



Red Hat build of Cryostat 3

配置高级 Cryostat 配置

法律通告

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

配置 Red Hat build of Cryostat 的高级功能集合，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 与部署架构集成。

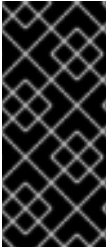
目录

前言	3
使开源包含更多	4
第 1 章 可插拔发现 API	5
1.1. 可插拔发现 API 概述	5
1.2. 发现注册端点	6
1.3. 发现注册检查端点	7
1.4. 发现 GET 端点	8
1.5. 发现 POST 端点	9
1.6. 发现取消注册端点	10
1.7. 错误代码	11
第 2 章 GRAPHQL API	13
2.1. 使用 GRAPHQL API 创建自定义查询	14
第 3 章 JMC 代理插件	17
3.1. 使用 JMC 代理插件添加自定义事件	17

前言

Red Hat build of Cryostat 是 JDK Flight Recorder (JFR)的容器原生虚拟化，可用于安全监控在 OpenShift Container Platform 集群上运行的工作负载的 Java 虚拟机(JVM)性能。您可以使用 Cryostat 3.0 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中的 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存储和分析 Red Hat OpenShift 集群上的记录，或者您可以将记录导出到外部监控应用程序，以对记录数据进行更深入分析。



重要

Red Hat build of 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

红帽致力于替换我们的代码、文档和 Web 属性中有问题的语言。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 Discovery API 端点 `/api/ <version> /discovery`，使用 Cryostat 注册外部插件，并提供有关可发现应用程序目标的信息到 Cryostat。



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，它提供比 JMX 端口更大的部署灵活性，因为代理的双角色作为发现插件。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 Discovery API 端点 `/api/ <version> /discovery`，使用 Cryostat 注册外部插件，然后提供有关可发现应用程序目标的信息到 Cryostat。在成功注册使用 Cryostat 后，插件可以取消注册，或持续将更新推送到 Cryostat 的相关目标应用程序。



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。

可插拔发现 API 提供了比 Red Hat OpenShift 服务帐户机制将 Cryostat 集成到部署架构中更灵活的方法。

例如，您需要编写一个创建应用程序 IP 地址静态映射到端口号的插件程序。该插件可以使用可插拔的 Discovery 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。

可插拔发现 API 使用以下端点来管理从 `/api/v2.2/discovery` 端点和 Cryostat 发送的请求：

- Discovery registration 端点
- 发现注册检查端点
- 发现 **POST** 端点
- 发现取消注册端点

可插拔发现 API 还使用 Discovery **GET** 端点来管理从 `/api/v3/discovery` 端点和 Cryostat 发送的 **GET** 请求。



注意

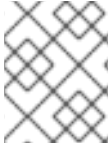
Discovery **GET** 端点仅适用于 `/api/v3/discovery`。所有其他发现端点都可用于 `/api/v2.2/discovery`。有关这些发现端点的更多信息，请参阅后续文档部分。

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

- 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 来发布 JSON 中可发现目标的信息。

如果您需要使用 Cryostat 注册自己的发现插件程序，您可以使用 Cryostat CR 中的 `spec.targetDiscoveryOptions.disableBuiltInDiscovery` `Options.disableBuiltInDiscovery` 字段禁用 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，则 Cryostat 会将任何存档记录存储在每个定义访问的同一存储位置。

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：

- 创建将服务 locator 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。

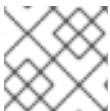


注意

之前列出的操作超出了 *配置高级 Cryostat 配置* 文档的范围。

1.2. 发现注册端点

可插拔发现 API 使用 Discovery 注册端点，将发现的插件注册到 Cryostat。此端点管理来自插件和 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。



注意

Discovery 注册端点仅适用于 `/api/v2.2/discovery`。

当您发现插件程序向 Cryostat 发送 **POST** 请求以注册目的时，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 通过响应初始 **POST** 请求来接受注册请求。此过程可确保插件处于活跃状态并可用于 Cryostat。

失败的请求可能会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。



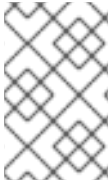
注意

注册过程后，Cryostat 会将常规 **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 `token` 元素。这些元素是可选的，但您可以在希望为之前使用 Cryostat 注册的插件重复使用信息时考虑使用它们。

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 `token` 信息，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 `token` 元素，则必须使用 Cryostat 重新注册插件。

在 Cryostat 将 **POST** 请求发送到插件的回调组件后，插件可能会向 Cryostat 发送 **POST** 请求，以刷新插件的注册详情。该插件必须在请求中包含其 **id** 和 **token** 信息。然后 Cryostat 可以使用 Discovery 注册端点来刷新插件的详细信息。Cryostat 发送包含更新令牌到插件的响应，插件可以使用此令牌来将来的请求到 Cryostat。



注意

Cryostat 可以定期向插件发出 **POST** 请求，以提醒有关使用相同的 **id** 和 **token** 信息重新注册带有 Cryostat 的插件。如果插件忽略这个请求，令牌可能会过期，插件必须使用 Cryostat 完成完整注册。

从外部插件发送的 POST 请求示例

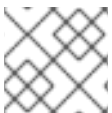
```
{
  "realm": "my-plugin",
  "callback": "http://my-plugin.my-namespace.svc.local:1234/callback"
}
```

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. 发现注册检查端点

可插拔发现 API 使用 Discovery 注册检查端点，以启用插件在 Cryostat 服务器实例上定期检查自己的注册状态。



注意

Discovery registration check 端点仅适用于 `/api/v2.2/discovery`。

Discovery registration check 端点管理从插件到 Cryostat 的 **GET** 请求。通过使用此端点，外部插件可以定期验证它们注册的 Cryostat 服务器实例是否仍然活跃，并识别插件之前的注册。

与 Cryostat 回调 URL 端点检查插件实例类似，其中 Cryostat 读取回调 URL 并将 **GET** 请求发送到插件，发现注册检查端点的工作方式相同，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 **Unexpected 401** 或 **Unexpected 404** 错误响应，插件可以丢弃其现有的注册信息，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** 请求检查成功并且 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"
  }
}
```

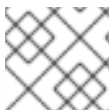
GET 请求检查失败时的 Cryostat 响应示例，因为 Cryostat 无法识别插件注册详情

```
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. 发现 **GET** 端点

Discovery **GET** 端点协调部署模式，并在分级树视图中打开这些模式，以便 Cryostat 可以与任何注册的可发现插件交互，以与特定部署架构集成。



注意

Discovery **GET** 端点仅适用于 `/api/v3/discovery`。

当您创建至少带有一个 pod 的部署，并且服务映射到 Red Hat OpenShift 上的部署或 pod 时，Red Hat OpenShift 会为 **Pod IP** 地址和 **服务端口** 的所有组合创建一个 **Endpoints** 对象。Discovery **GET** 端点会接收请求，并以 JSON 格式协调 Endpoints 和部署模式信息。

以下示例演示了此端点如何以 JSON 格式以分级树视图打开结果。在示例中，树的根是 **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，以及由可插拔发现 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. 发现 POST 端点

使用 Cryostat 注册的插件会向 Discovery **POST** 端点发送 **POST /api/v2.2/discovery/:id** 请求。请求的 **id** 参数指的是注册的插件的 ID。此端点处理与插件关联的任何子树，然后映射部署模式。



注意

Discovery **POST** 端点仅适用于 `/api/v2.2/discovery`。

Cryostat **POST** 请求定义一个子树 **REALM** 节点，以便插件端点可以在请求过程中在 **REALM** 节点上发布其子节点类型。该插件负责为Cryostat **REALM** 节点提供正确的信息，并将节点类型放在正确的子树位置中。插件必须提供令牌，其中包含它发送到 Discovery **POST** 端点的 **POST** 请求，以便插件可以绕过授权标头检查之前通过的插件。



注意

在插件向 Cryostat 发送更新后，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。插件必须在每个请求上向 Cryostat 发送可发现目标的完整列表或树。

插件的 **POST** 请求示例，该请求详细介绍了重要目标应用程序信息

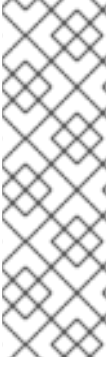
```
[
  {
    "labels": {},
    "nodeType": "JVM",
    "name": "service:jmx:rmi:///jndi/rmi://myapp.svc.local:9091/jmxrmi",
    "nodeType": "JVM",
    "target": {
      "alias": "com.MyApp",
      "annotations": {
        "cryostat": {},
        "platform": {}
      }
    },
    "connectUrl": "service:jmx:rmi:///jndi/rmi://myapp.svc.local:9091/jmxrmi",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

对插件 **POST** 请求的 Cryostat 响应示例

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

1.6. 发现取消注册端点

如果插件注册了 Cryostat 作为 **插件**，然后插件需要关闭，则插件通常会发出一个请求，以便从 Cryostat 中取消注册自身作为插件。**Cryostat** 发送 **DELETE** `/api/v2.2/discovery/:id?token=:token` 请求发送到 **Discovery deregistration** 端点，然后取消从 **Cryostat** 中取消注册插件。



注意

Discovery deregistration 端点仅适用于 `/api/v2.2/discovery`。

Cryostat 在注册过程后向插件发送常规 **POST** 请求，以确保插件仍在运行。如果插件不响应其中一个请求，则 **Cryostat** 将启动未注册插件的过程。

端点从发现模式和插件的注册信息中删除插件的 **REALM** 子树。

DELETE `/api/v2.2/discovery/:id?token=:token` 请求端点进程示例

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

1.7. 错误代码

当其任何端点完成任务或使用 **Cryostat** 时遇到问题时，可插拔 **Discovery API** 会返回消息。

当端点尝试根据 **GET** 和 **POST** 请求注册带有 **Cryostat** 的插件时，端点可以返回以下消息类型：

- **200**：端点成功完成任务。例如，**Discovery deregistration** 端点会返回 **JSON** 格式的消息，并在消息的 **id** 元素中定义的 **unregistered** 插件。
- **400**: **JSON** 文档结构无效，或者 **id** 元素以无效格式编写。
- **401**: 插件没有通过 **Authorization** 标头步骤进行注册。如果令牌已过期或者您取消注册插

件，端点也会返回此错误消息。

- **404: 插件 id 元素没有找到。插件可能失败的回调 检查。考虑重新注册插件。**

第 2 章 GRAPHQL API

GraphQL API 端点 `/api/v3/graphql` 会自动针对目标 JVM 运行更短且简单的查询。这些查询可以针对目标 JVM 的活动和存档记录运行。另外，API 可以根据常规 Cryostat 归档运行查询。您可以自定义查询来自动以下任务进行活跃或归档的记录：

您可以自定义查询来自动以下任务进行活跃或归档的记录：

- **archive**
- **删除**
- **Start**
- **Stop**

在创建自定义查询时，您必须在查询中指定 GraphQL API 端点的特定信息。然后，POST 请求可以处理信息并将信息发送到 Cryostat。以下示例指定了 GraphQL 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 更强大，其工作负载功能有限。例如，HTTP REST API 需要您为每个副本创建一个 API 请求，您想要在 OpenShift 上的容器内启动每个扩展的副本。GraphQL API 可以在一个 API 请求中实现此任务，这会提高 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，在对响应数据执行迭代操作时，您需要编写自定义客户端来解析 API JSON 响应。GraphQL API 不需要您完成此操作。

其他资源

请参阅 [GraphQL 简介 \(GraphQL\)](#)

2.1. 使用 GRAPHQL API 创建自定义查询

您可以使用 HTTPie 等 HTTP 客户端与 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，以便功能可以使用这些值来准确定位您需要的数据。

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 客户端请求可将查询发送到 Cryostat，以便 Cryostat 可以执行以下任务：

1. 对所有目标 JVM 应用进行快照记录。
2. 将每个应用程序中的记录信息复制到 Cryostat 归档中。
3. 创建自动规则，为任何检测到的目标 JVM 应用自动启动持续监控记录。



注意

存在很多查询可能性，因此请确保准确确定查询的数据类型和字段的值。否则，您可能会得到与您的要求不匹配的查询结果。

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

确定与 Cryostat 实例交互的所有已知目标 JVM 的简单查询示例

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

```

{
  targetNodes {
    name
    nodeType
    labels {
      key
      value
    }
    alias
    connectUrl
  }
}

```

前面的示例为 **targetNodes** 元素设置特定的值，如 **name**。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/v3/graphql query=@graphql/target-nodes-query.graphql
POST /api/v3/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 {
      target {
        doStartRecording(recording: {
          name: "myrecording",
          template: "Profiling",
          templateType: "TARGET",
          duration: 30
        }) {
          name
          state
        }
      }
    }
  }
}

```

前面的示例为 `environmentNodes` 元素设置以下特定值，示例不会返回 JVM 目标应用程序：

- `<application_pod_name >` : 确保查询以特定 pod 为目标，该 pod 在与 Cryostat 相同的命名空间中运行。
- `descendantTargets` : 提供 JVM 目标对象的数组。
- `doStartRecording` : GraphQL API 为查询列表的每个目标 JVM 启动 JFR 记录。

运行查询后，查询会返回您启动的 JFR 记录的信息，如活动应用程序节点列表。

对于 Red Hat OpenShift，这个查询会返回 `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，该请求返回任何与 Cryostat 和 Red Hat OpenShift 交互的 JVM 对象。然后，API 请求会为那些返回的对象启动 JFR 记录。

第 3 章 JMC 代理插件

您可以使用 JMC 代理插件将 JMC 代理实施添加到 Cryostat。然后，您可以使用 JMC Agent 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

当您需要诊断正在运行的 JVM 应用程序的问题时，Cryostat 与 JMC 代理相结合，您可以为您提供更多信息。

JMC Agent JAR 文件必须与目标 JVM 应用位于同一个 Red Hat OpenShift 容器中。否则，Cryostat 无法在应用程序上使用 JMC 代理功能。

在 Cryostat web 控制台中，您可以上传探测模板，然后将这些模板插入到 JVM 应用程序中。如果需要，您可以在后续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 可以在 JVM 应用上完成一系列 JMC 代理操作。

当您使用 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 代理运行。



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 软件包使用源 Red Hat Package Manager (SRPM) 作为产品化 RHEL 软件包构建，因此 CRB 软件包定期接收更新。请参阅 [下载并安装 JDK Mission Control \(JMC\) \(使用 Red Hat build of Cryostat\) 的 JDK Flight Recorder](#)

先决条件

- 下载并安装 `jmc` 软件包。

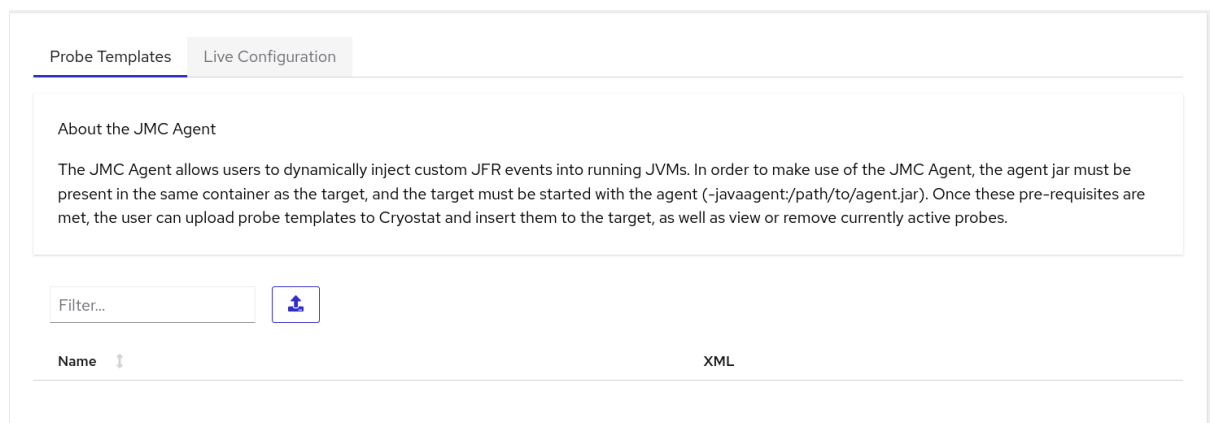
- 下载 Adoptium Agent JAR 文件。请参阅 [adoptium/jmc-build \(GitHub\)](#)。
- 使用 `--add-opens=java.base/jdk.internal.misc=ALL-UNNAMED` 标志启动 Java 应用程序。例如，`./<your_application> --add-opens=java.base/jdk.internal.misc=ALL-UNNAMED`。
- 为您的 Java 应用启动 JMC 代理。请参阅 [启动 JDK Mission Control \(JMC\)代理](#)（使用红帽构建的 JDK Flight Recorder）。

流程

1.

在 Cryostat web 控制台中进入 Events 菜单。如果 JMC Agent 已成功添加到 Cryostat 实例，则会在 Event Templates 窗格下打开 Probe Templates 窗格。

图 3.1. Cryostat web 控制台中的 Probe Templates 标签页



注意

Web 控制台中可能会打开 **Authentication Required** 对话框。如有提示，在 **Authentication Required** 对话框中输入您的用户名和密码，然后点 **Save** 向目标 JVM 提供您的 JMX 凭证。

2.

使用您的首选文本编辑器创建 XML 配置文件。使用 JFR 事件信息填充文件，如应用程序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。当此文件上传到 Cryostat 时，Cryostat 可以将名为 Cryostat 代理插件 Demo Event 的自定义 JFR 事件添加到应用程序中。当 JMC 代理的 `retrieveEventProbes` 方法被调用时，Cryostat 会启动 JFR 事件。

```
<jfragment>
```

```

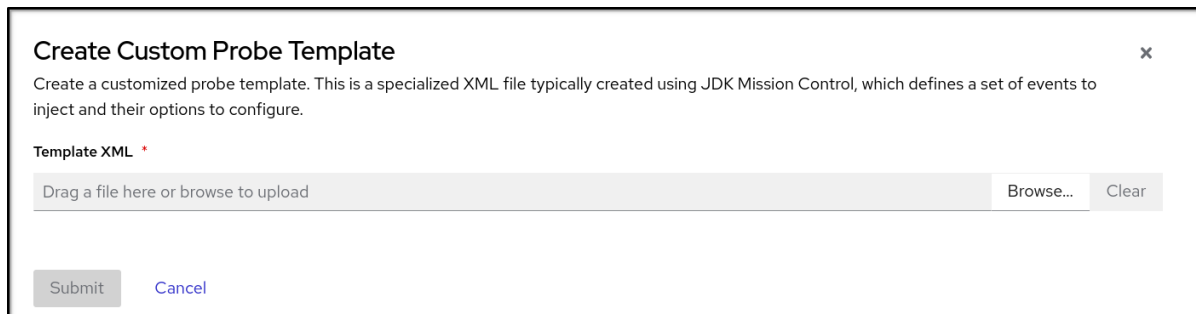
<!-- Global configuration options -->
<config>
  <classprefix>__JFREvent</classprefix>
  <allowtostring>true</allowtostring>
  <allowconverter>true</allowconverter>
</config>
<events>
  <event id="cryostat.demo.jfr.event9">
    <label>Cryostat Agent Plugin Demo Event</label>
    <description>Event for 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.

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

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



提示

如果要从此 **Template XML** 字段中删除上传的文件，请单击 **Clear** 按钮。

4.

单击 **Browse** 按钮找到 XML 文件。

5.

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

6.

点探测模板旁边的 **overflow** 菜单。

7. 点 **Insert Probes**。探测显示在 **Probe Templates** 选项卡下的表中，以及 **Live Configuration** 选项卡下的表。
8. *可选*：转至 **Live Configuration** 选项卡，您可以在其中查看每个活跃探测的信息，如名称、类 等等。
9. *可选*：在 **Live Configuration** 选项卡中，您可以点击 **Remove All Probes** 来删除表中列出的探测。

验证

1. 在 **Events** 菜单中点 **Event Types** 选项卡。
2. 检查表中是否列出了来自您的 XML 配置的命名 JFR 事件。对于此过程中使用的示例，表中显示 **Cryostat 代理插件 Demo Event**。

其他资源

- 请参阅在 [Red Hat build of Cryostat 中使用 JDK Flight Recorder](#)

更新于 **2024-07-02**