcEnabled**: Enables parallel processing of references during GC.

---

4. Container-Specific JVM GC Flags

When running JVMs in containerized environments, specific flags are useful for proper resource management.

Memory Management

- **-XX:+UseContainerSupport**: Ensures JVM respects container memory limits (enabled by default in Java 10+).
- **-XX:MaxRAMPercentage=<value>**: Specifies the maximum percentage of container memory to be used by the JVM (default: 25%).
- **-XX:InitialRAMPercentage=<value>**: Sets the initial heap size as a percentage of container memory.
- **-XX:MinRAMPercentage=<value>**: Sets the minimum heap size as a percentage of container memory.

CPU Management

- **-XX:+UseAdaptiveSizePolicy**: Dynamically adjusts heap size and GC-related settings based on load.
- **-XX:ParallelGCThreads=<n>**: Sets the number of threads used by parallel GC in proportion to available CPUs.
- **-XX:ConcGCThreads=<n>**: Sets the number of threads for concurrent GC operations in CMS or G1GC.

Logging

- **-XX:+PrintFlagsFinal**: Outputs all JVM flags and their final values, showing container-specific adjustments.
- **-Xlog:os+container=debug**: Logs container-specific resource details (CPU, memory limits).

---

5. Advanced GC Diagnostic Flags

- **-XX:+UnlockExperimentalVMOptions**: Unlocks experimental GC features (e.g., ZGC, Shenandoah).
- **-XX:+HeapDumpOnOutOfMemoryError**: Generates a heap dump when an OutOfMemoryError occurs.
- **-XX:HeapDumpPath=<path>**: Specifies the directory for heap dumps.
- **-XX:+ExitOnOutOfMemoryError**: Forces the JVM to exit immediately when an OutOfMemoryError occurs.
- **-XX:+UseGCOverheadLimit**: Enforces limits on the proportion of time spent in GC.
- **-XX:+TrackGCObject**: Tracks live objects and their sizes during GC.

---

6. Monitoring Tools Integration

These flags support monitoring tools like Prometheus, Grafana, and Elasticsearch.

- **-Dcom.sun.management.jmxremote**: Enables JMX for remote monitoring.
- **-Dcom.sun.management.jmxremote.port=<port>**: Specifies the JMX port.
- **-Dcom.sun.management.jmxremote.authenticate=false**: Disables JMX authentication (only for testing).
- **-Dcom.sun.management.jmxremote.ssl=false**: Disables JMX SSL.
- **-javaagent:/path/to/jolokia.jar**: Integrates JVM with Jolokia for REST-based JMX monitoring.
- **-javaagent:/path/to/prometheus-jmx-exporter.jar**: Enables Prometheus metrics for JVM.

---

7. Debugging GC Flags

- **-XX:+PrintAdaptiveSizePolicy**: Logs heap resizing and policy decisions.
- **-XX:+LogVMOutput**: Logs detailed VM activities, including GC.
- **-XX:LogFile=<file>**: Specifies the file for VM logs.

---

8. Common GC Configurations for Use Cases

Low-Latency Applications:

-XX:+UseG1GC -XX:MaxGCPauseMillis=200 -Xlog:gc* -XX:+PrintGCDetails

High-Throughput Applications:

-XX:+UseParallelGC -XX:GCTimeRatio=4 -Xlog:gc* -XX:+PrintGCDateStamps

Containerized Environments:

-XX:+UseContainerSupport -XX:MaxRAMPercentage=75 -XX:+UseG1GC -Xlog:gc*:file=/logs/gc.log

Debugging and Analysis:

-XX:+UnlockDiagnosticVMOptions -XX:+PrintFlagsFinal -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/tmp

Best Practices:

1. Start Simple and Measure Performance

- Use default GC settings initially, especially when using modern JVMs (Java 11+). Evaluate application performance before customizing flags.
- **Why?** Default settings are optimized for most use cases and provide a solid baseline.

---

2. Use Unified JVM Logging for Clarity

- Transition to **-Xlog** (Java 9+) over older flags for better control and granularity in logs.
- Use custom logging formats to include timestamps, heap usage, and uptime, e.g.:

  -Xlog:gc*:file=gc.log:time,uptime,level,tags

---

3. Leverage GC-Specific Tuning for Use Case

- **Low-latency applications**: Use G1GC, ZGC, or Shenandoah with -XX:MaxGCPauseMillis=<ms>.
- **High-throughput applications**: Use ParallelGC with -XX:GCTimeRatio=<value> to optimize throughput.
- **Memory-constrained environments**: Use adaptive heap sizing (-XX:+UseAdaptiveSizePolicy) and container-specific flags (-XX:MaxRAMPercentage).

---

4. Enable Detailed GC Logs for Diagnostics

- Always enable detailed GC logging in testing and production for troubleshooting. Example:

  -Xlog:gc+heap=debug -XX:+PrintGCDetails

- Use **log rotation** to prevent logs from consuming excessive disk space in production.

---

5. Optimize GC Settings for Containerized Applications

- Ensure JVM respects container limits by using:

  -XX:+UseContainerSupport -XX:MaxRAMPercentage=75

- Set ParallelGCThreads and ConcGCThreads explicitly based on container CPU allocation to avoid over-provisioning.

  -XX:ParallelGCThreads=<num> -XX:ConcGCThreads=<num>

---

6. Enable JMX Monitoring for Real-Time Insights

- Enable JMX for integration with monitoring tools:

  ```
  -Dcom.sun.management.jmxremote \
  -Dcom.sun.management.jmxremote.port=9090 \
  -Dcom.sun.management.jmxremote.authenticate=false \
  -Dcom.sun.management.jmxremote.ssl=false
  ```

- Use agents like **Prometheus JMX Exporter** for exporting metrics to monitoring dashboards.

---

7. Plan for GC Overhead in Capacity Planning

- Reserve ~10-20% of application CPU for GC activities when calculating resource requirements.
- Adjust GC thread counts (-XX:ParallelGCThreads) based on available CPUs to avoid overspending CPU cycles.

---

8. Use GC Flags Relevant to the JVM Version

- Avoid deprecated GC flags for modern JVMs (e.g., **CMS GC** is deprecated in Java 11+).
- Regularly review **Java release notes** to stay updated on new GC features and changes.

---

9. Tune Young and Old Generation Sizes for Your Application

- **For G1GC**:

  -XX:InitiatingHeapOccupancyPercent=45

  (Adjusts the percentage of heap usage to trigger concurrent GC cycles.)

- **For older collectors**:

  -XX:NewRatio=<ratio> -XX:SurvivorRatio=<ratio>

---

10. Conduct Regular GC Health Checks

- Periodically review:
  - GC pause times.
  - Frequency of Full GC events.
  - Survivorship patterns in PrintTenuringDistribution.
- Use dashboards for real-time visualization of GC metrics.

---

11. Avoid Over-Tuning Flags Without Data

- Don't over-complicate GC configurations without evidence from logs or monitoring data.
- **Rule of Thumb**: Let the JVM manage heap and GC by default unless specific issues are identified.
