# JDK Flight Recorder
## How to use with Mission Control

Liberica JDK
Revision 1.0
October 17, 2023

bellsoft

Copyright © BellSoft Corporation 2018-2025.

BellSoft software contains open source software. Additional information about third party code is available at https://bell-sw.com/third_party_licenses. You can also get more information on how to get a copy of source code by contacting info@bell-sw.com.

THIS INFORMATION MAY CHANGE WITHOUT NOTICE. TO THE EXTENT PERMITTED BY APPLICABLE LAW, BELLSOFT PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL BELLSOFT BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF BELLSOFT IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in this document is governed by the applicable license agreement, which is not modified in any way by the terms of this notice.

Alpaquita, Liberica and BellSoft are trademarks or registered trademarks of BellSoft Corporation. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Java and OpenJDK are trademarks or registered trademarks of Oracle and/or its affiliates. Other trademarks are the property of their respective owners and are used only for identification purposes.

# Contents


1. Introduction 5

2. What is Java Flight Recorder? 6

3. What is Mission Control? 7

4. How JFR + MC differ from other profiling tools 8

5. How to start using Java Flight Recorder 9

6. A brief overview of Mission Control reports 10

Method profiling 10

Memory 11

Lock Instances 11

File I/O and Socket I/O 12

Threads 14

Exceptions 15

Garbage Collections 15

VM Operations 16

Configuration and environment 16

7. Continuous flight recording 19

# 1. Introduction

JDK Flight Recorder, also Java Flight Recorder (JFR), is a powerful feature of Hotspot JVM. JFR takes its origin from JRockit JVM; later, it was ported to Hotspot JVM and introduced in Oracle Java 7. At the time of Java 9, JFR was fully open source, and now it is an integral part of OpenJDK. This document will highlight key features of Java Flight Recorder and how it can be useful for Java developers.

# 2. What is Java Flight Recorder?

Java Flight Recorder is a performance/diagnostic tool that can be a huge time saver for Java engineers while troubleshooting applications at runtime. As the name suggests, Java Flight Recorder collects various kinds of events from JVM runtime and records them in the form of binary log files (further referred to as recordings). In Liberica JDK 11, JFR is capable of tracing a few hundred types of events, creating a comprehensive JVM runtime picture. Recordings produced by JFR are self-contained files that can be further analyzed in Mission Control.

JDK Flight Recorder is integrated into Liberica JDK 8 and later. The Liberica JDK binaries are available in the Liberica JDK Download Center.

# 3. What is Mission Control?

Mission Control is essentially a graphical front end working with JFR recordings (binary log files) produced by Java Flight Recorder. Mission Control is also open-source. You can get Mission Control binaries in the Mission Control Download Center.

JFR recordings are straightforward, each one is a plain collection of events. For them to make sense, this data needs to be post-processed. Mission Control offers several built-in reports outlining various aspects of JVM runtime (e.g., code execution, object allocation, garbage collection, and more). Besides general reports, Mission Control has a number of rule-based heuristics for detecting typical problems of Java applications. Finally, you can use a generic event browser and customizable filters to configure reports tailored for your task.

# 4. How JFR + MC differ from other profiling tools

To a certain extent, features of JFR + Mission Control compete with those of profilers, but comparing them to traditional profilers is not very fair.

JFR is a tool of its own kind. It is a feature of JDK, so other Java profilers besides Mission Control can benefit from JFR capabilities, too.

For certain profiling tasks (e.g., identifying code hot spots or object allocation profiling), JFR + Mission Control is similar to a typical Java profiler (Visual VM, JProfiler, YourKit profiler). These are good open-source Java profilers that you can use to avoid paying for commercial tools.

Some features essential for profilers are not covered by JFR. For instance, if you want to trace SQL statements sent to the database, JFR is not your tool.

You can send application-specific events to JFR. This is a compelling way to collect telemetry peculiar to your application, though it requires upfront work to place event generation in application logic.

Finally, some kinds of JVM telemetry are available only via JFR.

# 5. How to start using Java Flight Recorder

JFR is an integral part of OpenJDK now. You do not need to download anything besides JDK or configure JVM in any special way to start using it.

JFR workflow is as follows:

- Start a flight recording session with a particular configuration.
- Once the session is active for some time, dump recording to a file.
- Stop the session after the dump (this step is optional, you can leave it active if you plan to record more information later).

There are three ways to start a flight recording session:

- Using the jcmd CLI tool included in JDK, start/stop/dump JFR sessions. You need access to the command-line on the server running your Java application to use jcmd.
- Using Mission Control application, you can do the same for both locally running and remote processes (you need a configured JMX to connect to JVM remotely).
- It is also possible to activate a JFR session on the application startup via JVM command-line options (this is very useful if you want to profile applications during startup).

Starting a flight recording session also requires a configuration (called a profile). The profile defines what kind of events will be collected during the session. It also determines per probe configuration (frequency for method sampling, duration threshold for I/O events, etc.).

There are two built-in profiles, "default" and "profiling." You can also configure and use custom profiles via Mission Control UI.

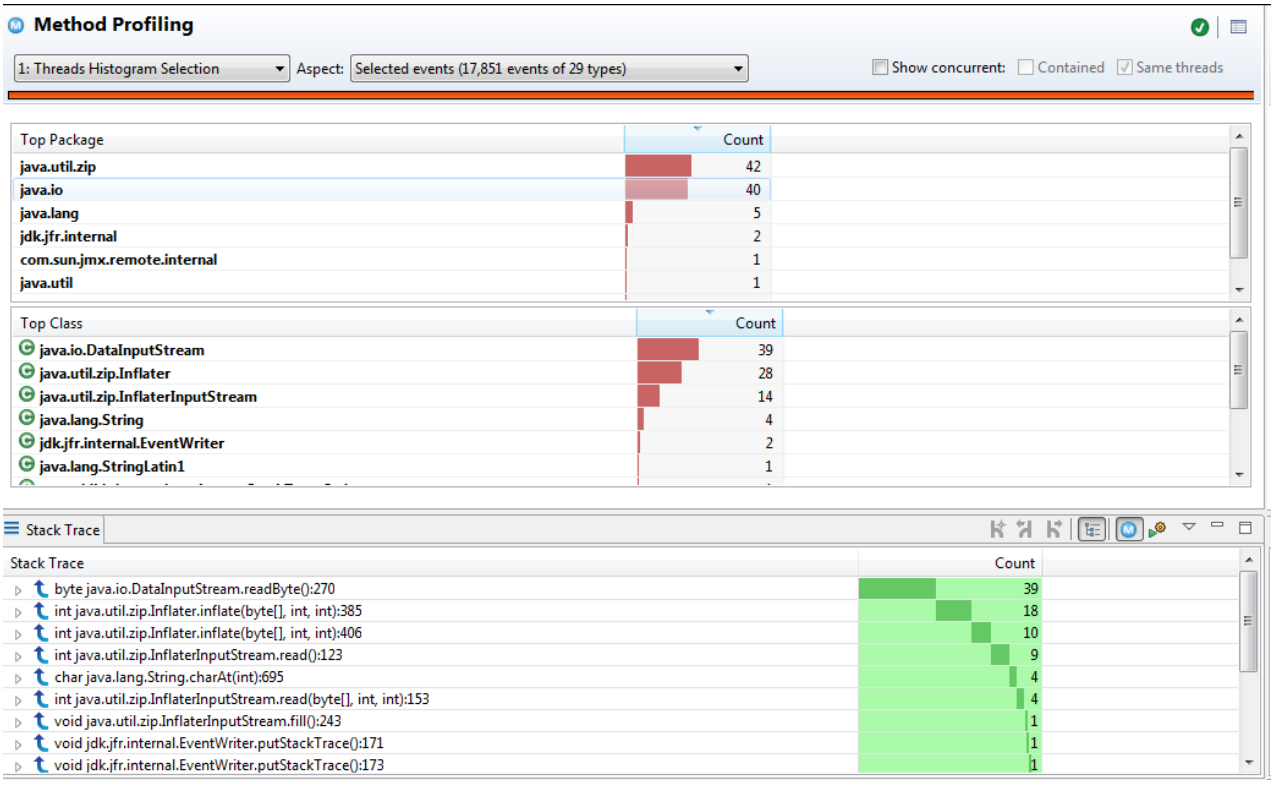
Once you have a JFR recording file, you need Mission Control to open it.

# 6. A brief overview of Mission Control reports

Mission Control has numerous features and a fairly steep learning curve. Below is a brief description of key reports available in Mission Control.

## Method profiling

JFR can periodically capture stack traces for running threads in JVM. This technique is known as sampling profiling and available in all Java profilers. Statistical analysis of a large corpus of stack traces gives an excellent picture of where the application spends most of the time down to method name or even line number. This report provides useful visualization for this subset of data collected by JFR.

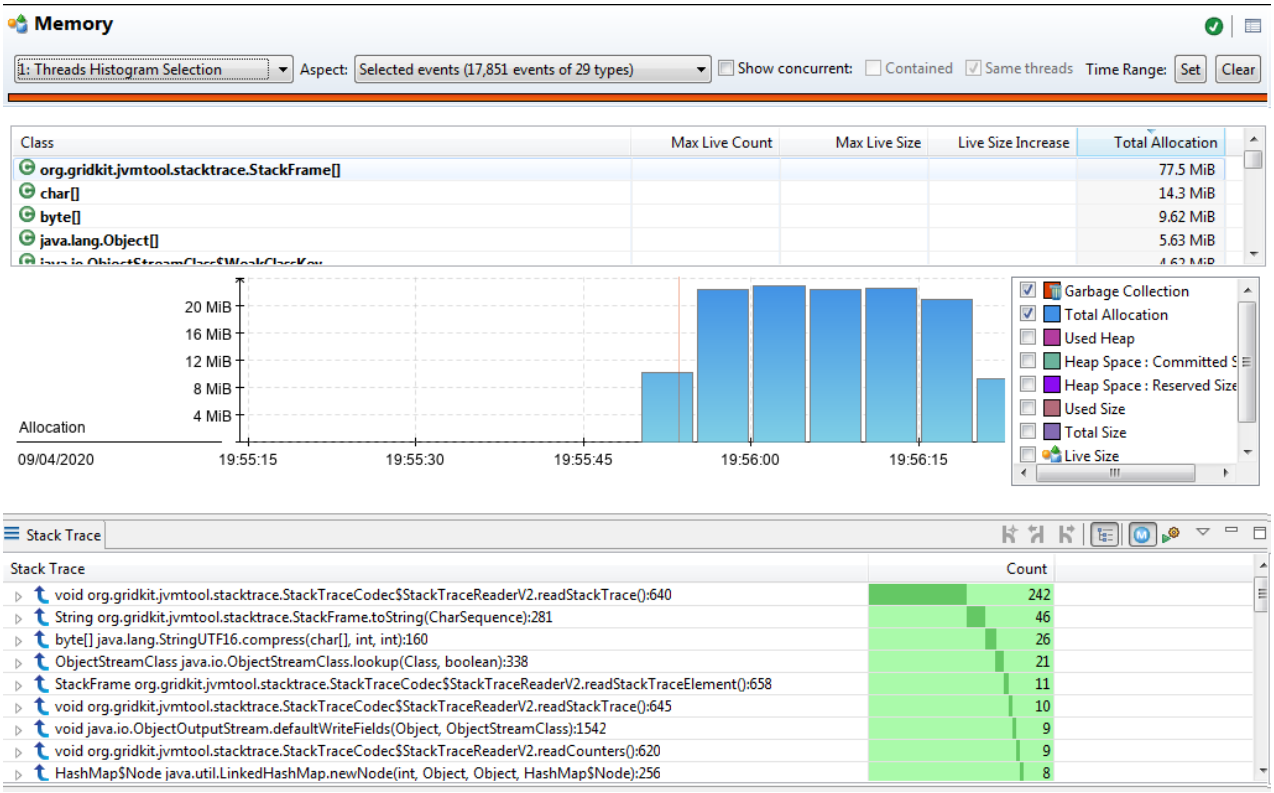



This method is typically used in cases where the application is CPU bound. Sampling helps to identify code that is inefficient or can be optimized for better performance.

# Memory

This report outlines the memory usage statistics of your Java application. It combines multiple data points, with the most important of them being object allocation samples.

If enabled, JFR records samples of new object allocation events, including allocated classes, size of object, and stack trace to the point of allocation. Statistical analysis of these samples can reveal a lot about your code.



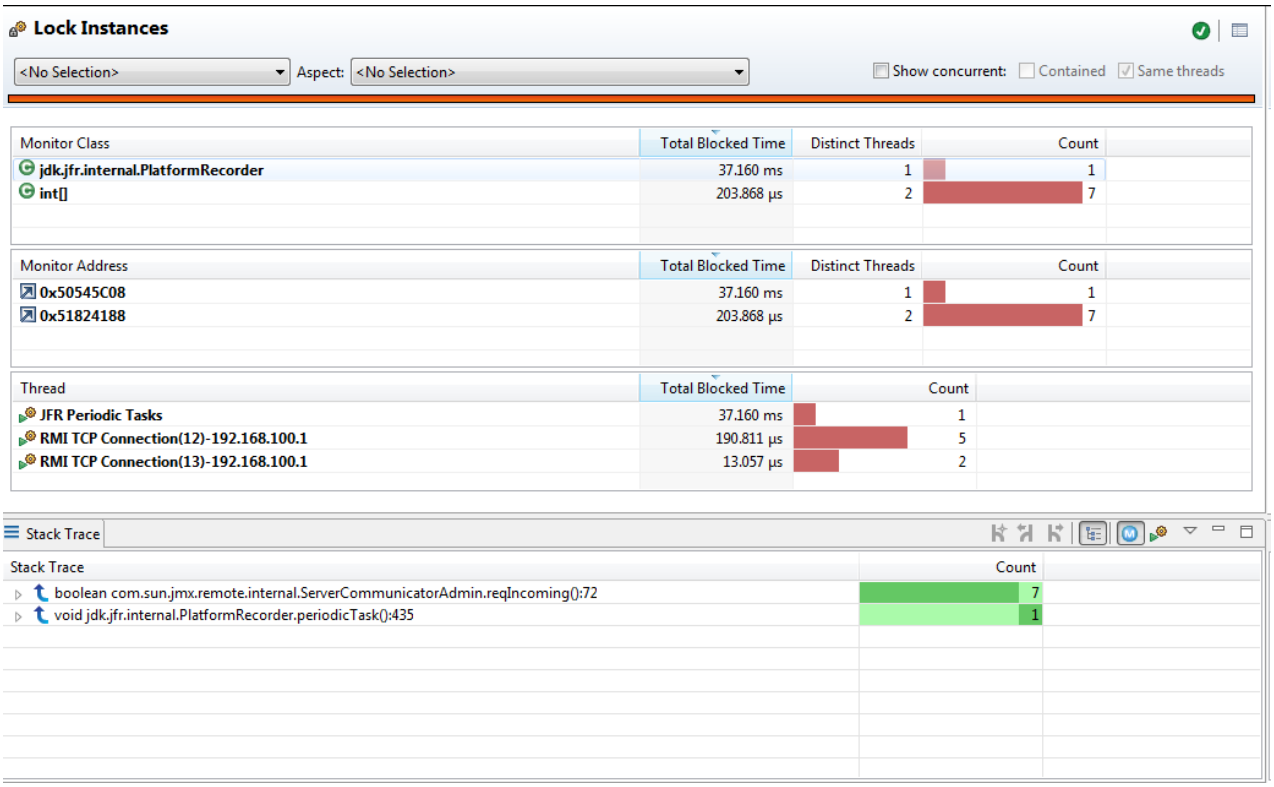

The "Memory" report shows top classes of objects being allocated. Information on actual allocated size, including stack trace to the point of allocation, is also available.

This report is most helpful if you want to reduce garbage collector pressure or optimize memory usage. Memory profiling is extremely valuable when it quickly identifies the "littering" code in your application and provides ideas for optimization.

# Lock Instances

Contentions and deadlocks is another class of Java problems hard to approach without a good tool. JFR collects events of thread blocking on synchronized sections and other concurrency primitives that

can be useful during the investigation of Java concurrency problems.

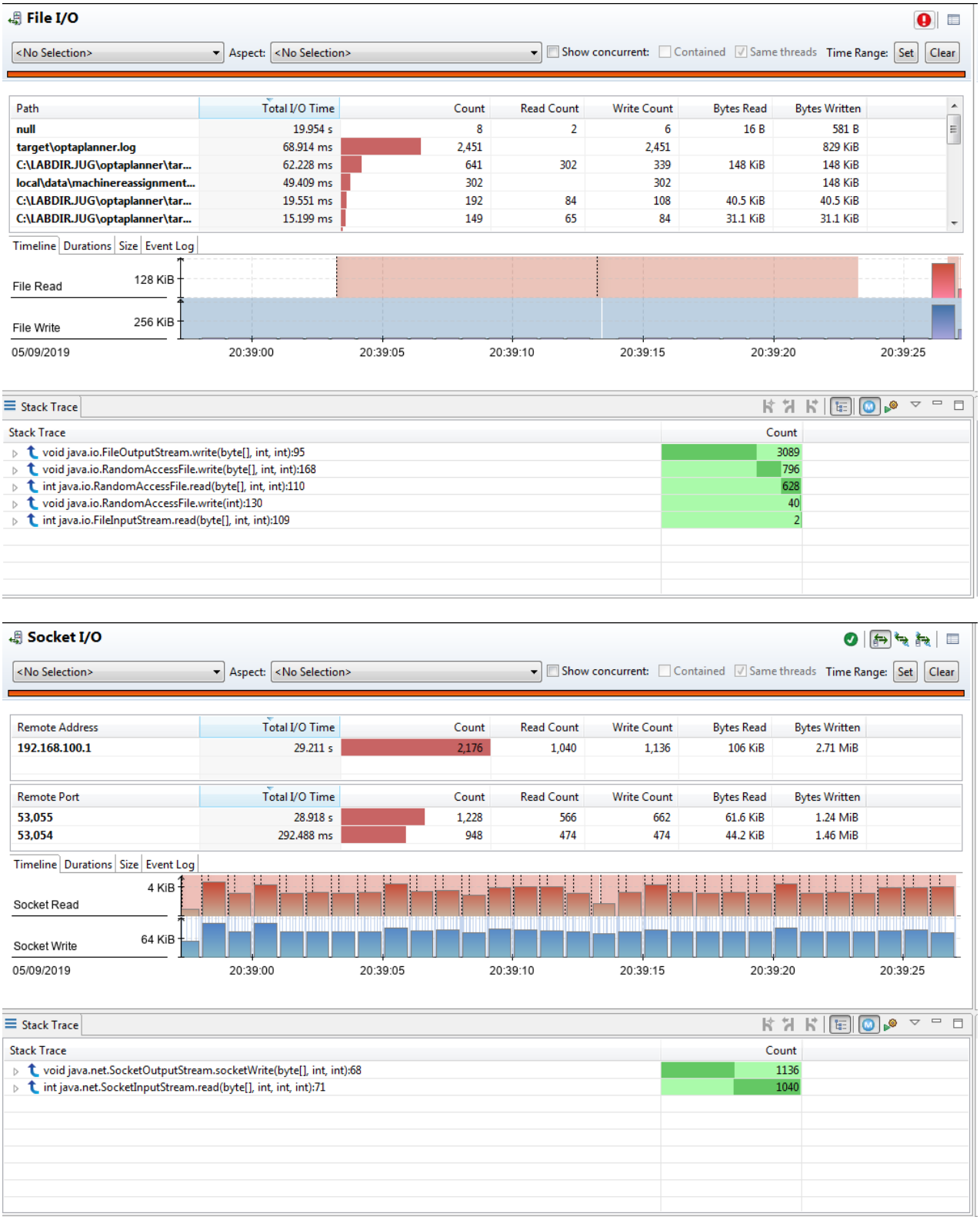



"Lock Instances" report shows you aggregated statistics for these events. "Threads" is another report helpful in tracking down concurrency issues.

A built-in "Profiling" configuration has a threshold on the duration for such events. Keep in mind that you may need to lower it to get a more detailed picture.

The cost of inter-thread contention becomes a harsh reality for high parallel code running on dozens of cores. Java ships with a few concurrency primitives, including lock-free ones. The contention report helps understand the cost of multithreading in your application and possibly highlights bottlenecks where switching to more elaborate locking is most beneficial.

## File I/O and Socket I/O




Blocking file and socket I/O operations are also traced by JFR. These reports show aggregated statistics built from file and socket I/O events, respectively.

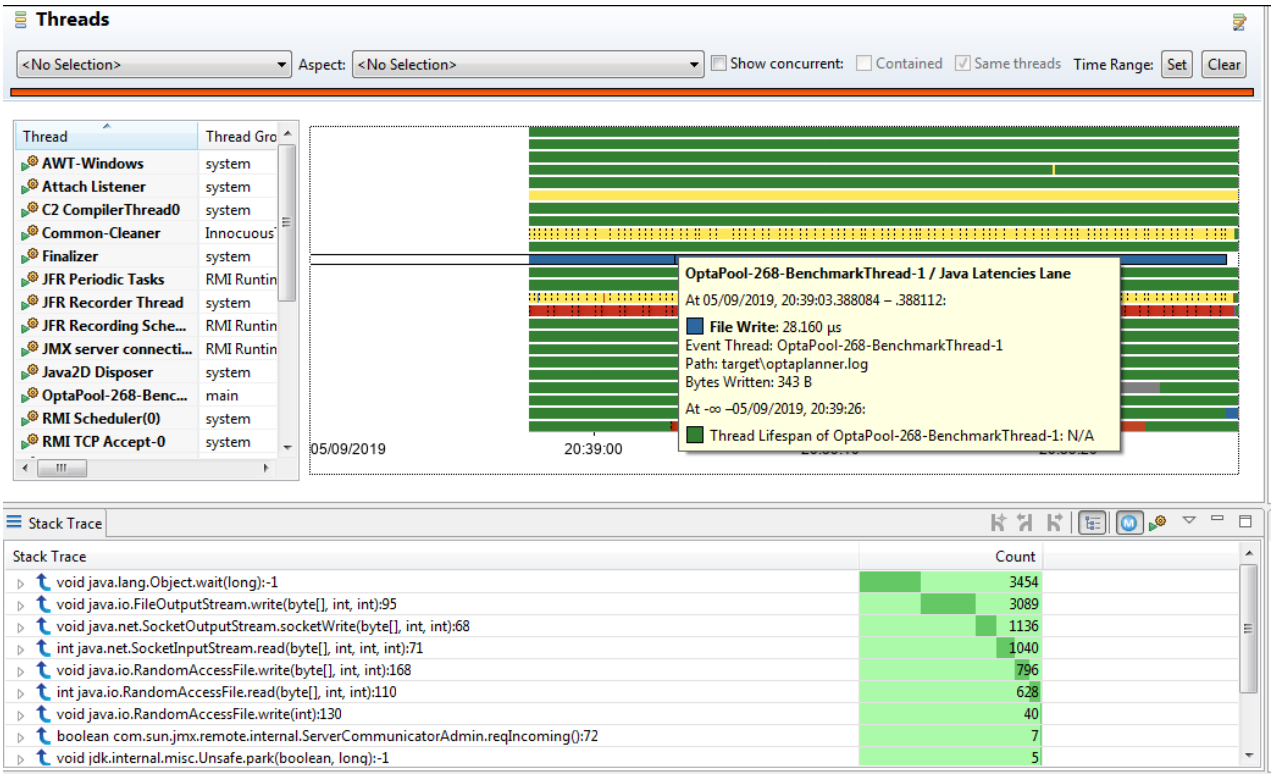
You can see which files your application has been reading and for how long or at which remote endpoints it has exchanged data.

There is a caveat, though. When using JFR for the first time, you are likely to find both of these reports to be empty or almost empty. The reason is the 10 ms threshold used by default for these kinds of events. If you want to see a full I/O picture, set this threshold to 0 when starting a flight recording session.

This report is very good at catching "unexpected" I/O events buried deep inside the layers of libraries and frameworks.

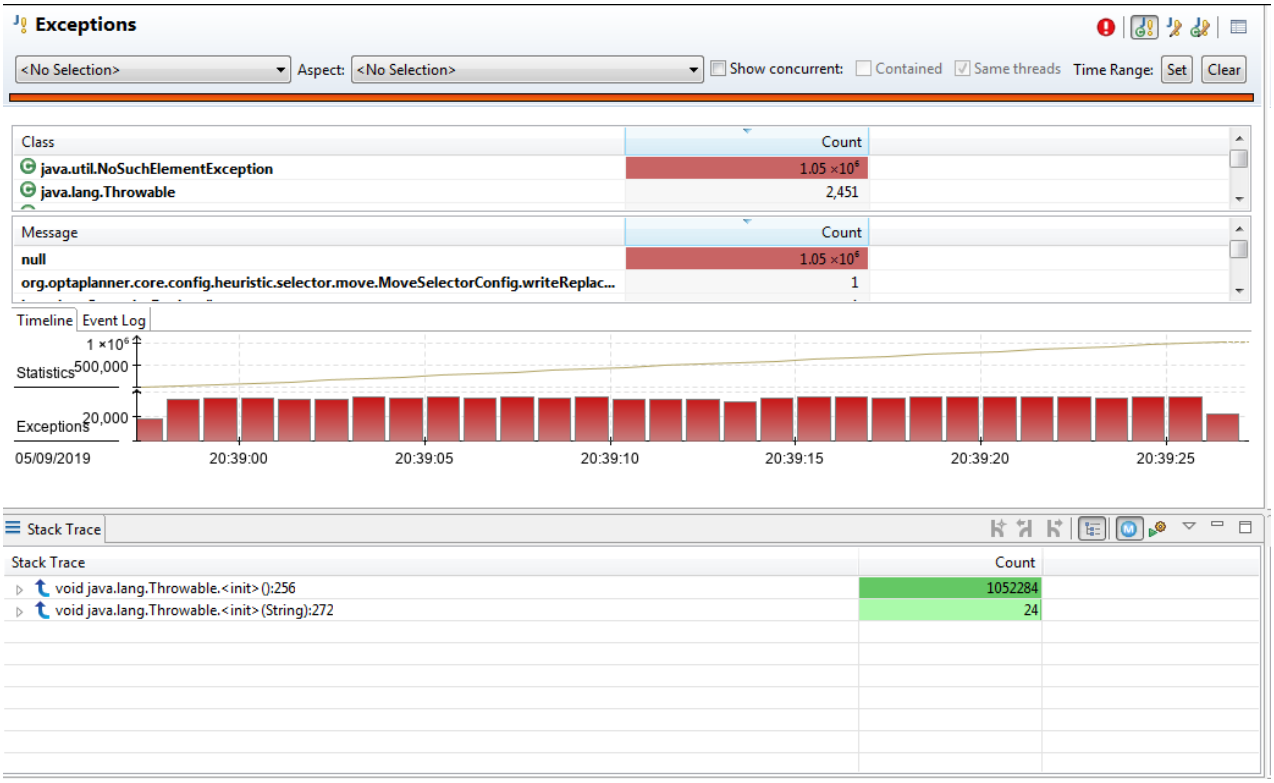
# Threads

In this report, you can see various events (contention related, I/O associated, Java thread state changes) put into a unified timeline. Here, you can see the interaction of threads in an intuitive visual way. The ability to zoom into the millisecond level of precision is helpful, too.



# Exceptions

Every Java application produces a handful of exceptions at runtime. Some of them end up in log files, but many are suppressed for one reason or another. Silently swallowed exceptions sometimes lead to many hours wasted during the incident investigation afterward.

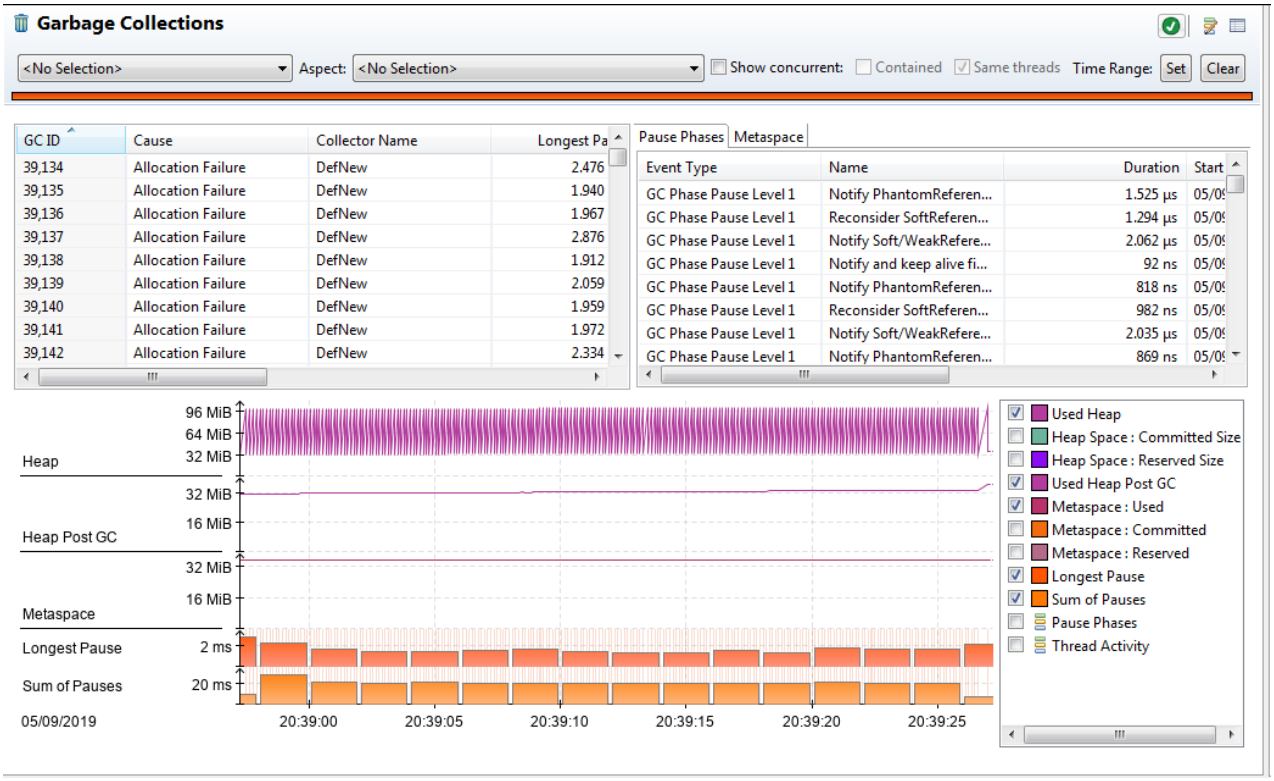



JFR sees all exceptions (whether they were suppressed or not). There are two kinds of related events. JFR records the total count of exceptions produced by runtime. This metric helps to identify exception misuse. JFR can also record every single exception created, including the stack trace, although you need to tweak the flight recording session's configuration to get this level of detail.

# Garbage Collections

GC troubleshooting typically involves digging through GC logs. JFR collects detailed information concerning GC events, and this report visualizes this data. Almost any metric of GC present in JVM is available here.

If you need to tune GC or troubleshoot abnormal GC pauses, this is a report to start with.

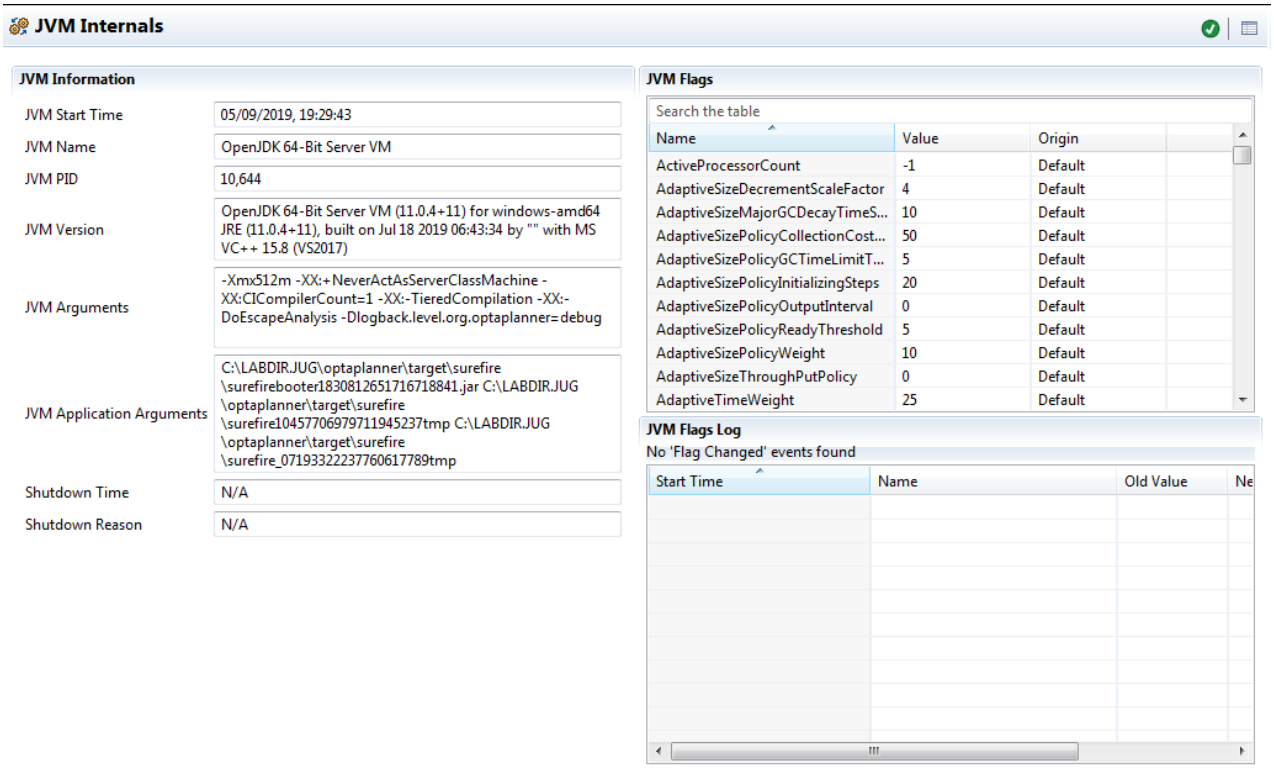



## VM Operations

HotSpot has a concept of VM operations ranging in types. But the critical point is that all of them require the Stop-the-World pause. With some JVM options, you can enable logging of VM operation or use JFR and this handy report.

VM operations are often mistaken for GC pauses (which is just one type of VM operations). Developers tend to blame Stop-the-World on garbage collectors. This report is your first stop if the application is experiencing Stop-the-World pauses.

## Configuration and environment



JVM Internals

JVM Information

| | |
|---|---|
| JVM Start Time | 05/09/2019, 19:29:43 |
| JVM Name | OpenJDK 64-Bit Server VM |
| JVM PID | 10,644 |
| JVM Version | OpenJDK 64-Bit Server VM (11.0.4+11) for windows-amd64 JRE (11.0.4+11), built on Jul 18 2019 06:43:34 by "" with MS VC++ 15.8 (VS2017) |
| JVM Arguments | -Xmx512m -XX:+NeverActAsServerClassMachine -XX:CICompilerCount=1 -XX:-TieredCompilation -XX:-DoEscapeAnalysis -Dlogback.level.org.optaplanner=debug |
| JVM Application Arguments | C:\LABDIR.JUG\optaplanner\target\surefire \surefirebooter1830812651716718841.jar C:\LABDIR.JUG \optaplanner\target\surefire \surefire1045770697971194523 7tmp C:\LABDIR.JUG \optaplanner\target\surefire \surefire_07193322237760617789tmp |
| Shutdown Time | N/A |
| Shutdown Reason | N/A |

JVM Flags

Search the table

| Name | Value | Origin |
|---|---|---|
| ActiveProcessorCount | -1 | Default |
| AdaptiveSizeDecrementScaleFactor | 4 | Default |
| AdaptiveSizeMajorGCDecayTimeS... | 10 | Default |
| AdaptiveSizePolicyCollectionCost... | 50 | Default |
| AdaptiveSizePolicyGCTimeLimitT... | 5 | Default |
| AdaptiveSizePolicyInitializingSteps | 20 | Default |
| AdaptiveSizePolicyOutputInterval | 0 | Default |
| AdaptiveSizePolicyReadyThreshold | 5 | Default |
| AdaptiveSizePolicyWeight | 10 | Default |
| AdaptiveSizeThroughPutPolicy | 0 | Default |
| AdaptiveTimeWeight | 25 | Default |

JVM Flags Log

No 'Flag Changed' events found

| Start Time | Name | Old Value | Ne |
|---|---|---|---|

Environment

CPU

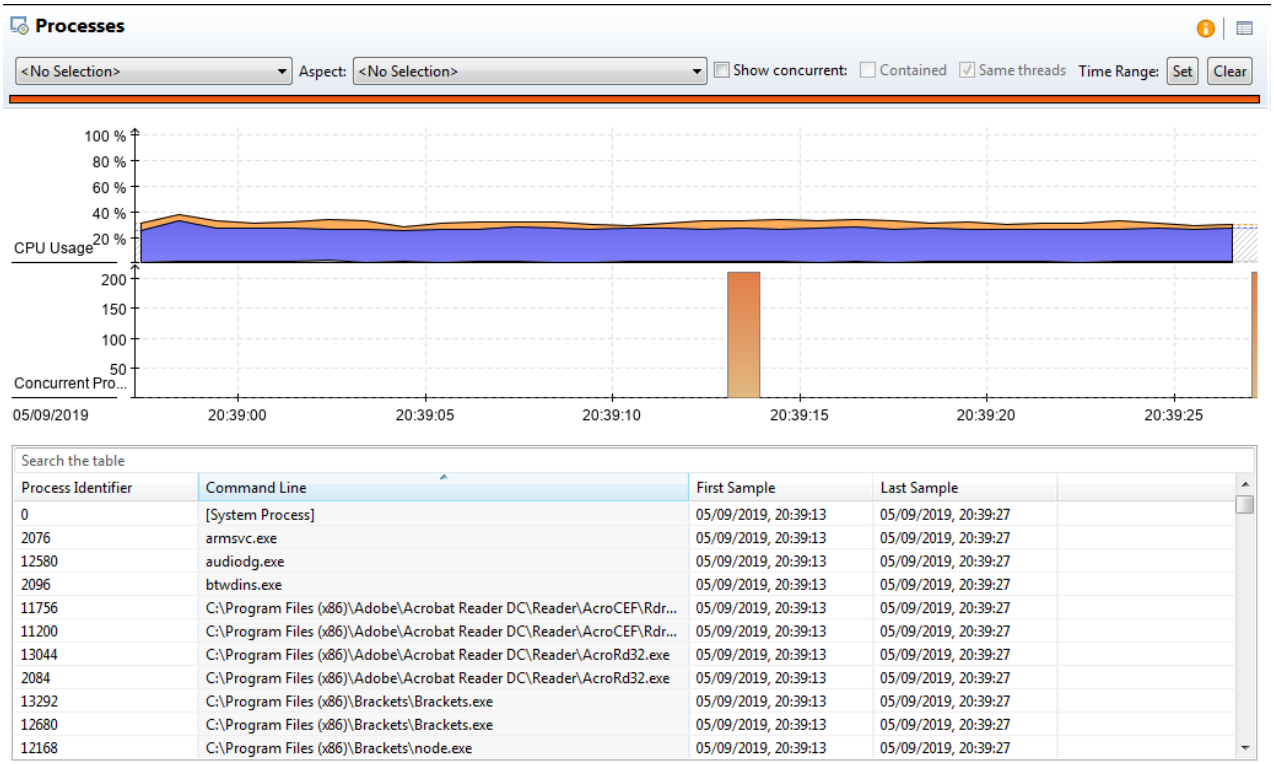
| | |
|---|---|
| CPU Type | Intel Ivy Bridge (HT) SSE SSE2 SSE3 SSSE3 SSE4.1 SSE4.2 Core Intel64 |
| Number of Cores | 2 |
| Number of Hardware Threads | 4 |
| Number of Sockets | 1 |
| CPU Description | Brand: Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz, Vendor: GenuineIntel<br>Family: Ivy Bridge (0x6), Model: Ivy Bridge (0x3a), Stepping: 0x9<br>Ext. family: 0x0, Ext. model: 0x3, Type: 0x0, Signature: 0x000306a9<br>Features: ebx: 0x01100800, ecx: 0x7fbae3ff, edx: 0xbfebfbff<br>Ext. features: eax: 0x00000000, ebx: 0x00000000, ecx: 0x00000001, edx: 0x28100800<br>Supports: On-Chip FPU, Virtual Mode Extensions, Debugging Extensions, Page Size Extensions, Time Stamp Counter, Model Specific Registers, Physical Address Extension, Machine Check Exceptions, CMPXCHG8B Instruction, On-Chip APIC, Fast System Call, Memory Type Range Registers, Page Global Enable, Machine Check Architecture, Conditional Mov Instruction, Page Attribute Table, 36-bit Page Size Extension, CLFLUSH Instruction, Debug Trace Store feature, ACPI registers in MSR space, Intel Architecture MMX Technology, Fast Float Point Save and Restore, Streaming SIMD extensions, Streaming SIMD extensions 2, Self-Snoop, Hyper Threading, Thermal Monitor, Streaming SIMD Extensions 3, PCLMULQDQ, 64-bit DS Area, MONITOR/MWAIT instructions, CPL Qualified Debug Store, Virtual Machine Extensions, Safer Mode Extensions, Enhanced Intel SpeedStep technology, Thermal Monitor 2, Supplemental Streaming SIMD Extensions 3, CMPXCHG16B, xTPR Update Control, Perfmon and Debug Capability, Process-context identifiers, Streaming SIMD extensions 4.1, Streaming SIMD extensions 4.2, x2APIC, Popcount instruction, TSC-Deadline, AESNI, XSAVE, OSXSAVE, AVX, F16C, LAHF/SAHF instruction support, SYSCALL/SYSRET, Execute Disable Bit, RDTSCP, Intel 64 Architecture, Invariant TSC |

Memory

| | |
|---|---|
| Available physical memory | 15.7 GiB |

Operating System

| | |
|---|---|
| OS Version | OS: Windows 7 , 64 bit Build 7601 (6.1.7601.23915) |




| Process Identifier | Command Line | First Sample | Last Sample |
|---|---|---|---|
| 0 | [System Process] | 05/09/2019, 20:39:13 | 05/09/2019, 20:39:27 |
| 2076 | armsvc.exe | 05/09/2019, 20:39:13 | 05/09/2019, 20:39:27 |
| 12580 | audiodg.exe | 05/09/2019, 20:39:13 | 05/09/2019, 20:39:27 |
| 2096 | btwdins.exe | 05/09/2019, 20:39:13 | 05/09/2019, 20:39:27 |
| 11756 | C:\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\AcroCEF\Rdr... | 05/09/2019, 20:39:13 | 05/09/2019, 20:39:27 |
| 11200 | C:\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\AcroCEF\Rdr... | 05/09/2019, 20:39:13 | 05/09/2019, 20:39:27 |
| 13044 | C:\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\AcroRd32.exe | 05/09/2019, 20:39:13 | 05/09/2019, 20:39:27 |
| 2084 | C:\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\AcroRd32.exe | 05/09/2019, 20:39:13 | 05/09/2019, 20:39:27 |
| 13292 | C:\Program Files (x86)\Brackets\Brackets.exe | 05/09/2019, 20:39:13 | 05/09/2019, 20:39:27 |
| 12680 | C:\Program Files (x86)\Brackets\Brackets.exe | 05/09/2019, 20:39:13 | 05/09/2019, 20:39:27 |
| 12168 | C:\Program Files (x86)\Brackets\node.exe | 05/09/2019, 20:39:13 | 05/09/2019, 20:39:27 |

You can send JFR files for analysis to an expert in the JVM domain. Files include JVM configuration, hardware configuration, and even a list of other processes running on a server and competing for resources without application.

# 7. Continuous flight recording

JFR + Mission Control can be used as a free and capable Java profiler, but using them in continuous mode can bring even more value.

To get the full benefit from continuous flight recording, you need to

- configure your JVM to be accessible via JMX;
- add JVM command-line arguments to activate Flight Recorder on startup.

A minimal command to start flight recording session on startup is the following:

```
-XX:StartFlightRecording=name=background,maxsize=100m
```

An important option is `maxsize` that limits how many events are retained in memory.

Now, you have JFR silently collecting events in the background. As long as the application runs fine, you can forget about JFR being active. The default JFR profile adds very little overhead.

At the same time, you can quickly connect to JVM via Mission Control and dump accumulated events from a problematic server, which is especially useful for tracking down problems reproducible only on live production.

One can argue that instead of pulling data dumps from individual VMs, it is better to have a centralized monitoring system where you can access diagnostics from any server. In theory, that will be perfect, but arranging quality centralized monitoring is extremely challenging. Central monitoring solutions have to compromise on the level of details.

Continuous flight recording and the ability to pull event dumps on demand are complementary to central logging and general monitoring solutions, as you can get very detailed telemetry from JVM.

# JDK Flight Recorder
## How to use with Mission Control

bellsoft