Red Hat build of Cryostat 2

配置高级 Cryostat 配置

Last Updated: 2024-01-03

法律通告

摘要

配置红帽构建的 Cryostat 高级功能，以便您可以自定义 Cryostat 以满足您的要求。配置高级 Cryostat 配置 文档描述了如何使用 API 将外部插件注册到 Cryostat，以便您可以更好地将 Cryostat 与部署模式集成。

目录

前言

Red Hat build of Cryostat 是 JDK Flight Recorder (JFR)的一个容器原生实现，可用于安全地监控在 OpenShift Container Platform 集群上运行的工作负载中的 Java 虚拟机(JVM)性能。您可以使用 Cryostat 2.4 使用 web 控制台或 HTTP API 启动、停止、检索、存档、导入和导出容器化应用程序中 JVM 的 JFR 数据。

根据您的用例，您可以使用 Cryostat 提供的内置工具直接在 Red Hat OpenShift 集群上存储和分析记录，或者您可以将记录导出到外部监控应用程序，以对记录的数据进行更深入的分析。

重要

红帽构建的 Cryostat 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

使开源包含更多

红帽承诺替换我们的代码、文档和网页属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 的信息。

第 1 章 可插拔发现 API

您可以使用可插拔发现 API 端点 **/api/v2.2/discovery**，将外部插件注册到 Cryostat，并提供有关可发现的应用程序目标的信息。

注意

作为可插拔发现 API 的替代选择，您可以使用 Cryostat 代理作为 Cryostat 发现插件。Cryostat 代理作为 Java Instrumentation Agent 实施，该代理充当 JVM 上运行的应用程序的插件。Cryostat 代理提供了一个 HTTP API，它因为代理的双重角色作为发现插件提供了比 JMX 端口更大的部署灵活性。您可以将目标应用程序配置为使用代理的 HTTP API 来通过 Cryostat 检测和连接。有关将目标应用程序配置为使用 Cryostat 代理的更多信息，请参阅配置 Java 应用程序。

1.1. 可插拔发现 API 概述

您可以使用可插拔发现 API 端点 **/api/v2.2/discovery**，将外部插件注册到 Cryostat，然后提供有关可发现的应用程序目标的信息。在成功注册 Cryostat 后，一个插件可以取消注册自身，或持续将更新推送到有关目标应用程序的 Cryostat。

注意

注册操作的目的是增强 Cryostat 安全性，并保持插件和 Cryostat 之间的数据一致性。

可插拔的 Discovery API 提供了更灵活的方法，用于将 Cryostat 集成到 Red Hat OpenShift 服务帐户机制的部署模式中。

例如，您需要编写插件程序来创建应用程序 IP 地址到端口号的静态映射。该插件可以使用可插拔发现 API 将此信息传送到 Cryostat，以便 Cryostat 能够更好地与目标应用程序连接。

Pluggable Discovery API 依赖于以下处理程序来管理从 **/api/v2.2/discovery** 端点和 Cryostat 发送的请求：

- **DiscoveryRegistrationHandler**
- **DiscoveryRegistrationCheckHandler**
- **DiscoveryGetHandler**
- **DiscoveryPostHandler**
- **DiscoveryDeregistrationHandler**

注意

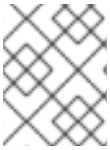
有关这些处理程序的更多信息，请参阅后续文档部分。

在使用 API 的端点与 Cryostat 交互前，您必须确保客户端（即插件的来源）符合以下先决条件：

- 接受 JSON 响应。
- 可以将 HTTP 请求发送到 Cryostat。
- 可以在 **POST** 请求的 Authorization 标头中输入正确的 Cryostat 凭证。

- 可以从 Cryostat 接收 **GET** 和 **POST** 请求。
- 通过将 **POST** 请求发送到 Cryostat，发布 JSON 中发现目标的信息。

如果需要使用 Cryostat 注册您自己的发现插件程序，您可以禁用 Cryostat 内置发现机制。禁用内置发现机制有助于减少在 Cryostat web 控制台中打开的重复定义（如果插件和 Cryostat 都可以访问相同的目标应用程序信息）。



注意

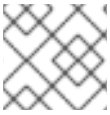
如果 Cryostat 检测到两个类似的定义指向同一 JVM，C Cryostat 会将任何归档的记录存储在每个定义访问的同一存储位置。

您可以在 Red Hat OpenShift 命令行控制台(CLI)中设置以下环境变量来禁用 Cryostat 内置发现机制：

```
CRYOSTAT_DISABLE_BUILTIN_DISCOVERY=true
```

您还可以考虑保留 Cryostat 内置发现实现的设置，然后完成以下操作之一：

- 创建一个将服务定位器附加到实施的程序。
- 修改目标应用程序以将目标信息直接发送到实施。



注意

之前列出的操作超出了 *配置高级 Cryostat 配置* 文档的范围。

1.2. DISCOVERYREGISTRATIONHANDLER

Pluggable Discovery API 使用 **DisoveryRegistrationHandler** 注册发现插件的 Cryostat。该处理程序管理来自插件和 Cryostat 的 **GET** 和 **POST** 请求。如果您没有使用 Cryostat 注册插件，则插件无法向 Cryostat 提供目标信息。

当发现插件程序向 Cryostat 发送 **POST** 请求以注册时，C Cryostat 会读取 **回调** URL，并将 **GET** 请求发送到插件。如果插件正确响应 **GET** 请求，则 Cryostat 会响应初始 **POST** 请求来接受注册请求。这个过程确保插件处于活跃状态，并可用于 Cryostat。

失败的请求可能表示插件失败或离线。在这种情况下，端点会从 Red Hat OpenShift 上的数据库中删除插件的信息。

注意

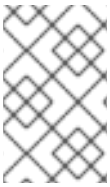
注册过程后，Cryostat 将定期 **POST** 请求发送到插件，以确保插件仍在运行。

您还可以在 **GET** 或 **POST** 请求中指定 **id 和令牌** 元素。这些元素是可选的，但您可以在想要为之前使用 Cryostat 注册的插件重复使用注册信息的情况。

Cryostat 为注册插件创建一个令牌，此令牌包含到期和授权信息。如果 **POST** 请求包含有效的 **id 和令牌** 信息，C Cryostat 可以重复使用插件注册信息并刷新令牌。如果请求只包含 **id** 元素 **或令牌** 元素，则必须将插件重新注册到 Cryostat。

在 Cryostat 发送 **POST** 请求到插件 **的回调** 组件后，该插件可能会向 Cryostat 发送 **POST** 请求以刷新插件的注册详情。该插件必须将其 **id 和令牌** 信息包含在其请求中。然后 Cryostat 可以使用 **DiscoveryRegistrationHandler** 刷新插件的详细信息。Cryostat 发送包含更新令牌到插件的响应，插件

就可以使用这个令牌到 Cryostat。



注意

Cryostat 定期向插件发出 **POST** 请求，以提醒插件使用相同 **id 和令牌** 信息重新注册到 Cryostat。如果插件忽略了这个请求，令牌可能会过期，插件必须完成具有 Cryostat 的完整注册。

来自外部插件的 POST 请求示例

```
{
    "realm": "my-plugin",
    "callback": "http://my-plugin.my-namespace.svc.local:1234/callback"
}
```

Cryostat 发送到插件的 POST 响应示例

```
{
"data": {
    "result": {
        "id": "922dd4f4-9d7c-4ae2-8982-0903868226a6",
        "token": "<key_value>"
    }
},
"meta": {
    "status": "Created",
    "type": "application/json"
}
```

1.3. DISCOVERYREGISTRATIONCHECKHANDLER

可插拔发现 API 使用 **DisoveryRegistrationCheckHandler** 处理程序使插件在 Cryostat 服务器实例上定期检查自己的注册状态。

DisoveryRegistrationCheckHandler 处理程序管理插件到 Cryostat 的 **GET** 请求。通过使用处理程序，外部插件可以定期验证已注册到的 Cryostat 服务器实例，它们仍然处于活动状态，并识别插件之前注册。

与 Cryostat **回调** URL 端点检查插件实例的方式类似，其中 Cryostat 读取 **回调** URL 并向插件发送 **GET** 请求，**DisoveryRegistrationCheckHandler** 处理程序的工作方式相同，但以相反方向发送请求。也就是说，插件向 Cryostat 服务器发送 **GET** 请求，以检查 Cryostat 服务器上的注册状态。如果请求失败，例如，如果收到了 **未预期的 401** 或 **Unexpected 404** 错误响应，插件可以丢弃它现有的注册信息，并尝试再次注册。

从外部插件发送的 GET 请求示例

```
$ http -v https://my-cryostat.my-namespace.cluster.local:8181/api/v2.2/discovery/<plugin-registration-id>?token=<current-plugin-registration-token>
```

当 GET 请求检查成功且 Cryostat 识别插件的当前注册时，Cryostat 响应示例

```
HTTP/1.1 200 OK
```

```
content-encoding: gzip
content-length: 86
content-type: application/json

{
 "data": {
    "result": null
 },
 "meta": {
    "mimeType": "JSON",
    "status": "OK"
 }
}
```

当 GET 请求检查失败时，GET 请求检查失败时的 Cryostat 响应示例，因为 Cryostat 无法识别插件注册详情

```
HTTP/1.1 404 Not Found
content-encoding: gzip
content-length: 95
content-type: application/json

{
 "data": {
    "reason": null
 },
 "meta": {
    "status": "Not Found",
    "type": "text/plain"
 }
}
```

1.4. DISCOVERYGETHANDLER

DiscoveryGetHandler 封装部署模式并在分层树视图中打开这些架构，以便 Cryostat 可以与任何已注册的可发现插件交互，以便与特定的部署模式集成。

当您创建至少一个 pod 的部署，服务映射到 Red Hat OpenShift 上的部署或 pod 时，Red Hat OpenShift 会为 **Pod IP** 地址和端口的所有组合创建一个端点对象 **/api/v2.2/discovery**。**DiscoveryGetHandler 从端点 GET 请求接收信息，然后以 JSON 格式合作部署模式信息。**

以下示例演示了处理程序如何在 JSON 格式以分级树视图中打开。在示例中，树根是 UNIVERSE 节点。此节点包含子 Realm 节点类型，它源自 Cryostat 的内置发现机制以及 Pluggable Discovery API 发现的插件。

```
{
   "data": {
      "result": {
         "children": [
            {
               "children": [],
```

```
            "labels": {},
            "name": "Custom Targets",
            "nodeType": "Realm"
          },
          {
            "children": [
              {
                "labels": {},
                "name": "service:jmx:rmi:///jndi/rmi://cryostat:9091/jmxrmi",
                "nodeType": "JVM",
                "target": {
                  "alias": "io.cryostat.Cryostat",
                  "annotations": {
                    "cryostat": {
                      "HOST": "cryostat",
                      "JAVA_MAIN": "io.cryostat.Cryostat",
                      "PORT": "9091",
                      "REALM": "JDP"
                    },
                    "platform": {}
                  },
                  "connectUrl": "service:jmx:rmi:///jndi/rmi://cryostat:9091/jmxrmi",
                  "labels": {}
                }
              }
            ],
            "labels": {},
            "name": "JDP",
            "nodeType": "Realm"
          }
        ],
        "labels": {},
        "name": "Universe",
        "nodeType": "Universe"
      }
    },
    "meta": {
      "status": "OK",
      "type": "application/json"
    }
  }
```


1.5. DISCOVERYPOSTHANDLER

通过 Cryostat 注册的插件会发送 POST /api/v2.2/discovery/:id 请求到 DiscoveryPostHandler。请求的 id 参数引用注册插件的 ID。此处理程序处理与插件关联的任何子树，然后映射部署架构。

Cryostat POST 请求定义了一个子树 REALM 节点，因此插件处理程序可以在请求过程中在 REALM 节点中发布其子节点类型。该插件负责为Cryostat REALM 节点提供正确的信息，并将节点类型放入正确的子树位置。该插件必须为令牌提供它发送到 DiscoveryPostHandler 的 POST 请求，以便插件可以绕过之前通过的授权标头检查。

注意

在插件将更新发送到 Cryostat 后，Cryostat 会替换之前插件发送的信息。Cryostat 将此信息存储在数据库中。插件必须为每个请求发送可发现目标的完整列表或树立到 Cryostat。

一个插件的 POST 请求示例，它详细介绍了重要目标应用程序信息

```
[
  {
    "labels": {},
    "nodeType": "JVM",
    "name": "service:jmx:rmi:///jndi/rmi://myapp.svc.local:9091/jmxrmi",
    "nodeType": "JVM",
    "target": {
      "alias": "com.MyApp",
      "annotations": {
        "cryostat": {},
        "platform": {}
    },
      "connectUrl": "service:jmx:rmi:///jndi/rmi://myapp.svc.local:9091/jmxrmi",
      "labels": {}
    }
  }
]
```

对插件 POST 请求的 Cryostat 响应示例

```
{
  "data": {
    "result": null
  },
  "meta": {
    "mimeType": "JSON",
    "status": "OK"
  }
}
```

1.6. DISCOVERYDEREGISTRATIONHANDLER

如果一个插件使用 Cryostat 作为 插件 注册，则插件需要关闭，插件通常会向 un-register 本身作为一个 插件 发出请求。Cryostat 发送 DELETE /api/v2.2/discovery/:id?token=:token 请求到 DiscoveryDeregistrationHandler，然后从 Cryostat 取消注册插件。

注意

Cryostat 在注册过程后将常规 POST 请求发送到插件，以确保插件仍在运行。如果插件不响应其中一个请求，C Cryostat 会启动插件取消注册的进程。

该处理程序从发现模式删除插件的 REALM 子树，以及插件的注册信息（来自 Cryostat）。

DELETE /api/v2.2/discovery/:id?token=:token 请求处理程序进程示例

```
{
    "data": {
      "result": "bcc0f3a6-dc48-402e-a3d6-9fbb63beff78"
    },
    "meta": {
      "mimeType": "JSON",
      "status": "OK"
    }
}
```

1.7. 错误处理程序代码

当插件通过 Cryostat 注册插件时，可插拔发现 API 会返回消息。

当处理程序试图根据 GET 和 POST 请求注册带有 Cryostat 的插件时，处理程序可以返回以下消息类型：

- 200 ：处理程序成功完成任务。例如，DiscoveryDeregistrationHandler 返回 JSON 格式的消息，带有消息的 id 元素中定义的 unregistered 插件。
- 400 ：JSON 文档结构无效，或者 id 元素采用无效格式编写。

- **401: 插件没有传递授权标题步骤进行注册。如果令牌已过期或您取消注册了插件，处理程序也会返回这个错误消息。**

- **404: 插件 id 元素未找到。插件可能具有 回调检查失败。考虑重新注册插件。**

第 2 章 GRAPHQL API

GraphQL API 端点 /api/v2.2/graphql 将自动针对目标 JVM 运行更短且更简单的查询。这些查询可以针对目标 JVM 的活跃记录和存档记录运行。另外，API 可以针对常规 Cryostat 归档运行查询。您可以自定义查询，为活跃记录或归档记录自动执行以下任务：

您可以自定义查询，为活跃记录或归档记录自动执行以下任务：

- 归档
- 删除
- Start
- stop

在创建自定义查询时，您必须在查询中为 GraphQL API 端点指定特定信息。POST 请求然后可以处理信息并将信息发送到 Cryostat。以下示例指定 Graph QL API 的信息：

```
POST /api/beta/graphql HTTP/1.1
Accept: application/json, /;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 171
Content-Type: application/json
Host: localhost:8181
User-Agent: HTTPie/3.1.0
```

GraphQL API 比 HTTP REST API 强大，具有有限的工作负载功能。例如，HTTP REST API 要求您为每个您要在 OpenShift 中的容器内的扩展副本上启动的记录副本创建一个 API 请求。GraphQL API 在一个 API 请求中实现此任务，这提高了 API 的性能，并减少 Cryostat 实例的任何网络流量。

HTTP REST API 的另一个有限工作负载功能是，此 API 需要更多的用户干预，例如，您需要编写自定义客户端以在对响应数据执行迭代操作时解析 API JSON 响应。GraphQL API 不要求您完成此操作。

其他资源

请参阅 GraphQL 简介 (GraphQL)

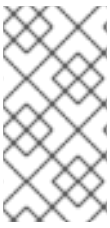
2.1. 使用 GRAPHQL API 创建自定义查询创建

您可以使用 HTTP 客户端（如 HTTPie ）与 GraphQL API 交互来生成自定义查询。

当使用 API 创建查询时，您必须为数据类型和字段指定特定值，以便函数可以使用这些值来准确定位所需数据。

考虑一个用例，您可以在其中自动执行在单个查询中执行多个操作的工作流。例如，HTTPie 客户端请求可能会向 Cryostat 发送查询，以便 Cryostat 可以执行以下任务：

1. 记录所有目标 JVM 应用。
2. 将每个应用程序中的记录信息复制到 Cryostat 归档中。
3. 创建一个自动规则，针对任何检测到的目标 JVM 应用自动启动持续监控记录。



注意

有很多查询可能性，因此请确保准确确定查询的数据类型和字段的值。否则，您可能会获得与要求不匹配的查询结果。

以下示例演示了使用 HTTP 客户端与 GraphQL API 交互，以在 Cryostat 数据上生成简单查询和复杂的查询。

确定与 Cryostat 实例交互的所有已知目标 JVM 的简单查询

```
$ https :8181/api/v1/targets
HTTP/1.1 200 OK
content-encoding: gzip
content-length: 223
content-type: application/json
```

```
{
targetNodes {
    name
    nodeType
    labels
    target {
      alias
      serviceUri
    }
}
}
```

上例为 **targetNodes** 元素设置特定的值，如名称。运行查询后，查询会返回与指定条件匹配的任何目标 **JVM** 列表。

确定属于特定 pod 的 Cryostat 实例可见的所有目标应用程序的复杂查询示例

```
$ https -v :8181/api/v2.2/graphql query=@graphql/target-nodes-query.graphql
POST /api/v2.2/graphql HTTP/1.1
Accept: application/json, /;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 171
Content-Type: application/json
Host: localhost:8181
User-Agent: HTTPie/2.6.0

{
environmentNodes(filter: { name: "<application_pod_name>" }) {
    descendantTargets {
      doStartRecording(recording: {
        name: "myrecording",
       template: "Profiling",
       templateType: "TARGET",
        duration: 30
       }) {
       name
       state
      }
    }
}
}
```

上例为 environmentNodes 元素设置以下特定值，示例不会返回 JVM 目标应用：

- <application_pod_name > ：确保查询以与 Cryostat 相同的命名空间中运行的特定 pod 为目标。
- descendantTargets: 提供 JVM 目标对象数组。
- doStartRecording: GraphQL API 为查询列表的每个目标 JVM 启动 JFR 记录。

运行查询后，查询会返回有关您启动的 JFR 记录的信息，如活跃应用程序节点列表。

对于 Red Hat OpenShift，此查询会返回 Deployment 和 DeploymentConfig API 对象，以及与 Cryostat 交互的 pod。

复杂的查询演示了 GraphQL API 如何执行单个 API 请求，该请求返回与 Cryostat 和 Red Hat OpenShift 交互的任何 JVM 对象。然后，API 请求会在这些返回的对象上启动 JFR 记录。

第 3 章 JMC 代理插件

您可以使用 JMC 代理插件将 JMC 代理实现添加到 Cryostat。然后，您可以使用 JMC 代理将自定义 JFR 事件添加到正在运行的目标 JVM 应用中。此操作不要求您重新启动 JVM 应用。

其他资源

- 请参阅在 Red Hat build of Cryostat 中使用 JDK Flight Recorder

3.1. 使用 JMC 代理插件添加自定义事件

Cryostat 与 JMC Agent 相结合，您可以在需要诊断正在运行的 JVM 应用问题时为您提供更多信息。

JMC Agent JAR 文件必须与目标 JVM 应用程序位于同一个 Red Hat OpenShift 容器中。否则，Cryostat 无法在应用程序中使用 JMC Agent 功能。

从 Cryostat Web 控制台，您可以上传探测模板，然后将这些模板插入到 JVM 应用中。如果需要，您可以在稍后阶段删除这些模板探测。探测模板描述了 Cryostat 可以处理的一组对象，以便 Cryostat 可以完成 JVM 应用中的 JMC Agent 操作序列。

当使用 JMC Agent 启动目标 JVM 应用程序时，Cryostat 会自动检测应用程序是否使用 JMC Agent 运行。

重要

对于 RHEL，JMC 软件包由 CodeReady Linux Builder (CRB)提供，也称为 *Builder*，存储库。您必须在 RHEL 上启用 CRB 存储库，以便在 RHEL 上安装 JMC。CRB 软件包使用 Source Red Hat Package Manager (SRPM)作为产品化的 RHEL 软件包构建，因此 CRB 软件包会定期接收更新。请参阅 使用 Red Hat build of Cryostat 下载并安装 JDK Mission Control (JMC) （使用 JDK Flight Recorder with Red Hat build of Cryostat）

前提条件

- 下载并安装 jmc 软件包。

- 下载 Adoptium Agent JAR 文件。请参阅 adoptium/jmc-build (GitHub)。

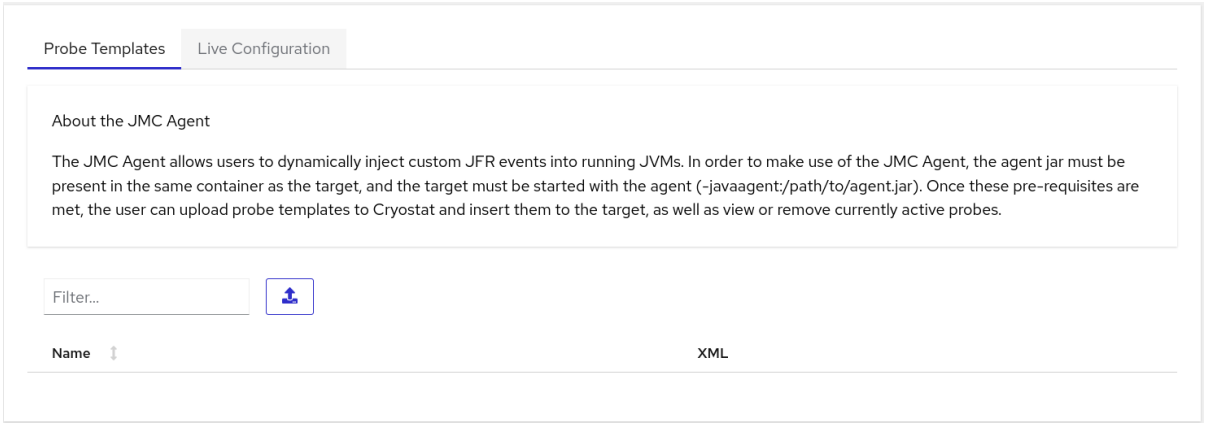
- 使用 --add-opens=java.base/jdk.internal.misc=ALL-UNNAMED 标志启动您的 Java 应用程序。例如，. /<your_application> --add-opens=java.base/jdk.internal.misc=ALL-UNNAMED。

- 为您的 Java 应用程序启动 JMC 代理。请参阅 启动 JDK Mission Control (JMC)代理 （使用带有红帽构建的 Cryostat 的 JDK Flight Recorder）。

流程

1. 从 Cryostat Web 控制台，进入 Events 菜单。如果 JMC Agent 成功添加到 Cryostat 实例中，则在 Event Templates 窗格下打开一个 Probe Templates 窗格。

图 3.1. Cryostat web 控制台中的探测模板标签页



注意

在 web 控制台中可能会打开一个身份验证所需的 对话框。若有提示，在 Authentication Required 对话框中输入您的 Username 和 Password，然后单击 Save 将您的 JMX 凭证提供给目标 JVM。

2. 使用您的首选文本编辑器创建 XML 配置文件。使用 JFR 事件信息填充该文件，如在应用上必须执行哪些事件 Cryostat。

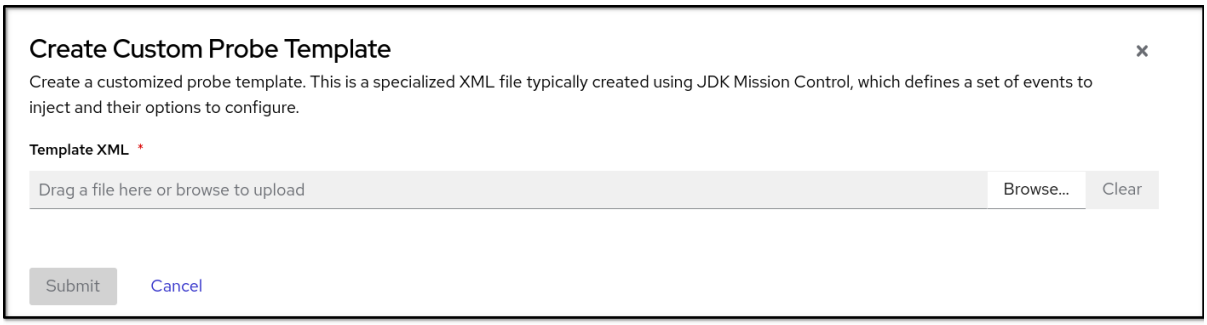
以下示例显示了包含 JFR 事件信息的自定义探测模板 XML 文件。将文件上传到 Cryostat 时，Cryostat 可以向应用程序添加名为 Cryostat Agent Plugin Demo Event 的自定义 JFR 事件。当调用 JMC Agent 的 retrieveEventProbes 方法时，Cryostat 会启动 JFR 事件。

```
<jfragent>
```

```
<!-- Global configuration options -->
<config>
 <classprefix>__JFREvent</classprefix>
 <allowtostring>true</allowtostring>\
 <allowconverter>true</allowconverter>
</config>
 <events>
  <event id="cryostat.demo.jfr.event9">
     <label>Cryostat Agent Plugin Demo Event</label>
     <description>Event for the agent plugin demo</description>
     <path>io/cryostat/demo/events</path>
     <stacktrace>true</stacktrace>
     <class>io.cryostat.core.agent.AgentJMXHelper</class>
     <method>
        <name>retrieveEventProbes</name>
   <descriptor>()Ljava/lang/String;</descriptor>
     </method>
    <location>WRAP</location>
  </event>
</events>
</jfragent>
```

3. 单击 **Upload** 按钮，将自定义事件模板添加到 **Cryostat**。**Create Custom Probe Template** 在 **Cryostat web** 控制台中打开。

图 3.2. Cryostat Web 控制台中的 Create Custom Probe Template 窗口



提示

如果要从此模板 XML 字段删除上传的文件，请单击 Clear 按钮。

4. **点 Browse 按钮找到您的 XML 文件。**

5. **上传文件后，单击 Submit。自定义探测模板文件会在 Probe Templates 表中打开。**

6. **点探测模板旁的 overflow 菜单。**

7. 点 Insert Probes。探测器显示在探测 模板 选项卡下的表中，以及 Live Configuration 选项卡下的表。

8. *可选：* 使用 Live Configuration 选项卡，您可以在其中查看每个活跃探测的名称、类 等信息。

9. *可选：* 在 Live Configuration 选项卡中，您可以点击 Remove All Probes 删除表中列出的探测。

验证

1. 在 Events 菜单中点 Event Types 选项卡。

2. 检查您的 XML 配置中的命名 JFR 事件是否在表中列出。对于此流程中使用的示例，表格中会显示 Cryostat Agent Plugin Demo Event。

其他资源

- 请参阅在 Red Hat build of Cryostat 中使用 JDK Flight Recorder

更新于 2024-01-02