Red Hat build of Cryostat 2

Enabling dynamic JFR recordings based on MBean custom triggers

Last Updated: 2023-12-08

Legal Notice

Abstract

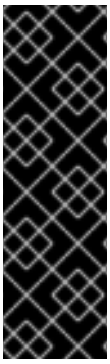
Configure the Cryostat agent to start JFR recordings dynamically based on MBean custom triggers.

Table of Contents

PREFACE

The Red Hat build of Cryostat is a container-native implementation of JDK Flight Recorder (JFR) that you can use to securely monitor the Java Virtual Machine (JVM) performance in workloads that run on an OpenShift Container Platform cluster. You can use Cryostat 2.4 to start, stop, retrieve, archive, import, and export JFR data for JVMs inside your containerized applications by using a web console or an HTTP API.

Depending on your use case, you can store and analyze your recordings directly on your Red Hat OpenShift cluster by using the built-in tools that Cryostat provides or you can export recordings to an external monitoring application to perform a more in-depth analysis of your recorded data.



IMPORTANT

Red Hat build of Cryostat is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

MAKING OPEN SOURCE MORE INCLUSIVE

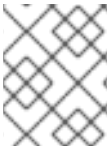
Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases.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

CHAPTER 1. CUSTOM TRIGGERS

The Cryostat 2.4 agent supports custom triggers that are based on MBean metric values. You can configure the Cryostat agent to start JFR recordings dynamically when these custom trigger conditions are met.

You can define a custom trigger condition that dynamically starts a JFR recording when this condition is met. A custom trigger condition is based on MBean counters that can cover a range of runtime, memory, thread, and operating system metrics. You can include one or more MBean counter types as part of the custom trigger condition for a JFR recording. You can also specify a duration or time period as part of the trigger condition, which means the conditional values must persist for the specified duration before the condition is met.

The Cryostat agent supports smart triggers that continually listen to the value of the specified MBean counters. Triggering occurs if the current values of the specified counters match the configured values in the custom trigger for the specified duration. If triggering occurs, the Cryostat agent dynamically starts the JFR recording at that point.



NOTE

A JFR recording will not start dynamically if the custom trigger condition associated with this recording is not met.

CHAPTER 2. CONFIGURATION OF A CUSTOM TRIGGER FOR A DYNAMIC RECORDING

When you configure your target application to load the Cryostat agent, you can define one or more custom triggers that are then passed as arguments to the agent.

For more information about configuring a target application to load the Cryostat agent, see Configuring Java applications.

2.1. OPTIONS FOR DEFINING A CUSTOM TRIGGER

You can define a custom trigger in any of the following ways:

Appending a custom trigger to the Cryostat agent's JAR file path

The following example shows how to append a simple custom trigger to the Cryostat agent's JAR file path:

```
JAVA_OPTS="-javaagent:/deployments/app/cryostat-agent-shaded.jar=\"[ProcessCpuLoad > 0.2 ;
TargetDuration > duration('30s')]~profile\""
```

The preceding example trigger instructs the agent to start a JFR recording if the **ProcessCpuLoad** metric has a value greater than 0.2 for a duration of more than 30 seconds: This example also instructs the agent to use the **profile** event template for the JFR recording.

Using a JVM system property flag

The following example shows how to specify a simple custom trigger by using a JVM system property flag:

```
-Dcryostat.agent.smart-trigger.definitions="[ProcessCpuLoad > 0.2 ; TargetDuration >
duration(\"30s\")]~profile"
```

This example uses the same custom trigger criteria as the preceding example.

Using an environment variable

The following example shows how to specify a simple custom trigger by using an environment variable:

```
- name: CRYOSTAT_AGENT_SMART_TRIGGER_DEFINITIONS
  value: "[ProcessCpuLoad > 0.2 ; TargetDuration > duration(\"30s\")]~profile"
```

This example uses the same custom trigger criteria as the preceding examples.

2.2. COMMON EXPRESSION LANGUAGE

You can use Common Expression Language (CEL) to define a custom trigger condition. CEL is a free-form expression syntax that provides great flexibility in defining rules and constraints for evaluating data. For example, you can use CEL to create relational statements for evaluating if any combination of MBean counter types have current values greater than, equal to, or less than specified configurable values. You can also include any combination of AND (**&&**) or OR (||) logic statements between different MBean counter types that are part of the same trigger condition.

For more information about CEL, see the CEL language specification.

2.3. GENERAL SYNTAX RULES FOR CUSTOM TRIGGERS

Consider the following syntax guidelines for defining custom triggers:

- A custom trigger definition must consist of both an expression that defines the overall trigger condition and the name of an event template that is used for the JFR recording.
- The entire trigger expression must be enclosed in square brackets (for example, **[ProcessCpuLoad > 0.2 ; TargetDuration < duration("30s")]**).
- For readability, you may use white space in a trigger expression as shown in the preceding example, but this is not a requirement.
- The name of the event template must be defined after the trigger expression and preceded by a tilde (~) character (for example, **~profile**).
- A trigger expression can consist of one or more constraints and a target duration. The set of constraints and target duration must be separated by a semicolon (**;**) character.
- Each constraint must include: the name of an MBean counter; a relational operator such as **>** (greater than), **=** (equal to), **<** (less than), and so on; and a specified value. The type of relational operator and value that you can specify depends on the associated MBean counter type (for example, **ProcessCpuLoad > 0.2**).
- Constraints can be grouped together by using logical operators such as **&&** (AND), || (OR), or **!** (NOT) logic. For readability and clarity around the order of operations and operator precedence, grouped constraints may be enclosed in round brackets, but this is not a requirement. For example:

  ```
  [(MetricA > value1 && MetricB < value2) || MetricC == 'stringvalue' ; TargetDuration > duration("30s")]
  ```

- The name of each MBean counter that is specified as part of a custom trigger must follow precise syntax rules in terms of spelling and capitalization. For a full list of the MBean metrics that you can specify, see MBean counter types.
- Only one target duration can be defined for a custom trigger. The target duration is applied to the entire trigger expression that is enclosed within the square brackets.
- A target duration can be expressed in terms of seconds, minutes, or hours. For example, **30s** means 30 seconds, **5m** means five minutes, **2h** means two hours, and so on.
- A target duration is optional. If a target duration is not specified, triggering will occur immediately once the trigger conditions are met.
- Multiple custom trigger definitions can be specified together, each of which relates to a separate JFR recording. Different custom trigger definitions must be separated by a comma (**,**) character. For example:

  ```
  [ProcessCpuLoad>0.2]~profile,[ThreadCount>30]~Continuous
  ```

CHAPTER 3. AUTOMATIC STARTING OF A DYNAMIC JFR RECORDING

When the Cryostat agent is enabled to start JFR recordings and the custom trigger condition for a dynamic recording is met, the Cryostat agent automatically starts the recording from within the target application.

The Cryostat agent automatically assigns a name to the JFR recording, which is always in a **cryostat-smart-trigger-*X*** format, where ***X*** represents the recording ID. The JVM automatically generates the recording ID, which is an incremental numeric value that is unique for each JFR recording that is started within the JVM.

When the Cryostat agent starts a dynamic JFR recording, you can subsequently view this recording in the **Active Recordings** tab in the Cryostat web console. For more information about using the **Active Recordings** tab, see Creating a JFR recording with Cryostat .

CHAPTER 4. MANUAL STOPPING OF A DYNAMIC JFR RECORDING

The Cryostat agent does not currently support the automatic stopping of dynamic JFR recordings. In this release, a dynamic JFR recording does not stop even if the conditions that triggered the recording are no longer being met. In this situation, if you want a dynamic JFR recording to stop, you must stop the recording manually from the **Active Recordings** tab in the Cryostat web console.

For more information about using the Cryostat web console to stop JFR recordings, see Creating a JFR recording with Cryostat.

CHAPTER 5. MULTIPLE JFR RECORDINGS BASED ON THE SAME CUSTOM TRIGGER DEFINITION

The Cryostat 2.4 agent can dynamically start a JFR recording for each custom trigger definition only once. In this release, the Cryostat agent cannot start multiple JFR recordings for the same custom trigger condition on a recurring basis. Once the Cryostat agent starts a JFR recording for a specific custom trigger definition, the agent then ignores this trigger definition for the rest of the agent session.

In this situation, if you want to enable the Cryostat agent to start new JFR recordings based on custom trigger conditions that previously triggered a recording, you must restart the Cryostat agent.

CHAPTER 6. INTEGRATION OF DYNAMIC JFR RECORDINGS WITH THE AGENT HARVESTER FOR ARCHIVING

When you enable the Cryostat agent to start dynamic JFR recordings based on MBean custom triggers, you can also integrate these JFR recordings with the agent harvester system. This integration means that any JFR recording data resulting from MBean custom triggers is periodically captured in JFR snapshots and pushed to the Cryostat server for archiving, based on the harvester's configured schedule.

MBean custom triggers with agent harvester periods

The agent harvester is another configurable feature, which enables the Cryostat agent to start JFR recordings automatically at agent startup based on a given event template. The agent harvester includes a configurable property that you can use to define a schedule for capturing and uploading recording snapshots to the Cryostat server.

By defining MBean custom triggers and an agent harvester period without a harvester template, you can achieve a setup where the agent does both of the following:

- Agent dynamically starts JFR recordings based on MBean custom triggers.
- Aent uses configured harvester periods to periodically capture snapshots of the recording data and upload this data to the Cryostat server.

In this situation, the agent continues to capture recording data until you manually stop the dynamic JFR recording or the host JVM shuts down.

Configuration of an agent harvester period

When you configure your target application to load the Cryostat agent, you can also configure an agent harvester period to enable regular uploads of JFR recording data. You can specify the value of the harvester period in milliseconds. By default, the Cryostat agent is not enabled to perform any scheduled harvest uploading of JFR recording data.

You can configure an agent harvester period in either of the following ways:

Using a JVM system property flag

The following example shows how to configure a harvester period by using a JVM system property flag:

```
-Dcryostat.agent.harvester.period-ms=1000
```

Using an environment variable

The following example shows how to configure a harvester period by using an environment variable:

```
- name: CRYOSTAT_AGENT_HARVESTER_PERIOD_MS
  value: 1000
```

The preceding examples show a harvester period value of 1000. Based on this example, the agent uploads JFR recording data for archiving every 1000 milliseconds (that is, at regular 1-second intervals).

NOTE

Cryostat supports each of the following different ways to start JFR recordings:

- You can start recordings manually from the Cryostat web console.
- The Cryostat agent can start recordings dynamically based on MBean custom triggers.
- The Cryostat agent can start recordings automatically at agent startup based on a given harvester template.
- The Cryostat server can send on-demand requests over JMX or an agent HTTP connection to start recordings based on automated rules.

In this situation, agent harvester settings control the capture and upload of all JFR recording data regardless of which way JFR recordings are started in the system.

CHAPTER 7. CONFIGURATION OF AN EVALUATION PERIOD FOR CUSTOM MBEAN TRIGGERS

The Cryostat agent supports smart triggers that continually listen to the current values of the specified MBean counters that you can define in custom trigger definitions. The trigger conditions are evaluated on a polling basis at regular configurable intervals. By default, trigger conditions are evaluated at regular 1-second intervals.

An evaluation period (polling frequency) of once per second means that there is a potential time delay of up to one second between a condition being met and the agent's ability to evaluate that this condition was met.

When you configure your target application to load the Cryostat agent, you can choose to configure a different evaluation period for MBean custom triggers. You can specify the value of the evaluation period in milliseconds.

You can configure an evaluation period in either of the following ways:

Using a JVM system property flag

The following example shows how to configure an evaluation period by using a JVM system property flag:

```
-Dcryostat.agent.smart-trigger.evaluation.period-ms=500
```

Using an environment variable

The following example shows how to configure a harvester period by using an environment variable:

```
- name: CRYOSTAT_AGENT_SMART-TRIGGER_EVALUATION_PERIOD_MS
  value: 500
```

The preceding examples show an evaluation period value of 500. Based on this example, the trigger conditions are evaluated every 500 milliseconds (that is, at regular half-second intervals).

CHAPTER 8. MBEAN COUNTER TYPES

The MBean counter types that you can specify as part of a custom trigger definition are standard MBean metrics that are generally available in the JDK by default. These MBean counters cover a range of runtime, memory, thread, and operating system metrics.

The following is a full list of the MBean counter types that you can specify in a custom trigger definition:

NOTE

The name of each MBean counter that you specify as part of a custom trigger definition must exactly match the spelling and capitalization of the MBean counter names in the following list.

Operating system metrics

- Arch
- AvailableProcessors
- Name
- SystemLoadAverage
- Version
- CommittedVirtualMemorySize
- FreePhysicalMemorySize
- FreeSwapSpaceSize
- ProcessCpuLoad
- ProcessCpuTime
- SystemCpuLoad
- TotalPhysicalMemorySize
- TotalSwapSpaceSize

Thread metrics

- AllThreadIds
- CurrentThreadCpuTime
- CurrentThreadUserTime
- DaemonThreadCount
- PeakThreadCount
- ThreadCount

- TotalStartedThreadCount
- CurrentThreadCpuTimeSupported
- ObjectMonitorUsageSupported
- SynchronizerUsageSupported
- ThreadContentionMonitoringEnabled
- ThreadContentionMonitoringSupported
- ThreadCpuTimeEnabled
- ThreadCpuTimeSupported

Runtime metrics

- BootClassPathSupported
- BootClassPath
- ClassPath
- InputArguments
- LibraryPath
- ManagementSpecVersion
- Name
- SpecName
- SpecVersion
- SystemProperties
- StartTime
- Uptime
- VmName
- VmVendor
- VmVersion

Memory metrics

- HeapMemoryUsage
- NonHeapMemoryUsage
- ObjectPendingFinalizationCount
- FreeHeapMemory

- FreeNonHeapMemory
- HeapMemoryUsagePercent
- Verbose

Revised on 2023-12-08 09:01:56 UTC