# OpenShift Container Platform 4.16

## 分布式追踪

在 OpenShift Container Platform 中配置和使用 Network Observability Operator

Last Updated: 2024-07-02

# OpenShift Container Platform 4.16 分布式追踪

在 OpenShift Container Platform 中配置和使用 Network Observability Operator

## 法律通告



## 摘要

使用 Network Observability Operator 观察和分析 OpenShift Container Platform 集群的网络流量流。

# 目录

# 第 1 章 发行注记

## 1.1. RED HAT OPENSHIFT DISTRIBUTED TRACING PLATFORM 3.2.1 发行注记

### 1.1.1. 分布式追踪概述

作为服务所有者，您可以使用分布式追踪来检测您的服务，以收集与服务架构相关的信息。您可以使用 Red Hat OpenShift distributed tracing 平台来监控、网络性能分析，并对现代、云原生的微服务应用程序中组件间的交互进行故障排除。

使用分布式追踪平台，您可以执行以下功能：

- 监控分布式事务
- 优化性能和延迟时间
- 执行根原因分析

您可以将分布式追踪平台 与 红帽构建的 OpenTelemetry 结合使用。

此 Red Hat OpenShift distributed tracing 平台发行版本包括 Red Hat OpenShift distributed tracing Platform (Tempo) 和已弃用的 Red Hat OpenShift distributed tracing 平台 (Jaeger)。

### 1.1.2. CVE

此发行版本解决了 CVE-2024-25062。

### 1.1.3. Red Hat OpenShift distributed tracing Platform (Tempo)

Red Hat OpenShift distributed tracing Platform (Tempo) 通过 Tempo Operator 提供。

#### 1.1.3.1. 已知问题

当前存在一个已知问题：

- 目前，分布式追踪平台(Tempo)在 IBM Z (**s390x**)架构中会失败。(TRACING-3545)

### 1.1.4. Red Hat OpenShift distributed tracing Platform (Jaeger)

Red Hat OpenShift distributed tracing Platform (Jaeger) 通过 Red Hat OpenShift distributed tracing Platform Operator 提供。

**重要**

Jaeger 不使用经 FIPS 验证的加密模块。

#### 1.1.4.1. 已知问题

当前存在一个已知问题：

- 目前，不支持 Apache Spark。

- 目前，IBM Z 和 IBM Power 架构不支持通过 AMQ/Kafka 进行流部署。

### 1.1.5. 获取支持

如果您在执行本文档所述的某个流程或 OpenShift Container Platform 时遇到问题，请访问 红帽客户门户网站。

通过红帽客户门户网站：

- 搜索或者浏览红帽知识库，了解与红帽产品相关的文章和解决方案。
- 提交问题单给红帽支持。
- 访问其他产品文档。

要识别集群中的问题，您可以在 OpenShift Cluster Manager 中使用 Insights。Insights 提供了问题的详细信息，并在有可用的情况下，提供了如何解决问题的信息。

如果您对本文档有任何改进建议，或发现了任何错误，请为相关文档组件提交 JIRA 问题。请提供具体详情，如章节名称和 OpenShift Container Platform 版本。

### 1.1.6. 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。

## 1.2. 以前版本的 RED HAT OPENSHIFT DISTRIBUTED TRACING 平台发行注记

### 1.2.1. 分布式追踪概述

作为服务所有者，您可以使用分布式追踪来检测您的服务，以收集与服务架构相关的信息。您可以使用 Red Hat OpenShift distributed tracing 平台来监控、网络性能分析，并对现代、云原生的微服务应用程序中组件间的交互进行故障排除。

使用分布式追踪平台，您可以执行以下功能：

- 监控分布式事务
- 优化性能和延迟时间
- 执行根原因分析

您可以将分布式追踪平台 与 红帽构建的 OpenTelemetry 结合使用。

### 1.2.2. Red Hat OpenShift distributed tracing Platform 3.2 发行注记

此 Red Hat OpenShift distributed tracing 平台发行版本包括 Red Hat OpenShift distributed tracing Platform (Tempo) 和已弃用的 Red Hat OpenShift distributed tracing 平台 (Jaeger)。

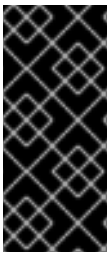
#### 1.2.2.1. Red Hat OpenShift distributed tracing Platform (Tempo)

Red Hat OpenShift distributed tracing Platform (Tempo) 通过 Tempo Operator 提供。

#### 1.2.2.1.1. 技术预览功能

这个版本包括以下技术预览功能：

- 支持 Tempo monolithic 部署。



**重要**

Tempo monolithic 部署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。

#### 1.2.2.1.2. 新功能及功能增强

这个版本引进了以下改进：

- Red Hat OpenShift distributed tracing Platform (Tempo) 3.2 基于开源 Grafana Tempo 2.4.1。
- 允许覆盖每个组件的资源。

#### 1.2.2.1.3. 程序错误修复

在这个版本中引进了以下程序错误修复：

- 在此次更新之前，Jaeger UI 只显示在前 15 分钟内发送 trace 的服务。在这个版本中，可以使用以下字段来配置服务和操作名称的可用性：**spec.template.queryFrontend.jaegerQuery.servicesQueryDuration**。(TRACING-3139)
- 在此次更新之前，在搜索大型 trace 后，**query-frontend** pod 可能会在内存不足 (OOM) 时停止。在这个版本中，可以设置资源限值以防止出现这个问题。(TRACING-4009)

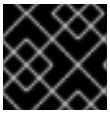
#### 1.2.2.1.4. 已知问题

当前存在一个已知问题：

- 目前，分布式追踪平台(Tempo)在 IBM Z (**s390x**)架构中会失败。(TRACING-3545)

### 1.2.2.2. Red Hat OpenShift distributed tracing Platform (Jaeger)

Red Hat OpenShift distributed tracing Platform (Jaeger) 通过 Red Hat OpenShift distributed tracing Platform Operator 提供。



**重要**

Jaeger 不使用经 FIPS 验证的加密模块。

#### 1.2.2.2.1. 支持 OpenShift Elasticsearch Operator

Red Hat OpenShift distributed tracing Platform (Jaeger) 3.2 支持与 OpenShift Elasticsearch Operator 5.6、5.7 和 5.8 一起使用。

#### 1.2.2.2.2. 过时的功能

在 Red Hat OpenShift distributed tracing Platform 3.2、Jaeger 和对 Elasticsearch 的支持仍被弃用，并计划在以后的发行版本中删除。红帽将在当前发行生命周期中为这些组件提供支持，并为严重的有高重要级别的 CVE 和程序错误提供修复，但不会再为这些组件提供功能增强。Tempo Operator 和红帽构建的 OpenTelemetry 是分布式追踪集合和存储的首选 Operator。用户需要采用 OpenTelemetry 和 Tempo 分布式追踪堆栈，因为它是要进一步增强的堆栈。

在 Red Hat OpenShift distributed tracing 平台 3.2 中，Jaeger 代理已弃用，计划在以下发行版本中删除。红帽将在当前发行生命周期中对 Jaeger 代理提供程序错误修正和支持，但 Jaeger 代理将不再获得改进，并将在以后被删除。Red Hat build of OpenTelemetry 提供的 OpenTelemetry Collector 是注入 trace 收集器代理的首选 Operator。

#### 1.2.2.2.3. 新功能及功能增强

在这个版本中，对分布式追踪平台(Jaeger)引进了以下改进：

- Red Hat OpenShift distributed tracing Platform (Jaeger) 3.2 基于开源 Jaeger 版本 1.57.0。

#### 1.2.2.2.4. 已知问题

当前存在一个已知问题：

- 目前，不支持 Apache Spark。
- 目前，IBM Z 和 IBM Power 架构不支持通过 AMQ/Kafka 进行流部署。

### 1.2.3. Red Hat OpenShift 发布的 tracing Platform 3.1.1 发行注记

此 Red Hat OpenShift distributed tracing 平台发行版本包括 Red Hat OpenShift distributed tracing Platform (Tempo) 和已弃用的 Red Hat OpenShift distributed tracing 平台 (Jaeger)。

#### 1.2.3.1. CVE

此发行版本解决了 CVE-2023-39326 的问题。

#### 1.2.3.2. Red Hat OpenShift distributed tracing Platform (Tempo)

Red Hat OpenShift distributed tracing Platform (Tempo) 通过 Tempo Operator 提供。

#### 1.2.3.2.1. 已知问题

当前已知的问题：

- 目前，当与 Tempo Operator 一起使用时，Jaeger UI 只显示在最后 15 分钟内发送了 trace 的服务。对于没有在最后 15 分钟内发送 trace 的服务，trace 仍然会被存储，但不会在 Jaeger UI 中显示。(TRACING-3139)
- 目前，分布式追踪平台(Tempo)在 IBM Z (**s390x**)架构中会失败。(TRACING-3545)

#### 1.2.3.3. Red Hat OpenShift distributed tracing Platform (Jaeger)

Red Hat OpenShift distributed tracing Platform (Jaeger) 通过 Red Hat OpenShift distributed tracing Platform Operator 提供。

**重要**

Jaeger 不使用经 FIPS 验证的加密模块。

#### 1.2.3.3.1. 支持 OpenShift Elasticsearch Operator

Red Hat OpenShift distributed tracing Platform (Jaeger) 3.1.1 支持与 OpenShift Elasticsearch Operator 5.6、5.7 和 5.8 一起使用。

#### 1.2.3.3.2. 过时的功能

在 Red Hat OpenShift distributed tracing Platform 3.1.1、Jaeger 和对 Elasticsearch 的支持仍被弃用，并计划在以后的发行版本中删除。红帽将在当前发行生命周期中对这些组件提供关键及以上的 CVE 程序错误修复和支持，但这些组件将不再获得功能增强。

在 Red Hat OpenShift distributed tracing 平台 3.1.1 中，由 Tempo Operator 提供的 Tempo，以及由红帽构建的 OpenTelemetry 提供的 OpenTelemetry Collector 是分布式追踪集合和存储的首选 Operator。OpenTelemetry 和 Tempo 分布式追踪堆栈供所有用户采用，因为这将进一步增强。

#### 1.2.3.3.3. 已知问题

当前已知的问题：

- 目前，不支持 Apache Spark。
- 目前，IBM Z 和 IBM Power 架构不支持通过 AMQ/Kafka 进行流部署。

### 1.2.4. Red Hat OpenShift distributed tracing Platform 3.1 发行注记

此 Red Hat OpenShift distributed tracing 平台发行版本包括 Red Hat OpenShift distributed tracing Platform (Tempo) 和已弃用的 Red Hat OpenShift distributed tracing 平台 (Jaeger)。

#### 1.2.4.1. Red Hat OpenShift distributed tracing Platform (Tempo)

Red Hat OpenShift distributed tracing Platform (Tempo) 通过 Tempo Operator 提供。

#### 1.2.4.1.1. 新功能及功能增强

在这个版本中，对分布式追踪平台(Tempo)引入了以下改进：

- Red Hat OpenShift distributed tracing Platform (Tempo) 3.1 基于开源 Grafana Tempo 2.3.1。
- 支持集群范围的代理环境。
- 支持 TraceQL 到网关组件。

#### 1.2.4.1.2. 程序错误修复

在这个版本中，为分布式追踪平台(Tempo)引入了以下程序错误修复：

- 在此次更新之前，当使用 OpenShift Container Platform 4.15 中启用 **monitorTab** 创建 TempoStack 实例时，不会创建所需的 **tempo-redmetrics-cluster-monitoring-view** ClusterRoleBinding。在这个版本中，当 Operator 部署到任意命名空间中时，通过为 monitor 选项卡修复 Operator RBAC 解决了这个问题。(TRACING-3786)

- 在此次更新之前，当在带有 IPv6 网络堆栈的 OpenShift Container Platform 集群上创建 TempoStack 实例时， compactor 和 ingestor pod 以 **CrashLoopBackOff** 状态运行，从而导致多个错误。这个版本支持 IPv6 集群。(TRACING-3226)

#### 1.2.4.1.3. 已知问题

当前已知的问题：

- 目前，当与 Tempo Operator 一起使用时，Jaeger UI 只显示在最后 15 分钟内发送了 trace 的服务。对于没有在最后 15 分钟内发送 trace 的服务，trace 仍然会被存储，但不会在 Jaeger UI 中显示。(TRACING-3139)
- 目前，分布式追踪平台(Tempo)在 IBM Z (**s390x**)架构中会失败。(TRACING-3545)

### 1.2.4.2. Red Hat OpenShift distributed tracing Platform (Jaeger)

Red Hat OpenShift distributed tracing Platform (Jaeger) 通过 Red Hat OpenShift distributed tracing Platform Operator 提供。

**重要**

Jaeger 不使用经 FIPS 验证的加密模块。

#### 1.2.4.2.1. 支持 OpenShift Elasticsearch Operator

Red Hat OpenShift distributed tracing Platform (Jaeger) 3.1 支持与 OpenShift Elasticsearch Operator 5.6、5.7 和 5.8 一起使用。

#### 1.2.4.2.2. 过时的功能

在 Red Hat OpenShift distributed tracing Platform 3.1、Jaeger 和对 Elasticsearch 的支持仍被弃用，并计划在以后的发行版本中删除。红帽将在当前发行生命周期中对这些组件提供关键及以上的 CVE 程序错误修复和支持，但这些组件将不再获得功能增强。

在 Red Hat OpenShift distributed tracing 平台 3.1 中，由 Tempo Operator 提供的 Tempo，以及由红帽构建的 OpenTelemetry 提供的 OpenTelemetry Collector 是分布式追踪集合和存储的首选 Operator。OpenTelemetry 和 Tempo 分布式追踪堆栈供所有用户采用，因为这将进一步增强。

#### 1.2.4.2.3. 新功能及功能增强

在这个版本中，对分布式追踪平台(Jaeger)引进了以下改进：

- Red Hat OpenShift distributed tracing Platform (Jaeger) 3.1 基于开源 Jaeger 版本 1.53.0。

#### 1.2.4.2.4. 程序错误修复

在这个版本中，为分布式追踪平台(Jaeger)引入了以下程序错误修复：

- 在此次更新之前，**jager-query** pod 中的 **jaeger-agent** 容器的连接目标 URL 被 OpenShift Container Platform 4.13 中的另一个命名空间 URL 覆盖。这是因为 **jaeger-operator** 中的 sidecar 注入代码中的一个错误，从而导致非确定的 **jaeger-agent** 注入。在这个版本中，Operator 会优先选择与目标部署相同的命名空间中的 Jaeger 实例。(TRACING-3722)

#### 1.2.4.2.5. 已知问题

当前已知的问题：

- 目前，不支持 Apache Spark。
- 目前，IBM Z 和 IBM Power 架构不支持通过 AMQ/Kafka 进行流部署。

## 1.2.5. Red Hat OpenShift distributed tracing Platform 3.0 发行注记

### 1.2.5.1. Red Hat OpenShift distributed tracing Platform 3.0 中的组件版本

| Operator | 组件 | Version |
| --- | --- | --- |
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.51.0 |
| Red Hat OpenShift distributed tracing Platform (Tempo) | Tempo | 2.3.0 |

### 1.2.5.2. Red Hat OpenShift distributed tracing Platform (Jaeger)

#### 1.2.5.2.1. 过时的功能

在 Red Hat OpenShift distributed tracing Platform 3.0、Jaeger 和对 Elasticsearch 的支持仍被弃用，并计划在以后的发行版本中删除。红帽将在当前发行生命周期中对这些组件提供关键及以上的 CVE 程序错误修复和支持，但这些组件将不再获得功能增强。

在 Red Hat OpenShift distributed tracing 平台 3.0 中，由 Tempo Operator 提供的 Tempo，以及由红帽构建的 OpenTelemetry 提供的 OpenTelemetry Collector 是分布式追踪集合和存储的首选 Operator。OpenTelemetry 和 Tempo 分布式追踪堆栈供所有用户采用，因为这将进一步增强。

#### 1.2.5.2.2. 新功能及功能增强

在这个版本中，对分布式追踪平台(Jaeger)引进了以下改进：

- 支持 ARM 架构。
- 支持集群范围的代理环境。

#### 1.2.5.2.3. 程序错误修复

在这个版本中，为分布式追踪平台(Jaeger)引入了以下程序错误修复：

- 在此次更新之前，Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 使用了与 **relatedImages** 以外的其他镜像。这会导致在启动 **jaeger** pod 时在断开连接的网络环境中出现 **ImagePullBackOff** 错误，因为 **oc adm catalog mirror** 命令会镜像 **relatedImages** 中指定的镜像。在这个版本中，在使用 **oc adm catalog mirror** CLI 命令时为断开连接的环境提供支持。(TRACING-3546)

#### 1.2.5.2.4. 已知问题

当前存在一个已知问题：

- 目前，不支持 Apache Spark。
- 目前，IBM Z 和 IBM Power 架构不支持通过 AMQ/Kafka 进行流部署。

#### 1.2.5.3. Red Hat OpenShift distributed tracing Platform (Tempo)

##### 1.2.5.3.1. 新功能及功能增强

在这个版本中，对分布式追踪平台(Tempo)引入了以下改进：

- 支持 ARM 架构。
- 支持 span request count, duration, 和 error count (RED)指标。在 Jaeger 控制台中，可以在作为 Tempo 的一部分或 **Observe** 菜单的 web 控制台中视觉化指标。

##### 1.2.5.3.2. 程序错误修复

在这个版本中，为分布式追踪平台(Tempo)引入了以下程序错误修复：

- 在此次更新之前，**TempoStack** CRD 不接受自定义 CA 证书，尽管选项可以选择 CA 证书。在这个版本中，对连接到对象存储的自定义 TLS CA 选项的支持。(TRACING-3462)
- 在此次更新之前，当将 Red Hat OpenShift distributed tracing Platform operator 镜像镜像到断开连接的集群中使用的镜像 registry 时，**tempo**, **tempo-gateway**, **opa-openshift**, 和 **tempo-query** 的相关 Operator 镜像没有被镜像(mirror)。在这个版本中，在使用 **oc adm catalog mirror** CLI 命令时对断开连接的环境的支持。(TRACING-3523)
- 在此次更新之前，当没有部署网关时，Red Hat OpenShift distributed tracing 平台的查询 frontend 服务会使用内部 mTLS。这会导致端点失败。在这个版本中修复了在没有部署网关时的 mTLS。(TRACING-3510)

##### 1.2.5.3.3. 已知问题

当前已知的问题：

- 目前，当与 Tempo Operator 一起使用时，Jaeger UI 只显示在最后 15 分钟内发送了 trace 的服务。对于没有在最后 15 分钟内发送 trace 的服务，trace 仍然会被存储，但不会在 Jaeger UI 中显示。(TRACING-3139)
- 目前，分布式追踪平台(Tempo)在 IBM Z (**s390x**)架构中会失败。(TRACING-3545)

### 1.2.6. Red Hat OpenShift distributed tracing Platform 2.9.2 发行注记

#### 1.2.6.1. Red Hat OpenShift distributed tracing Platform 2.9.2 中的组件版本

| Operator | 组件 | Version |
|---|---|---|
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.47.0 |
| Red Hat OpenShift distributed tracing Platform (Tempo) | Tempo | 2.1.1 |

#### 1.2.6.2. CVE

此发行版本解决了 CVE-2023-46234 的问题。

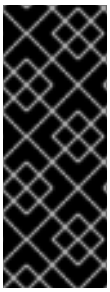
#### 1.2.6.3. Red Hat OpenShift distributed tracing Platform (Jaeger)

##### 1.2.6.3.1. 已知问题

当前已知的问题：

- 不支持 Apache spark。
- IBM Z 和 IBM Power 架构上不支持通过 AMQ/Kafka 进行流部署。

#### 1.2.6.4. Red Hat OpenShift distributed tracing Platform (Tempo)



**重要**

Red Hat OpenShift distributed tracing Platform (Tempo)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。

##### 1.2.6.4.1. 已知问题

当前已知的问题：

- 目前，没有为连接对象存储而实施自定义 TLS CA 选项。(TRACING-3462)
- 目前，当与 Tempo Operator 一起使用时，Jaeger UI 只显示在最后 15 分钟内发送了 trace 的服务。对于没有在最后 15 分钟内发送 trace 的服务，trace 仍然会被存储，但不会在 Jaeger UI 中显示。(TRACING-3139)
- 目前，分布式追踪平台(Tempo)在 IBM Z (**s390x**)架构中会失败。(TRACING-3545)
- 目前，在未部署网关时，Tempo 查询前端服务不得使用内部 mTLS。这个问题不会影响 Jaeger Query API。解决办法是禁用 mTLS。(TRACING-3510)

  临时**解决方案**

  禁用 mTLS，如下所示：

  1. 运行以下命令，打开 Tempo Operator ConfigMap 进行编辑：

     ```
     $ oc edit configmap tempo-operator-manager-config -n openshift-tempo-operator
     ```
     (1)

     (1) 安装 Tempo Operator 的项目。

  2. 通过更新 YAML 文件来禁用 Operator 配置中的 mTLS：

     ```
     data:
       controller_manager_config.yaml: |
     ```

```
featureGates:
  httpEncryption: false
  grpcEncryption: false
  builtInCertManagement:
    enabled: false
```

3. 运行以下命令来重启 Tempo Operator pod：

```
$ oc rollout restart deployment.apps/tempo-operator-controller -n openshift-tempo-operator
```

- 缺少在受限环境中运行 Tempo Operator 的镜像。Red Hat OpenShift distributed tracing Platform (Tempo) CSV 缺少对操作对象镜像的引用。(TRACING-3523)

  临时**解决方案**

  在镜像工具中添加 Tempo Operator 相关镜像，将镜像复制到 registry：

```
kind: ImageSetConfiguration
apiVersion: mirror.openshift.io/v1alpha2
archiveSize: 20
storageConfig:
  local:
    path: /home/user/images
mirror:
  operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.13
    packages:
    - name: tempo-product
      channels:
      - name: stable
  additionalImages:
  - name: registry.redhat.io/rhosdt/tempo-rhel8@sha256:e4295f837066efb05bcc5897f31eb2bdbd81684a8c59d6f9498dd3590c62c12a
  - name: registry.redhat.io/rhosdt/tempo-gateway-rhel8@sha256:b62f5cedfeb5907b638f14ca6aaeea50f41642980a8a6f87b7061e88d90fac23
  - name: registry.redhat.io/rhosdt/tempo-gateway-opa-rhel8@sha256:8cd134deca47d6817b26566e272e6c3f75367653d589f5c90855c59b2fab01e9

  - name: registry.redhat.io/rhosdt/tempo-query-rhel8@sha256:0da43034f440b8258a48a0697ba643b5643d48b615cdb882ac7f4f1f80aad08e
```

## 1.2.7. Red Hat OpenShift distributed tracing Platform 2.9.1 发行注记

### 1.2.7.1. Red Hat OpenShift distributed tracing Platform 2.9.1 中的组件版本

| Operator | 组件 | Version |
|---|---|---|
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.47.0 |

| | | |
|---|---|---|
| Red Hat OpenShift distributed tracing Platform (Tempo) | Tempo | 2.1.1 |

#### 1.2.7.2. CVE

此发行版本修复了 CVE-2023-44487。

### 1.2.7.3. Red Hat OpenShift distributed tracing Platform (Jaeger)

#### 1.2.7.3.1. 已知问题

当前已知的问题：

- 不支持 Apache spark。
- IBM Z 和 IBM Power 架构上不支持通过 AMQ/Kafka 进行流部署。

### 1.2.7.4. Red Hat OpenShift distributed tracing Platform (Tempo)

**重要**

Red Hat OpenShift distributed tracing Platform (Tempo)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。

#### 1.2.7.4.1. 已知问题

当前已知的问题：

- 目前，没有为连接对象存储而实施自定义 TLS CA 选项。(TRACING-3462)
- 目前，当与 Tempo Operator 一起使用时，Jaeger UI 只显示在最后 15 分钟内发送了 trace 的服务。对于没有在最后 15 分钟内发送 trace 的服务，trace 仍然会被存储，但不会在 Jaeger UI 中显示。(TRACING-3139)
- 目前，分布式追踪平台(Tempo)在 IBM Z (**s390x**)架构中会失败。(TRACING-3545)
- 目前，在未部署网关时，Tempo 查询前端服务不得使用内部 mTLS。这个问题不会影响 Jaeger Query API。解决办法是禁用 mTLS。(TRACING-3510)

  临时**解决方案**

  禁用 mTLS，如下所示：

  1. 运行以下命令，打开 Tempo Operator ConfigMap 进行编辑：

     ```
     $ oc edit configmap tempo-operator-manager-config -n openshift-tempo-operator
     ```
     (1)

     (1) 安装 Tempo Operator 的项目。

2. 通过更新 YAML 文件来禁用 Operator 配置中的 mTLS：

```
data:
  controller_manager_config.yaml: |
    featureGates:
      httpEncryption: false
      grpcEncryption: false
      builtInCertManagement:
        enabled: false
```

3. 运行以下命令来重启 Tempo Operator pod：

```
$ oc rollout restart deployment.apps/tempo-operator-controller -n openshift-tempo-operator
```

- 缺少在受限环境中运行 Tempo Operator 的镜像。Red Hat OpenShift distributed tracing Platform (Tempo) CSV 缺少对操作对象镜像的引用。(TRACING-3523)

  临时**解决方案**

  在镜像工具中添加 Tempo Operator 相关镜像，将镜像复制到 registry：

```
kind: ImageSetConfiguration
apiVersion: mirror.openshift.io/v1alpha2
archiveSize: 20
storageConfig:
  local:
    path: /home/user/images
mirror:
  operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.13
    packages:
    - name: tempo-product
      channels:
      - name: stable
  additionalImages:
  - name: registry.redhat.io/rhosdt/tempo-rhel8@sha256:e4295f837066efb05bcc5897f31eb2bdbd81684a8c59d6f9498dd3590c62c12a
  - name: registry.redhat.io/rhosdt/tempo-gateway-rhel8@sha256:b62f5cedfeb5907b638f14ca6aaeea50f41642980a8a6f87b7061e88d90fac23
  - name: registry.redhat.io/rhosdt/tempo-gateway-opa-rhel8@sha256:8cd134deca47d6817b26566e272e6c3f75367653d589f5c90855c59b2fab01e9

  - name: registry.redhat.io/rhosdt/tempo-query-rhel8@sha256:0da43034f440b8258a48a0697ba643b5643d48b615cdb882ac7f4f1f80aad08e
```

## 1.2.8. Red Hat OpenShift distributed tracing Platform 2.9 发行注记

### 1.2.8.1. Red Hat OpenShift distributed tracing Platform 2.9 中的组件版本

| Operator | 组件 | Version |
|---|---|---|

| | | |
|---|---|---|
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.47.0 |
| Red Hat OpenShift distributed tracing Platform (Tempo) | Tempo | 2.1.1 |

### 1.2.8.2. Red Hat OpenShift distributed tracing Platform (Jaeger)

#### 1.2.8.2.1. 程序错误修复

- 在此次更新之前，因为 **jaeger-query** 部署中缺少一个 gRPC 端口，连接会被拒绝。此问题会导致 **transport: Error while dialing: dial tcp :16685: connect: connection refused** 错误信息。在这个版本中，Jaeger Query gRPC 端口 (16685) 可以在 Jaeger Query 服务上成功公开。(TRACING-3322)
- 在此次更新之前，为 **jaeger-production-query** 公开的端口是错误的，并导致连接被拒绝。在这个版本中，这个问题已通过在 Jaeger Query 部署上公开 Jaeger Query gRPC 端口(16685) 被解决。(TRACING-2968)
- 在此次更新之前，当在断开连接的环境中的单节点 OpenShift 集群上部署 Service Mesh 时，Jaeger pod 会经常进入 **Pending** 状态。在这个版本中，这个问题已被解决。(TRACING-3312)
- 在此次更新之前，因为 **reason: OOMKilled** 错误信息，Jaeger Operator pod 会以默认内存值重启。在这个版本中，通过删除资源限值解决了这个问题。(TRACING-3173)

#### 1.2.8.2.2. 已知问题

当前已知的问题：

- 不支持 Apache spark。
- IBM Z 和 IBM Power 架构上不支持通过 AMQ/Kafka 进行流部署。

### 1.2.8.3. Red Hat OpenShift distributed tracing Platform (Tempo)

**重要**

Red Hat OpenShift distributed tracing Platform (Tempo)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。

#### 1.2.8.3.1. 新功能及功能增强

此发行版本包括对分布式追踪平台 (Tempo) 的以下改进：

- 支持 Operator 成熟度 级别 IV、Deep Insights，它启用了 TempoStack 实例的升级、监控和警报，以及 Tempo Operator。
- 为网关添加 Ingress 和 Route 配置。

- 支持 **TempoStack** 自定义资源中的 **managed** 和 **unmanaged** 状态。
- 在 Distributor 服务中公开以下额外的 ingestion 协议：Jaeger Thrift 二进制、Jaeger Thrift compact、Jaeger gRPC 和 Zipkin。启用网关时，只启用 OpenTelemetry 协议 (OTLP) gRPC。
- 在 Query Frontend 服务上公开 Jaeger Query gRPC 端点。
- 支持没有网关身份验证和授权的多租户。

#### 1.2.8.3.2. 程序错误修复

- 在此次更新之前，Tempo Operator 与断开连接的环境不兼容。在这个版本中，Tempo Operator 支持断开连接的环境。(TRACING-3145)
- 在此次更新之前，带有 TLS 的 Tempo Operator 无法在 OpenShift Container Platform 上启动。在这个版本中，mTLS 通信会在 Tempo 组件、Operand 启动成功时启用，并且可以访问 Jaeger UI。(TRACING-3091)
- 在此次更新之前，来自 Tempo Operator 的资源限值会导致错误消息，如 **reason: OOMKilled**。在这个版本中，Tempo Operator 的资源限值被删除，以避免此类错误。(TRACING-3204)

#### 1.2.8.3.3. 已知问题

当前已知的问题：

- 目前，没有为连接对象存储而实施自定义 TLS CA 选项。(TRACING-3462)
- 目前，当与 Tempo Operator 一起使用时，Jaeger UI 只显示在最后 15 分钟内发送了 trace 的服务。对于没有在最后 15 分钟内发送 trace 的服务，trace 仍然会被存储，但不会在 Jaeger UI 中显示。(TRACING-3139)
- 目前，分布式追踪平台(Tempo)在 IBM Z (**s390x**)架构中会失败。(TRACING-3545)
- 目前，在未部署网关时，Tempo 查询前端服务不得使用内部 mTLS。这个问题不会影响 Jaeger Query API。解决办法是禁用 mTLS。(TRACING-3510)

  临时**解决方案**

  禁用 mTLS，如下所示：

  1. 运行以下命令，打开 Tempo Operator ConfigMap 进行编辑：

     ```
     $ oc edit configmap tempo-operator-manager-config -n openshift-tempo-operator
     ```
     (1)

     (1) 安装 Tempo Operator 的项目。

  2. 通过更新 YAML 文件来禁用 Operator 配置中的 mTLS：

     ```
     data:
       controller_manager_config.yaml: |
         featureGates:
           httpEncryption: false
           grpcEncryption: false
           builtInCertManagement:
             enabled: false
     ```

3. 运行以下命令来重启 Tempo Operator pod：

```
$ oc rollout restart deployment.apps/tempo-operator-controller -n openshift-tempo-operator
```

- 缺少在受限环境中运行 Tempo Operator 的镜像。Red Hat OpenShift distributed tracing Platform (Tempo) CSV 缺少对操作对象镜像的引用。(TRACING-3523)

  临时**解决方案**

  在镜像工具中添加 Tempo Operator 相关镜像，将镜像复制到 registry：

```
kind: ImageSetConfiguration
apiVersion: mirror.openshift.io/v1alpha2
archiveSize: 20
storageConfig:
  local:
    path: /home/user/images
mirror:
  operators:
  - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.13
    packages:
    - name: tempo-product
      channels:
      - name: stable
  additionalImages:
  - name: registry.redhat.io/rhosdt/tempo-rhel8@sha256:e4295f837066efb05bcc5897f31eb2bdbd81684a8c59d6f9498dd3590c62c12a
  - name: registry.redhat.io/rhosdt/tempo-gateway-rhel8@sha256:b62f5cedfeb5907b638f14ca6aaeea50f41642980a8a6f87b7061e88d90fac23
  - name: registry.redhat.io/rhosdt/tempo-gateway-opa-rhel8@sha256:8cd134deca47d6817b26566e272e6c3f75367653d589f5c90855c59b2fab01e9

  - name: registry.redhat.io/rhosdt/tempo-query-rhel8@sha256:0da43034f440b8258a48a0697ba643b5643d48b615cdb882ac7f4f1f80aad08e
```

## 1.2.9. Red Hat OpenShift distributed tracing Platform 2.8 发行注记

### 1.2.9.1. Red Hat OpenShift distributed tracing Platform 2.8 中的组件版本

| Operator | 组件 | Version |
|---|---|---|
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.42 |
| Red Hat OpenShift distributed tracing Platform (Tempo) | Tempo | 0.1.0 |

### 1.2.9.2. 技术预览功能

此发行版本引进了对 Red Hat OpenShift distributed tracing 平台(Tempo)的支持，作为 Red Hat OpenShift distributed tracing 平台的技术预览功能。

**重要**

Red Hat OpenShift distributed tracing Platform (Tempo)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。

该功能使用 Red Hat OpenShift distributed tracing Platform (Tempo)版本 0.1.0 和上游分布式追踪平台(Tempo)组件的 2.0.1 版本。

您可以使用分布式追踪平台(Tempo)替换 Jaeger，以便您可以使用 S3 兼容存储而不是 ElasticSearch。大多数使用分布式追踪平台(Tempo)而不是 Jaeger 的用户都不会注意到功能的任何区别，因为分布式追踪平台(Tempo)支持与 Jaeger 相同的 ingestion 和查询协议，并使用相同的用户界面。

如果您启用了此技术预览功能，请注意当前实现的以下限制：

- 分布式追踪平台(Tempo)目前不支持断开连接的安装。(TRACING-3145)
- 当您将 Jaeger 用户界面(UI)与分布式追踪平台(Tempo)搭配使用时，Jaeger UI 只会列出最后 15 分钟内发送 trace 的服务。对于没有在最后 15 分钟内发送 trace 的服务，这些 trace 仍然被存储，即使它们在 Jaeger UI 中不可见。(TRACING-3139)

计划在以后的 Red Hat OpenShift distributed tracing Platform 版本中对 Tempo Operator 提供支持。可能的额外功能可能包括对 TLS 身份验证、多租户和多个集群的支持。如需有关 Tempo Operator 的更多信息，请参阅 Tempo 社区文档。

#### 1.2.9.3. 程序错误修复

此发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

### 1.2.10. Red Hat OpenShift distributed tracing Platform 2.7 发行注记

#### 1.2.10.1. Red Hat OpenShift distributed tracing Platform 2.7 中的组件版本

| Operator | 组件 | Version |
| --- | --- | --- |
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.39 |

#### 1.2.10.2. 程序错误修复

此发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

### 1.2.11. Red Hat OpenShift distributed tracing Platform 2.6 发行注记

#### 1.2.11.1. Red Hat OpenShift distributed tracing Platform 2.6 中的组件版本

| Operator | 组件 | Version |
| --- | --- | --- |

| | | |
|---|---|---|
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.38 |

#### 1.2.11.2. 程序错误修复

此发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

### 1.2.12. Red Hat OpenShift distributed tracing Platform 2.5 发行注记

#### 1.2.12.1. Red Hat OpenShift distributed tracing Platform 2.5 中的组件版本

| Operator | 组件 | Version |
|---|---|---|
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.36 |

#### 1.2.12.2. 新功能及功能增强

此发行版本为 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 引进了对 OpenTelemetry 协议(OTLP)的支持。Operator 现在自动启用 OTLP 端口：

- OTLP gRPC 协议的端口 4317。
- OTLP HTTP 协议的端口 4318.

此发行版本还添加了对在红帽构建的 OpenTelemetry Operator 中收集 Kubernetes 资源属性的支持。

#### 1.2.12.3. 程序错误修复

此发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

### 1.2.13. Red Hat OpenShift distributed tracing Platform 2.4 发行注记

#### 1.2.13.1. Red Hat OpenShift distributed tracing Platform 2.4 中的组件版本

| Operator | 组件 | Version |
|---|---|---|
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.34.1 |

#### 1.2.13.2. 新功能及功能增强

此发行版本添加了对使用 OpenShift Elasticsearch Operator 自动置备证书的支持。

使用 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 在安装过程中调用 OpenShift Elasticsearch Operator 进行自我置备。

+

**重要**

当升级到 Red Hat OpenShift distributed tracing platform 2.4 时，Operator 会重新创建 Elasticsearch 实例，这可能需要 5 到 10 分钟。在此期间，分布式追踪将停机且不可用。

#### 1.2.13.3. 技术预览功能

首先创建 Elasticsearch 实例和证书，然后将分布式追踪平台(Jaeger)配置为使用证书是本发行版本的 技术预览。

#### 1.2.13.4. 程序错误修复

此发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

### 1.2.14. Red Hat OpenShift distributed tracing Platform 2.3 发行注记

#### 1.2.14.1. Red Hat OpenShift distributed tracing Platform 2.3.1 中的组件版本

| Operator | 组件 | Version |
| --- | --- | --- |
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.30.2 |

#### 1.2.14.2. Red Hat OpenShift distributed tracing Platform 2.3.0 中的组件版本

| Operator | 组件 | Version |
| --- | --- | --- |
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.30.1 |

#### 1.2.14.3. 新功能及功能增强

在这个版本中，Red Hat OpenShift distributed tracing platform (Jaeger) Operator 被默认安装到 **openshift-distributed-tracing** 命名空间。在此次更新之前，默认安装位于 **openshift-operators** 命名空间中。

#### 1.2.14.4. 程序错误修复

此发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

### 1.2.15. Red Hat OpenShift distributed tracing Platform 2.2 发行注记

#### 1.2.15.1. 技术预览功能

2.1 发行版本中包含的 OpenTelemetry Collector 组件已被删除。

#### 1.2.15.2. 程序错误修复

此 Red Hat OpenShift distributed tracing platform 版本解决了 CVE 报告的安全漏洞问题以及程序错误。

## 1.2.16. Red Hat OpenShift distributed tracing Platform 2.1 发行注记

### 1.2.16.1. Red Hat OpenShift distributed tracing Platform 2.1 中的组件版本

| Operator | 组件 | Version |
| --- | --- | --- |
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.29.1 |

### 1.2.16.2. 技术预览功能

- 此发行版本引入了一个具有破坏性的更改，这个变化与如何在 OpenTelemetry 自定义资源文件中配置证书相关。在这个版本中，**ca_file** 在自定义资源中移到 **tls** 下，如下例所示。

  **OpenTelemetry 版本 0.33 的 CA 文件配置**

```
spec:
  mode: deployment
  config: |
    exporters:
      jaeger:
        endpoint: jaeger-production-collector-headless.tracing-system.svc:14250
        ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
```

  **OpenTelemetry 版本 0.41.1 的 CA 文件配置**

```
spec:
  mode: deployment
  config: |
    exporters:
      jaeger:
        endpoint: jaeger-production-collector-headless.tracing-system.svc:14250
        tls:
          ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
```

### 1.2.16.3. 程序错误修复

此发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

## 1.2.17. Red Hat OpenShift distributed tracing Platform 2.0 发行注记

### 1.2.17.1. Red Hat OpenShift distributed tracing Platform 2.0 中的组件版本

| Operator | 组件 | Version |
| --- | --- | --- |
| Red Hat OpenShift distributed tracing Platform (Jaeger) | Jaeger | 1.28.0 |

#### 1.2.17.2. 新功能及功能增强

此发行版本引进了以下新功能和增强：

- 重新将 Red Hat OpenShift Jaeger 作为 Red Hat OpenShift distributed tracing 平台。
- 将 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 更新至 Jaeger 1.28。在未来，Red Hat OpenShift distributed tracing 平台只支持 **stable** Operator 频道。独立发行版本的频道不再被支持。
- 为 Query 服务添加了对 OpenTelemetry 协议(OTLP)的支持。
- 引入了 OperatorHub 中出现的新分布式追踪图标。
- 包含对文档的滚动更新，以支持名称更改和新功能。

#### 1.2.17.3. 技术预览功能

此发行版本添加了 OpenTelemetry 的红帽构建作为技术预览，您使用红帽构建的 OpenTelemetry Operator 安装。Red Hat build of OpenTelemetry 基于 OpenTelemetry API 和工具。红帽构建的 OpenTelemetry 包括 OpenTelemetry Operator 和 Collector。您可以使用 Collector 在 OpenTelemetry 或 Jaeger 协议中接收 trace，并将 trace 数据发送到 Red Hat OpenShift distributed tracing 平台。目前还不支持 Collector 的其他功能。OpenTelemetry 收集器允许开发人员使用与供应商无关的 API 检测其代码，避免了供应商锁定并启用不断增长的可观察性工具生态系统。

#### 1.2.17.4. 程序错误修复

此发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

### 1.2.18. 获取支持

如果您在执行本文档所述的某个流程或 OpenShift Container Platform 时遇到问题，请访问 红帽客户门户网站。

通过红帽客户门户网站：

- 搜索或者浏览红帽知识库，了解与红帽产品相关的文章和解决方案。
- 提交问题单给红帽支持。
- 访问其他产品文档。

要识别集群中的问题，您可以在 OpenShift Cluster Manager 中使用 Insights。Insights 提供了问题的详细信息，并在有可用的情况下，提供了如何解决问题的信息。

如果您对本文档有任何改进建议，或发现了任何错误，请为相关文档组件提交 JIRA 问题。请提供具体详情，如章节名称和 OpenShift Container Platform 版本。

### 1.2.19. 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 2 章 分布式追踪架构

## 2.1. 分布式追踪架构

每次用户在某个应用程序中执行一项操作时，一个请求都会在所在的系统上执行，而这个系统可能需要几十个不同服务的共同参与才可以做出相应的响应。Red Hat OpenShift distributed tracing 平台可让您执行分布式追踪，在组成一个应用程序的多个微服务间记录请求的路径。

*分布式追踪*是用来将不同工作单元的信息关联起来的技术，通常是在不同进程或主机中执行的，以便理解分布式事务中的整个事件链。开发人员可以视觉化在大型微服务架构中调用的流程。它对理解序列化、并行性和延迟来源有价值。

Red Hat OpenShift distributed tracing 平台记录了在微服务的整个堆栈间执行单个请求，并将其显示为 trace。*trace*是系统的数据/执行路径。一个端到端的 trace 由一个或者多个 span 组成。

*span* 代表 Red Hat OpenShift distributed tracing 平台中的逻辑工作单元，它包含操作名称、操作的开始时间和持续时间，以及可能的标签和日志。span 可能会被嵌套并排序以模拟因果关系。

### 2.1.1. 分布式追踪概述

作为服务所有者，您可以使用分布式追踪来检测您的服务，以收集与服务架构相关的信息。您可以使用 Red Hat OpenShift distributed tracing 平台来监控、网络性能分析，并对现代、云原生的微服务应用程序中组件间的交互进行故障排除。

使用分布式追踪平台，您可以执行以下功能：

- 监控分布式事务
- 优化性能和延迟时间
- 执行根原因分析

### 2.1.2. Red Hat OpenShift distributed tracing Platform 功能

Red Hat OpenShift distributed tracing 平台提供以下功能：

- 与 Kiali 集成 - 当正确配置时，您可以从 Kiali 控制台查看分布式追踪平台数据。
- 高可伸缩性 - 分布式追踪平台后端设计具有单一故障点，而且能够按照业务需求进行扩展。
- 分布式上下文发布 – 允许您通过不同的组件连接数据以创建完整的端到端的 trace。
- 与 Zipkin 的后向兼容性 - Red Hat OpenShift distributed tracing platform 有 API，它能将其用作 Zipkin 的简易替代品，但红帽在此发行版本中不支持 Zipkin 的兼容性。

### 2.1.3. Red Hat OpenShift distributed tracing Platform 架构

Red Hat OpenShift distributed tracing 平台由多个组件组成，它们一起收集、存储和显示追踪数据。

- **Red Hat OpenShift distributed tracing Platform (Tempo)**- 此组件基于开源 Grafana Tempo 项目。
    - **网关** - 网关处理身份验证、授权和将请求转发到分布式或查询前端服务。

- **Distributor** - Distributor 接受多种格式（包括 Jaeger、OpenTelemetry 和 Zipkin）的 span。它通过哈希 **traceID** 并将分布式一致的哈希环路由到 Ingester。
- **Ingester** - Ingester 将 trace 批处理到块中，创建 bloom 过滤器和索引，然后将其全部刷新到后端。
- **Query Frontend** - Query Frontend 负责为传入的查询对搜索空间进行分片。然后，搜索查询会发送到 Queriers。Query Frontend 部署通过 Tempo Query sidecar 公开 Jaeger UI。
- **Querier** - Querier 负责在 Ingester 或后端存储中查找请求的 trace ID。根据参数，它可以查询 Ingesters，并从后端拉取 Bloom 索引，以便在对象存储中搜索块。
- **compactor** - Compactors 流块到后端存储中，以减少块总数。

- **红帽构建的 OpenTelemetry** - 此组件基于开源 OpenTelemetry 项目。
  - **OpenTelemetry Collector** - OpenTelemetry Collector 是一个与厂商无关的方式来接收、处理和导出遥测数据。OpenTelemetry Collector 支持开源可观察数据格式，如 Jaeger 和 Prometheus，发送到一个或多个开源或商业后端。Collector 是默认位置检测库来导出其遥测数据。
- **Red Hat OpenShift distributed tracing Platform (Jaeger)** - 此组件基于开源 Jaeger 项目。
  - **客户端** （Jaeger 客户端、跟踪器、报告程序、客户端库）- 分布式追踪平台 (Jaeger) 客户端是 OpenTracing API 的特定语言实施。它们可以用来为各种现有开源框架（如 Camel (Fuse)、Spring Boot (RHOAR)、MicroProfile (RHOAR/Thorntail)、Wilfly (EAP) 等提供分布式追踪工具。
  - **代理** （Jaeger 代理，Server Queue, Processor Workers）- 分布式追踪平台 (Jaeger) 代理是一个网络守护进程，侦听通过用户数据报协议(UDP)发送并发送到 Collector。这个代理应被放置在要管理的应用程序的同一主机上。这通常是通过容器环境（如 Kubernetes）中的 sidecar 来实现。
  - **Jaeger Collector** (Collector, Queue, Workers)- 与 Jaeger 代理类似，Jaeger Collector 接收 span，并将它们放置在内部队列中进行处理。这允许 Jaeger Collector 立即返回到客户端/代理，而不是等待 span 变为存储。
  - **Storage** (Data Store) - 收集器需要一个持久的存储后端。Red Hat OpenShift distributed tracing Platform (Jaeger) 提供了用于 span 存储的可插拔机制。Red Hat OpenShift distributed tracing Platform (Jaeger)支持 Elasticsearch 存储。
  - **Query** (Query Service) - Query 是一个从存储中检索 trace 的服务。
  - **Ingester** (Ingester Service)- Red Hat OpenShift distributed tracing 平台可以使用 Apache Kafka 作为 Collector 和实际的 Elasticsearch 后端存储之间的缓冲。Ingester 是一个从 Kafka 读取数据并写入 Elasticsearch 存储后端的服务。
  - **Jaeger 控制台** - 使用 Red Hat OpenShift distributed tracing 平台 (Jaeger) 用户界面，您可以视觉化您的分布式追踪数据。在搜索页面中，您可以查找 trace，并查看组成一个独立 trace 的 span 详情。

### 2.1.4. 其他资源

- Red Hat build of OpenTelemetry

# 第 3 章 分布式追踪平台(TEMPO)

## 3.1. 安装

安装分布式追踪平台 (Tempo) 需要 Tempo Operator，并选择最适合您的用例的部署类型：

- 对于微服务模式，在专用的 OpenShift 项目中部署 TempoStack 实例。
- 对于单体模式，在专用的 OpenShift 项目中部署 TempoMonolithic 实例。

**重要**

使用对象存储需要在部署 TempoStack 或 TempoMonolithic 实例前设置受支持的对象存储并为对象存储凭据创建一个 secret。

### 3.1.1. 安装 Tempo Operator

您可以使用 Web 控制台或命令行安装 Tempo Operator。

#### 3.1.1.1. 使用 Web 控制台安装 Tempo Operator

您可以通过 Web 控制台的 **Administrator** 视图安装 Tempo Operator。

**先决条件**

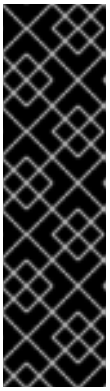
- 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform Web 控制台。
- 对于 Red Hat OpenShift Dedicated，您必须使用具有 **dedicated-admin** 角色的帐户登录。
- 您已完成由支持的供应商设置所需的对象存储：Red Hat OpenShift Data Foundation, MinIO, Amazon S3, Azure Blob Storage, Google Cloud Storage。如需更多信息，请参阅"对象存储设置"。

**警告**

对象存储是必需的，它没有包含在分布式追踪平台(Tempo) 中。在安装分布式追踪平台 (Tempo) 前，您必须通过受支持的供应商选择和设置对象存储。

**流程**

1. 进入 **Operators → OperatorHub** 并搜索 **Tempo Operator**。
2. 选择 **由红帽提供的 Tempo Operator**。



**重要**

以下选择是此 Operator 的默认预设置：

- **Update channel → stable**
- **Installation mode → All namespaces on the cluster**
- **Installed Namespace → openshift-tempo-operator**
- **Update approval → Automatic**

3. 选择 **Enable Operator recommended cluster monitoring on this Namespace**复选框。
4. 选择 **Install → Install → View Operator**。

**验证**

- 在已安装 Operator 页面的 **Details** 选项卡中，在 **ClusterServiceVersion details** 下验证安装 **Status** 是否为 **Succeeded**。

### 3.1.1.2. 使用 CLI 安装 Tempo Operator

您可以从命令行安装 Tempo Operator。

**先决条件**

- 集群管理员具有 **cluster-admin** 角色的活跃 OpenShift CLI (**oc**) 会话。

  **提示**

  - 确保您的 OpenShift CLI (**oc**) 版本为最新版本，并与您的 OpenShift Container Platform 版本匹配。
  - 运行 **oc login**:

    ```
    $ oc login --username=<your_username>
    ```

- 您已完成由支持的供应商设置所需的对象存储：Red Hat OpenShift Data Foundation, MinIO, Amazon S3, Azure Blob Storage, Google Cloud Storage。如需更多信息，请参阅"对象存储设置"。

**警告**

对象存储是必需的，它没有包含在分布式追踪平台(Tempo) 中。在安装分布式追踪平台 (Tempo) 前，您必须通过受支持的供应商选择和设置对象存储。

**流程**

1. 运行以下命令，为 Tempo Operator 创建项目：

```
$ oc apply -f - << EOF
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  labels:
    kubernetes.io/metadata.name: openshift-tempo-operator
    openshift.io/cluster-monitoring: "true"
  name: openshift-tempo-operator
EOF
```

2. 运行以下命令来创建 Operator 组：

```
$ oc apply -f - << EOF
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-tempo-operator
  namespace: openshift-tempo-operator
spec:
  upgradeStrategy: Default
EOF
```

3. 运行以下命令来创建订阅：

```
$ oc apply -f - << EOF
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: tempo-product
  namespace: openshift-tempo-operator
spec:
  channel: stable
  installPlanApproval: Automatic
  name: tempo-product
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

**验证**

- 运行以下命令检查 Operator 状态：

```
$ oc get csv -n openshift-tempo-operator
```

### 3.1.2. 安装 TempoStack 实例

您可以使用 Web 控制台或命令行安装 TempoStack 实例。

#### 3.1.2.1. 使用 Web 控制台安装 TempoStack 实例

您可以从 Web 控制台的 **Administrator** 视图安装 TempoStack 实例。

**先决条件**

- 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform Web 控制台。
- 对于 Red Hat OpenShift Dedicated，您必须使用具有 **dedicated-admin** 角色的帐户登录。
- 您已完成由支持的供应商设置所需的对象存储：Red Hat OpenShift Data Foundation, MinIO, Amazon S3, Azure Blob Storage, Google Cloud Storage。如需更多信息，请参阅"对象存储设置"。

**警告**

对象存储是必需的，它没有包含在分布式追踪平台(Tempo) 中。在安装分布式追踪平台 (Tempo) 前，您必须通过受支持的供应商选择和设置对象存储。

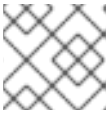
**流程**

1. 进入 **Home → Projects → Create Project**，为在后续步骤中创建的 TempoStack 实例创建一个项目。
2. 进入 **Workloads → Secrets → Create → From YAML**，在您为 TempoStack 实例创建的项目中为您的对象存储桶创建一个 secret。如需更多信息，请参阅"对象存储设置"。

   **Amazon S3 和 MinIO 存储的 secret 示例**

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: minio-test
   stringData:
     endpoint: http://minio.minio.svc:9000
     bucket: tempo
     access_key_id: tempo
     access_key_secret: <secret>
   type: Opaque
   ```

3. 创建 TempoStack 实例。

   **注意**

   您可以在同一集群中的独立项目中创建多个 TempoStack 实例。

   a. 进入 **Operators → Installed Operators**。

   b. 选择 **TempoStack → Create TempoStack → YAML view**。

   c. 在 **YAML 视图中**，自定义 **TempoStack** 自定义资源(CR)：

   ```
   apiVersion: tempo.grafana.com/v1alpha1
   kind: TempoStack
   ```

```
metadata:
  name: sample
  namespace: <project_of_tempostack_instance>
spec:
  storageSize: 1Gi
  storage:
    secret: (1)
      name: <secret_name> (2)
      type: <secret_provider> (3)
  template:
    queryFrontend:
      jaegerQuery:
        enabled: true
        ingress:
          route:
            termination: edge
          type: route
```

1. 您在第 2 步中创建的 secret，用于作为其中一个先决条件设置的对象存储。
2. secret **metadata** 中 **name** 的值。
3. 可接受的值是 **azure**（Azure Blob Storage），**gcs**（Google Cloud Storage）和 **s3**（Amazon S3, MinIO, 或 Red Hat OpenShift Data Foundation）。

### AWS S3 和 MinIO 存储的 TempoStack CR 示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: simplest
  namespace: <project_of_tempostack_instance>
spec:
  storageSize: 1Gi
  storage: (1)
    secret:
      name: minio-test
      type: s3
  resources:
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  template:
    queryFrontend:
      jaegerQuery: (2)
        enabled: true
        ingress:
          route:
            termination: edge
          type: route
```

1. 在本例中，对象存储被设置为先决条件之一，并在第 2 步中创建对象存储 secret。

2 本例中部署的堆栈被配置为接收通过 HTTP 和 OpenTelemetry 协议(OTLP)的 Jaeger Thrift，它允许使用 Jaeger UI 可视化数据。

d. 选择 **Create**。

**验证**

1. 使用 **Project:** 下拉列表选择 **TempoStack** 实例的项目。
2. 进入 **Operators → Installed Operators**，以验证 **TempoStack** 实例的 **Status** 是否为 **Condition: Ready**。
3. 进入 **Workloads → Pods**，以验证 **TempoStack** 实例的所有组件 pod 都在运行。
4. 访问 Tempo 控制台：
   a. 进入 **Networking → Routes** 和 **Ctrl+F**，以搜索 **tempo**。
   b. 在 **Location** 列中，打开 URL 以访问 Tempo 控制台。

**注意**

Tempo 控制台最初不会在 Tempo 控制台安装后显示 trace 数据。

### 3.1.2.2. 使用 CLI 安装 TempoStack 实例

您可以从命令行安装 TempoStack 实例。

**先决条件**

- 集群管理员具有 **cluster-admin** 角色的活跃 OpenShift CLI (**oc**) 会话。

  **提示**

  - 确保您的 OpenShift CLI (**oc**) 版本为最新版本，并与您的 OpenShift Container Platform 版本匹配。
  - 运行 **oc login** 命令：

    ```
    $ oc login --username=<your_username>
    ```

- 您已完成由支持的供应商设置所需的对象存储：Red Hat OpenShift Data Foundation, MinIO, Amazon S3, Azure Blob Storage, Google Cloud Storage。如需更多信息，请参阅"对象存储设置"。

**警告**

对象存储是必需的，它没有包含在分布式追踪平台(Tempo) 中。在安装分布式追踪平台 (Tempo) 前，您必须通过受支持的供应商选择和设置对象存储。

**流程**

1. 运行以下命令，为您将在后续步骤中创建的 TempoStack 实例创建您选择的项目：

```
$ oc apply -f - << EOF
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  name: <project_of_tempostack_instance>
EOF
```

2. 在您为 TempoStack 实例创建的项目中，运行以下命令来为您的对象存储桶创建一个 secret：

```
$ oc apply -f - << EOF
<object_storage_secret>
EOF
```

如需更多信息，请参阅"对象存储设置"。

**Amazon S3 和 MinIO 存储的 secret 示例**

```
apiVersion: v1
kind: Secret
metadata:
  name: minio-test
stringData:
  endpoint: http://minio.minio.svc:9000
  bucket: tempo
  access_key_id: tempo
  access_key_secret: <secret>
type: Opaque
```

3. 在为您创建的项目中创建一个 TempoStack 实例：

**注意**

您可以在同一集群中的独立项目中创建多个 TempoStack 实例。

a. 自定义 **TempoStack** 自定义资源(CR)：

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: sample
  namespace: <project_of_tempostack_instance>
spec:
  storageSize: 1Gi
  storage:
    secret: (1)
      name: <secret_name> (2)
      type: <secret_provider> (3)
  template:
    queryFrontend:
      jaegerQuery:
```

```
      enabled: true
      ingress:
        route:
          termination: edge
        type: route
```

1. 您在第 2 步中创建的 secret，用于作为其中一个先决条件设置的对象存储。
2. secret **metadata** 中 **name** 的值。
3. 可接受的值是 **azure**（Azure Blob Storage），**gcs**（Google Cloud Storage）和 **s3**（Amazon S3, MinIO, 或 Red Hat OpenShift Data Foundation）。

**AWS S3 和 MinIO 存储的 TempoStack CR 示例**

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: simplest
  namespace: <project_of_tempostack_instance>
spec:
  storageSize: 1Gi
  storage: 1
    secret:
      name: minio-test
      type: s3
  resources:
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  template:
    queryFrontend:
      jaegerQuery: 2
        enabled: true
        ingress:
          route:
            termination: edge
          type: route
```

1. 在本例中，对象存储被设置为先决条件之一，并在第 2 步中创建对象存储 secret。
2. 本例中部署的堆栈被配置为接收通过 HTTP 和 OpenTelemetry 协议(OTLP)的 Jaeger Thrift，它允许使用 Jaeger UI 可视化数据。

b. 运行以下命令来应用自定义 CR：

```
$ oc apply -f - << EOF
<tempostack_cr>
EOF
```

**验证**

1. 运行以下命令，验证所有 TempoStack 组件 的状态是否为 **Running**，并且条件为 **type: Ready**：

```
$ oc get tempostacks.tempo.grafana.com simplest -o yaml
```

2. 运行以下命令，验证所有 TempoStack 组件 pod 是否正在运行：

```
$ oc get pods
```

3. 访问 Tempo 控制台：

   a. 运行以下命令来查询路由详情：

```
$ oc get route
```

   b. 在网页浏览器中打开 **https://<route_from_previous_step>**。

**注意**

Tempo 控制台最初不会在 Tempo 控制台安装后显示 trace 数据。

### 3.1.3. 安装 TempoMonolithic 实例

**重要**

TempoMonolithic 实例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。

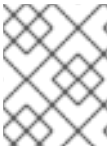
有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

您可以使用 Web 控制台或命令行安装 TempoMonolithic 实例。

**TempoMonolithic** 自定义资源(CR) 以单体模式创建 Tempo 部署。Tempo 部署的所有组件（如紧凑器、经销商、ingester、querier 和查询前端）都包含在一个容器中。

TempoMonolithic 实例支持将 trace 存储在内存中存储、持久性卷或对象存储。

在单体模式下部署临时是小型部署、演示、测试和作为 Red Hat OpenShift distributed tracing 平台 (Jaeger) 全体部署的迁移路径的首选。



**注意**

Tempo 的单体部署无法水平扩展。如果您需要水平扩展，请在微服务模式中将 **TempoStack** CR 用于 Tempo 部署。

#### 3.1.3.1. 使用 Web 控制台安装 TempoMonolithic 实例

**重要**

TempoMonolithic 实例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。

您可以从 web 控制台的 **Administrator** 视图安装 TempoMonolithic 实例。

**先决条件**

- 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform Web 控制台。
- 对于 Red Hat OpenShift Dedicated，您必须使用具有 **dedicated-admin** 角色的帐户登录。

**流程**

1. 进入 **Home → Projects → Create Project**，为在后续步骤中创建的 **TempoMonolithic** 实例创建一个项目。
2. 决定用于存储 trace 的存储类型：内存中存储、持久性卷或对象存储。

**重要**

对象存储不包括在分布式追踪平台(Tempo)中，需要由受支持的供应商设置对象存储： Red Hat OpenShift Data Foundation、MinIO、Amazon S3、Azure Blob Storage 或 Google Cloud Storage。

另外，选择对象存储需要在您为 **TempoMonolithic** 实例创建的项目中为您的对象存储桶创建一个 secret。您可以在 **Workloads → Secrets → Create → From YAML** 中执行此操作。

如需更多信息，请参阅"对象存储设置"。

**Amazon S3 和 MinIO 存储的 secret 示例**

```
apiVersion: v1
kind: Secret
metadata:
  name: minio-test
stringData:
  endpoint: http://minio.minio.svc:9000
  bucket: tempo
  access_key_id: tempo
  access_key_secret: <secret>
type: Opaque
```

3. 创建 **TempoMonolithic** 实例：

**注意**

您可以在同一集群的单独项目中创建多个 **TempoMonolithic** 实例。

a. 进入 **Operators → Installed Operators**。

b. 选择 **TempoMonolithic → Create TempoMonolithic → YAML view**。

c. 在 **YAML 视图中**，自定义 **TempoMonolithic** 自定义资源 (CR)。
以下 **TempoMonolithic** CR 创建一个 TempoMonolithic 部署，它通过 OTLP/gRPC 和 OTLP/HTTP 将 trace 存储在支持的存储中，并通过路由公开 Jaeger UI：

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic
metadata:
  name: <metadata_name>
  namespace: <project_of_tempomonolithic_instance>
spec:
  storage:
    traces:
      backend: <supported_storage_type> 1
      size: <value>Gi 2
      s3: 3
        secret: <secret_name> 4
  jaegerui:
    enabled: true 5
    route:
      enabled: true 6
```

1. 用于存储 trace 的存储类型：内存存储、持久性卷或对象存储。**tmpfs** 内存中存储的值是 **memory**。持久性卷的值是 **pv**。对象存储接受的值是 **s3**、**gcs** 或 **azure**，具体取决于使用的对象存储类型。

2. 内存大小：对于内存存储，这意味着 **tmpfs** 卷的大小，默认值为 **2Gi**。对于持久性卷，这意味着持久性卷声明的大小，默认值为 **10Gi**。对于对象存储，这意味着 Tempo WAL 的持久性卷声明的大小，默认值为 **10Gi**。

3. 可选：对于对象存储，对象存储的类型。接受的值包括 **s3**、**gcs** 和 **azure**，具体取决于使用的对象存储类型。

4. 可选：对于对象存储，存储 secret 的 **metadata** 中的 **name** 值。存储 secret 必须与 TempoMonolithic 实例位于同一个命名空间中，并包含 "Table 1 中指定的字段。"Object storage setup" 部分中所需的 secret 参数"。

5. 启用 Jaeger UI。

6. 启用为 Jaeger UI 创建路由。

d. 选择 **Create**。

**验证**

1. 使用 **Project:** 下拉列表，选择 **TempoMonolithic** 实例的项目。
2. 进入 **Operators → Installed Operators**，以验证 **TempoMonolithic** 实例的 **Status** 是否为 **Condition: Ready**。
3. 进入 **Workloads → Pods**，以验证 **TempoMonolithic** 实例的 pod 是否正在运行。

4. 访问 Jaeger UI：

   a. 进入 **Networking → Routes** 和 **Ctrl+F**，以搜索 **jaegerui**。

   **注意**

   Jaeger UI 使用 **tempo-<metadata_name_of_TempoMonolithic_CR>-jaegerui** 路由。

   b. 在 **Location** 列中，打开 URL 以访问 Jaeger UI。

5. 当 **TempoMonolithic** 实例的 pod 就绪时，您可以将 trace 发送到集群中的 **tempo-<metadata_name_of_TempoMonolithic_CR>:4317** (OTLP/gRPC) 和 **tempo-<metadata_name_of_TempoMonolithic_CR>:4318** (OTLP/HTTP) 端点。
   Tempo API 位于集群中的 **tempo-<metadata_name_of_TempoMonolithic_CR>:3200** 端点。

### 3.1.3.2. 使用 CLI 安装 TempoMonolithic 实例

**重要**

TempoMonolithic 实例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。

您可以从命令行安装 TempoMonolithic 实例。

**先决条件**

- 集群管理员具有 **cluster-admin** 角色的活跃 OpenShift CLI (**oc**) 会话。

  **提示**

  - 确保您的 OpenShift CLI (**oc**) 版本为最新版本，并与您的 OpenShift Container Platform 版本匹配。
  - 运行 **oc login** 命令：

    ```
    $ oc login --username=<your_username>
    ```

**流程**

1. 运行以下命令，为您将在后续步骤中创建的 TempoMonolithic 实例创建您选择的项目：

   ```
   $ oc apply -f - << EOF
   apiVersion: project.openshift.io/v1
   kind: Project
   metadata:
     name: <project_of_tempomonolithic_instance>
   EOF
   ```

2. 决定用于存储 trace 的存储类型：内存中存储、持久性卷或对象存储。

**重要**

对象存储不包括在分布式追踪平台(Tempo)中，需要由受支持的供应商设置对象存储： Red Hat OpenShift Data Foundation、MinIO、Amazon S3、Azure Blob Storage 或 Google Cloud Storage。

另外，选择对象存储需要在您为 TempoMonolithic 实例创建的项目中为您的对象存储桶创建一个 secret。您可以运行以下命令来完成此操作：

```
$ oc apply -f - << EOF
<object_storage_secret>
EOF
```

如需更多信息，请参阅"对象存储设置"。

**Amazon S3 和 MinIO 存储的 secret 示例**

```
apiVersion: v1
kind: Secret
metadata:
  name: minio-test
stringData:
  endpoint: http://minio.minio.svc:9000
  bucket: tempo
  access_key_id: tempo
  access_key_secret: <secret>
type: Opaque
```

3. 在您为其创建的项目中创建一个 TempoMonolithic 实例。

**提示**

您可以在同一集群的单独项目中创建多个 TempoMonolithic 实例。

a. 自定义 **TempoMonolithic** 自定义资源 (CR)。
以下 **TempoMonolithic** CR 创建一个 TempoMonolithic 部署，它通过 OTLP/gRPC 和 OTLP/HTTP 将 trace 存储在支持的存储中，并通过路由公开 Jaeger UI：

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic
metadata:
  name: <metadata_name>
  namespace: <project_of_tempomonolithic_instance>
spec:
  storage:
    traces:
      backend: <supported_storage_type> 1
      size: <value>Gi 2
      s3: 3
        secret: <secret_name> 4
  jaegerui:
```

```
    enabled: true 5
  route:
    enabled: true 6
```

1. 用于存储 trace 的存储类型：内存存储、持久性卷或对象存储。**tmpfs** 内存中存储的值是 **memory**。持久性卷的值是 **pv**。对象存储接受的值是 **s3**、**gcs** 或 **azure**，具体取决于使用的对象存储类型。
2. 内存大小：对于内存存储，这意味着 **tmpfs** 卷的大小，默认值为 **2Gi**。对于持久性卷，这意味着持久性卷声明的大小，默认值为 **10Gi**。对于对象存储，这意味着 Tempo WAL 的持久性卷声明的大小，默认值为 **10Gi**。
3. 可选：对于对象存储，对象存储的类型。接受的值包括 **s3**、**gcs** 和 **azure**，具体取决于使用的对象存储类型。
4. 可选：对于对象存储，存储 secret 的 **metadata** 中的 **name** 值。存储 secret 必须与 TempoMonolithic 实例位于同一个命名空间中，并包含 "Table 1 中指定的字段。"Object storage setup" 部分中所需的 secret 参数"。
5. 启用 Jaeger UI。
6. 启用为 Jaeger UI 创建路由。

b. 运行以下命令来应用自定义 CR：

```
$ oc apply -f - << EOF
<tempomonolithic_cr>
EOF
```

**验证**

1. 运行以下命令，验证所有 TempoMonolithic **components** 的状态是否为 **Running**，且 **conditions** 是 **type: Ready**

```
$ oc get tempomonolithic.tempo.grafana.com <metadata_name_of_tempomonolithic_cr> -o yaml
```

2. 运行以下命令，以验证 TempoMonolithic 实例的 pod 是否正在运行：

```
$ oc get pods
```

3. 访问 Jaeger UI：

a. 运行以下命令，查询 **tempo-<metadata_name_of_tempomonolithic_cr>-jaegerui** 路由的路由详情：

```
$ oc get route
```

b. 在网页浏览器中打开 **https://<route_from_previous_step>**。

4. 当 TempoMonolithic 实例的 pod 就绪时，您可以将 trace 发送到集群中的 **tempo-<metadata_name_of_tempomonolithic_cr>:4317** (OTLP/gRPC) 和 **tempo-<metadata_name_of_tempomonolithic_cr>:4318** (OTLP/HTTP) 端点。

Tempo API 位于集群中的 **tempo-<metadata_name_of_tempomonolithic_cr>:3200** 端点。

### 3.1.4. 对象存储设置

在设置受支持的对象存储时，您可以使用以下配置参数。

**表 3.1. 所需的 secret 参数**

| 存储供应商 |
| --- |
| Secret 参数 |
| Red Hat OpenShift Data Foundation |
| **name: tempostack-dev-odf # example**<br>**bucket: <bucket_name> # requires an ObjectBucketClaim**<br>**endpoint: https://s3.openshift-storage.svc**<br>**access_key_id: <data_foundation_access_key_id>**<br>**access_key_secret: <data_foundation_access_key_secret>** |
| MinIO |
| 请参阅 MinIO Operator。<br>**name: tempostack-dev-minio # example**<br>**bucket: <minio_bucket_name> # MinIO documentation**<br>**endpoint: <minio_bucket_endpoint>**<br>**access_key_id: <minio_access_key_id>**<br>**access_key_secret: <minio_access_key_secret>** |
| Amazon S3 |
| **name: tempostack-dev-s3 # example**<br>**bucket: <s3_bucket_name> # Amazon S3 documentation**<br>**endpoint: <s3_bucket_endpoint>**<br>**access_key_id: <s3_access_key_id>**<br>**access_key_secret: <s3_access_key_secret>** |
| Microsoft Azure Blob Storage |

| 存储供应商 |
| --- |
| **name: tempostack-dev-azure # example**<br>**container: <azure_blob_storage_container_name> # Microsoft Azure documentation**<br>**account_name: <azure_blob_storage_account_name>**<br>**account_key: <azure_blob_storage_account_key>** |
| Google Cloud Storage on Google Cloud Platform (GCP) |
| **name: tempostack-dev-gcs # example**<br>**bucketname: <google_cloud_storage_bucket_name> # requires a bucket created in a GCP project**<br>**key.json: <path/to/key.json> # requires a service account in the bucket's GCP project for GCP authentication** |

### 3.1.5. 其他资源

- 创建集群管理员
- OperatorHub.io
- 访问Web控制台
- 使用 Web 控制台从 OperatorHub 安装
- 从已安装的 Operator 创建应用程序
- OpenShift CLI 入门

## 3.2. 配置

Tempo Operator 使用自定义资源定义(CRD)文件来定义创建和部署分布式追踪平台(Tempo)资源时要使用的架构和配置设置。您可以安装默认配置或修改该文件。

### 3.2.1. 自定义部署

有关配置后端存储的详情，请参考 了解持久性存储 以及您选择的存储选项的适当配置主题。

#### 3.2.1.1. 默认配置选项

**TempoStack** 自定义资源 (CR)定义创建分布式追踪平台 (Tempo) 资源时要使用的架构和设置。您可以修改这些参数以根据您的业务需求自定义分布式追踪平台(Tempo)实现。

**通用 Tempo YAML 文件示例**

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
```

```
  name: name
spec:
  storage: {}
  resources: {}
  storageSize: 200M
  replicationFactor: 1
  retention: {}
  template:
    distributor:{}
    ingester: {}
    compactor: {}
    querier: {}
    queryFrontend: {}
    gateway: {}
```

**表 3.2. Tempo 参数**

| 参数 | 描述 | 值 | 默认值 |
|---|---|---|---|
| **apiVersion：** | 创建对象时要使用的 API 版本。 | **tempo.grafana.com/v1alpha1** | **tempo.grafana.com/v1alpha1** |
| **kind：** | 定义要创建的 Kubernetes 对象的种类。 | **tempo** | |
| **metadata：** | 唯一标识对象的数据，包括 **name** 字符串，**UID**，和可选的 **namespace**。 | | OpenShift Container Platform 会自动生成 **UID** 并使用创建对象的项目名称完成 **namespace**。 |
| **name：** | 对象的名称。 | TempoStack 实例的名称。 | **tempo-all-in-one-inmemory** |
| **spec：** | 要创建的对象的规格。 | 包含 TempoStack 实例的所有配置参数。当需要所有 Tempo 组件的通用定义时，会在 **spec** 节点下定义它。当定义与单个组件相关时，它将放置在 **spec/template/<component>** 节点下。 | N/A |
| **resources:** | 分配给 TempoStack 实例的资源。 | | |
| **storageSize ：** | ingester PVC 的存储大小。 | | |
| **replicationFactor:** | 复制因素的配置。 | | |

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| **retention:** | 保留 trace 的配置选项。 | | |
| **storage：** | 定义存储的配置选项。所有与存储相关的选项都必须放在 **storage** 下，而不是放在 **allInOne** 或其他组件选项下。 | | |
| **template.distributor:** | Tempo **distributor** 的配置选项。 | | |
| **template.ingester:** | Tempo **ingester** 的配置选项。 | | |
| **template.compactor:** | Tempo **compactor** 的配置选项。 | | |
| **template.querier:** | Tempo **querier** 的配置选项。 | | |
| **template.queryFrontend:** | Tempo **query-frontend** 的配置选项。 | | |
| **template.gateway:** | Tempo **gateway** 的配置选项。 | | |

### 最低要求配置

以下是使用默认设置创建分布式追踪平台(Tempo)部署所需的最小值：

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: simplest
spec:
  storage: 1
    secret:
      name: minio
      type: s3
  resources:
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  template:
    queryFrontend:
      jaegerQuery:
```

```
    enabled: true
  ingress:
    type: route
```

1 本节指定部署的对象存储后端，它需要一个含有凭据的 secret 才能访问对象存储。

#### 3.2.1.2. 存储配置

您可以在 **spec.storage** 下的 **TempoStack** 自定义资源中为分布式追踪平台(Tempo)配置对象存储。您可以从支持的多个存储供应商中选择。

**表 3.3. Tempo Operator 用来定义分布式追踪存储的一般存储参数**

| 参数 | 描述 | 值 | 默认值 |
|---|---|---|---|
| **spec.storage.secret.type** | 要在部署中使用的存储类型。 | **内存**。内存存储仅适用于开发、测试、演示和概念验证环境，因为数据在 pod 关闭时不会保留。 | **内存** |
| **storage.secretname** | 包含设置对象类型凭证的 secret 名称。 | | N/A |
| **storage.tls.caName** | CA 是包含 CA 证书的 **ConfigMap** 对象的名称。 | | |

**表 3.4. 所需的 secret 参数**

| 存储供应商 |
|---|
| Secret 参数 |
| Red Hat OpenShift Data Foundation |
| **name: tempostack-dev-odf # example**<br>**bucket: <bucket_name> # requires an ObjectBucketClaim**<br>**endpoint: https://s3.openshift-storage.svc**<br>**access_key_id: <data_foundation_access_key_id>**<br>**access_key_secret: <data_foundation_access_key_secret>** |
| MinIO |

| 存储供应商 |
| --- |
| 请参阅 MinIO Operator。<br>**name: tempostack-dev-minio # example**<br>**bucket: <minio_bucket_name> # MinIO documentation**<br>**endpoint: <minio_bucket_endpoint>**<br>**access_key_id: <minio_access_key_id>**<br>**access_key_secret: <minio_access_key_secret>** |
| Amazon S3 |
| **name: tempostack-dev-s3 # example**<br>**bucket: <s3_bucket_name> # Amazon S3 documentation**<br>**endpoint: <s3_bucket_endpoint>**<br>**access_key_id: <s3_access_key_id>**<br>**access_key_secret: <s3_access_key_secret>** |
| Microsoft Azure Blob Storage |
| **name: tempostack-dev-azure # example**<br>**container: <azure_blob_storage_container_name> # Microsoft Azure documentation**<br>**account_name: <azure_blob_storage_account_name>**<br>**account_key: <azure_blob_storage_account_key>** |
| Google Cloud Storage on Google Cloud Platform (GCP) |
| **name: tempostack-dev-gcs # example**<br>**bucketname: <google_cloud_storage_bucket_name> # requires a bucket created in a GCP project**<br>**key.json: <path/to/key.json> # requires a service account in the bucket's GCP project for GCP authentication** |

#### 3.2.1.3. 查询**配置**选项

分布式追踪平台 (Tempo) 的两个组件，即 querier 和 query frontend，用于管理查询。您可以配置这两个组件。

querier 组件在 ingesters 或后端存储中查找请求的 trace ID。根据设置的参数，querier 组件可以查询 ingesters，并从后端拉取 bloom 或索引，以便在对象存储中搜索块。querier 组件在 **GET /querier/api/traces/<trace_id>** 公开 HTTP 端点，但不预期直接使用。查询必须发送到查询前端。

表 3.5. querier 组件的配置参数

| 参数 | 描述 | 值 |
| --- | --- | --- |
| **nodeSelector** | node-selection 约束的简单形式。 | 类型：对象 |
| **replicas** | 为组件创建的副本数。 | 类型：整数；格式： int32 |
| **容限（tolerations）** | 特定于组件的 pod 容限。 | 类型：数组 |

查询前端组件负责为传入的查询对搜索空间进行分片。查询前端通过简单的 HTTP 端点公开 trace：**GET /api/traces/<trace_id>**。在内部，查询 frontend 组件将 **blockID** 空间分成可配置的分片数量，然后对这些请求进行队列。querier 组件通过流 gRPC 连接连接到查询 frontend 组件，以处理这些分片查询。

表 3.6. 查询前端组件的配置参数

| 参数 | 描述 | 值 |
| --- | --- | --- |
| **component** | 配置查询前端组件。 | 类型：对象 |
| **component.nodeSelector** | 节点选择约束的简单形式。 | 类型：对象 |
| **component.replicas** | 为查询前端组件创建的副本数。 | 类型：整数；格式： int32 |
| **component.tolerations** | 特定于查询前端组件的 Pod 容限。 | 类型：数组 |
| **jaegerQuery** | 特定于 Jaeger Query 组件的选项。 | 类型：对象 |
| **jaegerQuery.enabled** | 启用后，创建 Jaeger Query 组件**jaegerQuery**。 | 类型：布尔值 |
| **jaegerQuery.ingress** | Jaeger Query ingress 的选项。 | 类型：对象 |
| **jaegerQuery.ingress.annotations** | ingress 对象的注解。 | 类型：对象 |
| **jaegerQuery.ingress.host** | ingress 对象的主机名。 | 类型：字符串 |
| **jaegerQuery.ingress.ingressClassName** | IngressClass 集群资源的名称。定义哪个入口控制器提供此入口资源。 | 类型：字符串 |
| **jaegerQuery.ingress.route** | OpenShift 路由的选项。 | 类型：对象 |
| **jaegerQuery.ingress.route.termination** | 终止类型。默认为 **edge**。 | 类型：字符串 (enum: insecure, edge, passthrough, reencrypt) |

| 参数 | 描述 | 值 |
| --- | --- | --- |
| **jaegerQuery.ingress.type** | Jaeger Query UI 的 ingress 类型。支持的类型有 **ingress**、**route** 和 **none**。 | 类型：字符串 (enum: ingress, route) |
| **jaegerQuery.monitorTab** | monitor 选项卡配置。 | 类型：对象 |
| **jaegerQuery.monitorTab.enabled** | 在 Jaeger 控制台中启用 monitor 选项卡。必须配置 **PrometheusEndpoint**。 | 类型：布尔值 |
| **jaegerQuery.monitorTab.prometheusEndpoint** | 包含 span rate、error 和 duration (RED) 指标的 Prometheus 实例的端点。例如：**https://thanos-querier.openshift-monitoring.svc.cluster.local:9091**。 | 类型：字符串 |

**TempoStack CR 中的查询前端组件的配置示例**

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: simplest
spec:
  storage:
    secret:
      name: minio
      type: s3
  storageSize: 200M
  resources:
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  template:
    queryFrontend:
      jaegerQuery:
        enabled: true
        ingress:
          route:
            termination: edge
          type: route
```

#### 3.2.1.3.1. 其他资源

- 了解污点和容限

### 3.2.1.4. 在 Jaeger UI 中配置 monitor 选项卡

跟踪数据包含丰富的信息，数据在检测的语言和框架中规范化。因此，请求率、错误和持续时间(RED)指标可以从 trace 中提取。指标可以在 Jaeger 控制台的 **Monitor** 选项卡中视觉化。

指标派生自 OpenTelemetry Collector 中的 span，由 user-workload 监控堆栈中部署的 Prometheus 从 Collector 中提取。Jaeger UI 从 Prometheus 端点查询这些指标，并视觉化它们。

#### 3.2.1.4.1. OpenTelemetry Collector 配置

OpenTelemetry Collector 需要配置 **spanmetrics** 连接器，该连接器从 trace 中派生指标，并以 Prometheus 格式导出指标。

**OpenTelemetry Collector 自定义资源，用于 span RED**

```
kind: OpenTelemetryCollector
apiVersion: opentelemetry.io/v1alpha1
metadata:
  name: otel
spec:
  mode: deployment
  observability:
    metrics:
      enableMetrics: true (1)
  config: |
    connectors:
      spanmetrics: (2)
        metrics_flush_interval: 15s

    receivers:
      otlp: (3)
        protocols:
          grpc:
          http:

    exporters:
      prometheus: (4)
        endpoint: 0.0.0.0:8889
        add_metric_suffixes: false
        resource_to_telemetry_conversion:
          enabled: true # by default resource attributes are dropped

      otlp:
        endpoint: "tempo-simplest-distributor:4317"
        tls:
          insecure: true

    service:
      pipelines:
        traces:
          receivers: [otlp]
          exporters: [otlp, spanmetrics] (5)
        metrics:
          receivers: [spanmetrics] (6)
          exporters: [prometheus]
```

1. 创建 **ServiceMonitor** 自定义资源，以启用 Prometheus exporter 的提取。
2. Spanmetrics 连接器接收 trace 和 exports 指标。
3. OTLP 接收器在 OpenTelemetry 协议中接收 span。
4. Prometheus exporter 用于导出 Prometheus 格式的指标。
5. Spanmetrics 连接器在 trace 管道中被配置为 exporter。
6. Spanmetrics 连接器在 metrics 管道中配置为接收器。

#### 3.2.1.4.2. Tempo 配置

**TempoStack** 自定义资源必须指定以下内容： **Monitor** 选项卡已启用，Prometheus 端点则设置为 Thanos querier 服务，以从用户定义的监控堆栈查询数据。

带有启用的 Monitor 选项卡的 TempoStack 自定义资源

```
kind: TempoStack
apiVersion: tempo.grafana.com/v1alpha1
metadata:
  name: simplest
spec:
  template:
    queryFrontend:
      jaegerQuery:
        enabled: true
        monitorTab:
          enabled: true (1)
          prometheusEndpoint: https://thanos-querier.openshift-monitoring.svc.cluster.local:9091 (2)
        ingress:
          type: route
```

1. 在 Jaeger 控制台中启用 Monitoring 选项卡。
2. 来自 user-workload 监控的 Thanos Querier 的服务名称。

#### 3.2.1.4.3. Span RED 指标和警报规则

**spanmetrics** 连接器生成的指标可用于警报规则。例如，对于有关较慢的服务或定义服务级别目标(SLO)的警报，连接器会创建一个 **duration_bucket** 直方图和 **调用** 计数器指标。这些指标具有标识服务、API 名称、操作类型和其他属性的标签。

**表 3.7. 在 spanmetrics 连接器中创建的指标标签**

| 标签 | 描述 | 值 |
| --- | --- | --- |
| **service_name** | 由 **otel_service_name** 环境变量设置的服务名称。 | **frontend** |

| 标签 | 描述 | 值 |
| --- | --- | --- |
| **span_name** | 操作的名称。 | • /<br>• **/customer** |
| **span_kind** | 标识服务器、客户端、消息传递或内部操作。 | • **SPAN_KIND_SERVER**<br>• **SPAN_KIND_CLIENT**<br>• **SPAN_KIND_PRODUCER**<br>• **SPAN_KIND_CONSUMER**<br>• **SPAN_KIND_INTERNAL** |

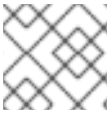
**PrometheusRule CR 示例，它为 SLO 定义了一个警告规则 - 在前端服务中，有 95% 的请求在 2000ms 内没有被处理完。**

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: span-red
spec:
  groups:
  - name: server-side-latency
    rules:
    - alert: SpanREDFrontendAPIRequestLatency
      expr: histogram_quantile(0.95, sum(rate(duration_bucket{service_name="frontend",
span_kind="SPAN_KIND_SERVER"}[5m])) by (le, service_name, span_name)) > 2000 (1)
      labels:
        severity: Warning
      annotations:
        summary: "High request latency on {{$labels.service_name}} and {{$labels.span_name}}"
        description: "{{$labels.instance}} has 95th request latency above 2s (current value: {{$value}}s)"
```

1. 这个表达式检查，是否 95% 的前端服务器响应时间值低于 2000 ms。时间范围 (**[5m]**) 必须至少是提取间隔的四倍，并且足以适应指标的变化。

#### 3.2.1.5. 多租户

Tempo Gateway 服务中提供了带有身份验证和授权的多租户。身份验证使用 OpenShift OAuth 和 Kubernetes **TokenReview** API。授权使用 Kubernetes **SubjectAccessReview** API。



**注意**

Tempo Gateway 服务只支持通过 OTLP/gRPC 处理 trace。不支持 OTLP/HTTP。

**带有两个租户的 Tempo CR 示例， dev 和 prod**

```
apiVersion: tempo.grafana.com/v1alpha1
kind:  TempoStack
metadata:
  name: simplest
spec:
  tenants:
    mode: openshift (1)
    authentication: (2)
      - tenantName: dev (3)
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa" (4)
      - tenantName: prod
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfb"
  template:
    gateway:
      enabled: true (5)
    queryFrontend:
      jaegerQuery:
        enabled: true
```

1. 必须设置为 **openshift**。
2. 租户列表。
3. 租户名称。在选择数据时，必须在 **X-Scope-OrgId** 标头中提供。
4. 唯一的租户 ID。
5. 启用执行身份验证和授权的网关。Jaeger UI 通过 **http://<gateway-ingress>/api/traces/v1/<tenant-name>/search** 公开。

授权配置使用 Kubernetes 基于角色的访问控制(RBAC)的 **ClusterRole** 和 **ClusterRoleBinding**。默认情况下，任何用户都没有读取或写入权限。

**允许经过身份验证的用户读取 dev 和 prod 租户的 trace 数据的读取 RBAC 配置示例**

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tempostack-traces-reader
rules:
  - apiGroups:
      - 'tempo.grafana.com'
    resources: (1)
      - dev
      - prod
    resourceNames:
      - traces
```

```
    verbs:
      - 'get' 2
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tempostack-traces-reader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tempostack-traces-reader
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated 3
```

1 列出租户。

2 **get** 值可启用读取操作。

3 授予所有经过身份验证的用户的 trace 数据读取权限。

**允许 otel-collector 服务帐户编写 dev 租户的 trace 数据的写入 RBAC 配置示例**

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: otel-collector 1
  namespace: otel
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tempostack-traces-write
rules:
  - apiGroups:
      - 'tempo.grafana.com'
    resources: 2
      - dev
    resourceNames:
      - traces
    verbs:
      - 'create' 3
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tempostack-traces
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tempostack-traces-write
subjects:
```

```
- kind: ServiceAccount
  name: otel-collector
  namespace: otel
```

1. 导出 trace 数据时要使用的客户端的服务帐户名称。客户端必须将服务帐户令牌 **/var/run/secrets/kubernetes.io/serviceaccount/token** 发送为 bearer 令牌标头。
2. 列出租户。
3. **create** 值启用写入操作。

追踪数据可以从 OpenTelemetry 收集器发送到 Tempo 实例，该收集器使用带有 RBAC 的服务帐户来写入数据。

**OpenTelemetry CR 配置示例**

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: cluster-collector
  namespace: tracing-system
spec:
  mode: deployment
  serviceAccount: otel-collector
  config: |
    extensions:
      bearertokenauth:
        filename: "/var/run/secrets/kubernetes.io/serviceaccount/token"
    exporters:
      otlp/dev:
        endpoint: tempo-simplest-gateway.tempo.svc.cluster.local:8090
        tls:
          insecure: false
          ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
        auth:
          authenticator: bearertokenauth
        headers:
          X-Scope-OrgID: "dev"
    service:
      extensions: [bearertokenauth]
      pipelines:
        traces:
          exporters: [otlp/dev]
```

### 3.2.2. 配置监控和警报

Tempo Operator 支持每个 TempoStack 组件的监控和警报，如经销商、ingester 等，并公开有关 Operator 本身的升级和操作指标。

#### 3.2.2.1. 配置 TempoStack 指标和警报

您可以启用 TempoStack 实例的指标和警报。

**先决条件**

- 在集群中启用对用户定义的项目的监控。

**流程**

1. 要启用 TempoStack 实例的指标，请将 **spec.observability.metrics.createServiceMonitors** 字段设置为 **true** ：

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
spec:
  observability:
    metrics:
      createServiceMonitors: true
```

2. 要为 TempoStack 实例启用警报，请将 **spec.observability.metrics.createPrometheusRules** 字段设置为 **true** ：

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
spec:
  observability:
    metrics:
      createPrometheusRules: true
```

验证

您可以使用 Web 控制台的 **Administrator** 视图来验证配置是否成功：

1. 进入 **Observe → Targets**，过滤 **Source: User**, 检查 **ServiceMonitors**（格式为 **tempo-<instance_name>-<component>**）的状态为 **Up**。

2. 要验证警报是否已正确设置，请转至 **Observe → Alerting → Alerting rules**，过滤 **Source: User**，并检查 TempoStack 实例组件的 **Alert 规则** 是否可用。

#### 3.2.2.1.1. 其他资源

- 为用户定义的项目启用监控

### 3.2.2.2. 配置 Tempo Operator 指标和警报

从 web 控制台安装 Tempo Operator 时，您可以选择 **Enable Operator recommended cluster monitoring on this Namespace** 复选框，它允许创建 Tempo Operator 的指标和警报。

如果在安装过程中没有选择复选框，您可以在安装 Tempo Operator 后手动启用指标和警报。

**流程**

- 在安装了 Tempo Operator 的项目中添加 **openshift.io/cluster-monitoring: "true"** 标签，默认为 **openshift-tempo-operator**。

验证

您可以使用 Web 控制台的 **Administrator** 视图来验证配置是否成功：

1. 进入 **Observe → Targets**，过滤 **Source: Platform**，并搜索 **tempo-operator**，它必须具有 **Up** 状态。
2. 要验证警报是否已正确设置，请转至 **Observe → Alerting → Alerting rules**，过滤 **Source: Platform**，再找到 **Tempo Operator** 的 **Alert** 规则。

## 3.3. 升级

对于版本升级，Tempo Operator 使用 Operator Lifecycle Manager (OLM)，用于控制集群中 Operator 的安装、升级和基于角色的访问控制(RBAC)。

OLM 默认在 OpenShift Container Platform 中运行。OLM 可以查询可用的 Operator 以及已安装的 Operator 的升级。

当 Tempo Operator 升级到新版本时，它会扫描它管理的 TempoStack 实例，并将其升级到与 Operator 新版本对应的版本。

### 3.3.1. 其他资源

- Operator Lifecycle Manager 概念和资源
- 更新安装的 Operator

## 3.4. 删除

从 OpenShift Container Platform 集群中删除 Red Hat OpenShift distributed tracing Platform (Tempo) 的步骤如下：

1. 关闭所有分布式追踪平台(Tempo) pod。
2. 删除任何 TempoStack 实例。
3. 删除 Tempo Operator。

### 3.4.1. 使用 Web 控制台删除

您可以在 web 控制台的 **Administrator** 视图中删除 TempoStack 实例。

**先决条件**

- 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform Web 控制台。
- 对于 Red Hat OpenShift Dedicated，您必须使用具有 **dedicated-admin** 角色的帐户登录。

**流程**

1. 进入 **Operators → Installed Operators → Tempo Operator → TempoStack**。
2. 要删除 TempoStack 实例，请选择  **→ Delete TempoStack → Delete**。
3. 可选：删除 Tempo Operator。

### 3.4.2. 使用 CLI 删除

您可以在命令行中删除 TempoStack 实例。

**先决条件**

- 集群管理员具有 **cluster-admin** 角色的活跃 OpenShift CLI (**oc**) 会话。

  **提示**

  - 确保您的 OpenShift CLI (**oc**) 版本为最新版本，并与您的 OpenShift Container Platform 版本匹配。
  - 运行 **oc login**:

    ```
    $ oc login --username=<your_username>
    ```

**流程**

1. 运行以下命令，获取 TempoStack 实例的名称：

   ```
   $ oc get deployments -n <project_of_tempostack_instance>
   ```

2. 运行以下命令来删除 TempoStack 实例：

   ```
   $ oc delete tempo <tempostack_instance_name> -n <project_of_tempostack_instance>
   ```

3. 可选：删除 Tempo Operator。

**验证**

1. 运行以下命令，以验证输出中没有找到 TempoStack 实例，这表示其删除成功：

   ```
   $ oc get deployments -n <project_of_tempostack_instance>
   ```

### 3.4.3. 其他资源

- 从集群中删除 Operator
- OpenShift CLI 入门

# 第 4 章 分布式追踪平台(JAEGER)

## 4.1. 安装

**重要**

Red Hat OpenShift distributed tracing Platform (Jaeger) 是一个已弃用的功能。弃用的功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。

有关 OpenShift Container Platform 中已弃用或删除的主要功能的最新列表，请参阅 OpenShift Container Platform 发行注记中 *已弃用和删除的功能*部分。

您可以通过以下两种方式之一在 OpenShift Container Platform 上安装 Red Hat OpenShift distributed tracing 平台：

- 作为 Red Hat OpenShift Service Mesh 的一部分安装 Red Hat OpenShift distributed tracing 平台。Service Mesh 安装中默认包含了分布式追踪。要将 Red Hat OpenShift distributed tracing 平台作为服务网格的一部分安装，请按照 Red Hat Service Mesh 安装 说明操作。您必须在与服务网格相同的命名空间中安装 Red Hat OpenShift distributed tracing 平台，即 **ServiceMeshControlPlane** 和 Red Hat OpenShift distributed tracing 平台资源必须位于同一命名空间中。

- 如果您不想安装服务网格，您可以使用 Red Hat OpenShift distributed tracing platform Operator 来自行安装分布式追踪平台。要在没有服务网格的情况下安装 Red Hat OpenShift distributed tracing 平台，请使用以下说明。

### 4.1.1. 先决条件

在安装 Red Hat OpenShift distributed tracing 平台前，请查看安装活动，并确保满足先决条件：

- 您的红帽帐户中拥有活跃的 OpenShift Container Platform 订阅。如果您没有相关订阅，请联络您的销售代表以获得更多信息。

- 查看 OpenShift Container Platform 4.16 概述。

- 安装 OpenShift Container Platform 4.16。
  - 在 AWS 上安装 OpenShift Container Platform 4.16
  - 在用户置备的 AWS 上安装 OpenShift Container Platform 4.16
  - 在裸机上安装 OpenShift Container Platform 4.16
  - 在 vSphere 上安装 OpenShift Container Platform 4.16

- 安装与 OpenShift Container Platform 版本匹配的 **oc** CLI 工具版本，并将其添加到您的路径中。

- 具有 **cluster-admin** 角色的帐户。

### 4.1.2. Red Hat OpenShift distributed tracing 平台安装概述

安装 Red Hat OpenShift distributed tracing 平台的步骤如下：

- 查看文档并确定您的部署策略。
- 如果您的部署策略需要持久性存储，请通过 OperatorHub 安装 OpenShift Elasticsearch Operator。
- 通过 OperatorHub 安装 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。
- 修改自定义资源 YAML 文件，以支持您的部署策略。
- 将一个或多个 Red Hat OpenShift distributed tracing 平台(Jaeger)实例部署到 OpenShift Container Platform 环境中。

### 4.1.3. 安装 OpenShift Elasticsearch Operator

默认的 Red Hat OpenShift distributed tracing 平台(Jaeger)部署使用内存存储，因为它旨在为评估 Red Hat OpenShift distributed tracing 平台、提供演示或在测试环境中使用 Red Hat OpenShift distributed tracing 平台(Jaeger)的用户快速安装。如果您计划在生产环境中使用 Red Hat OpenShift distributed tracing 平台 (Jaeger)，则必须安装并配置持久性存储选项，即 Elasticsearch。

**先决条件**

- 访问 OpenShift Container Platform web 控制台。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 **dedicated-admin** 角色的帐户。

**警告**

不要安装 Operators 的 Community 版本。不支持社区 Operator。

**注意**

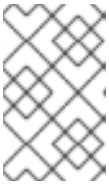
如果您已经安装了 OpenShift Elasticsearch Operator 作为 OpenShift Logging 的一部分，则不需要再次安装 OpenShift Elasticsearch Operator。Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 使用已安装的 OpenShift Elasticsearch Operator 创建 Elasticsearch 实例。

**流程**

1. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform web 控制台。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 **dedicated-admin** 角色的帐户。
2. 导航至 **Operators → OperatorHub**。
3. 在过滤器框中键入 **Elasticsearch** 以找到 OpenShift Elasticsearch Operator。
4. 点由红帽提供的 **OpenShift Elasticsearch Operator** 来显示有关 Operator 的信息。
5. 点击 **Install**。

6. 在 **Install Operator** 页中，选择 **stable** Update Channel。这可在发布新版本时自动更新您的 Operator。

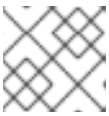
7. 接受默认的 **All namespaces on the cluster (default)**。这会在默认的 **openshift-operators-redhat** 项目中安装 Operator，并使 Operator 可供集群中的所有项目使用。



**注意**

Elasticsearch 安装需要 OpenShift Elasticsearch Operator 的 **openshift-operators-redhat** 命名空间。其他 Red Hat OpenShift distributed tracing Platform Operator 安装在 **openshift-operators** 命名空间中。

8. 接受默认的 **Automatic** 批准策略。默认情况下，当这个 Operator 的新版本可用时，Operator Lifecycle Manager(OLM)将自动升级 Operator 的运行实例，而无需人为干预。如果选择**手动** 更新，则当有新版 Operator 可用时，OLM 会创建更新请求。作为集群管理员，您必须手动批准该更新请求，才可将 Operator 更新至新版本。



**注意**

**Manual** 批准策略需要具有适当凭证的用户批准 Operator 的安装和订阅过程。

9. 点击 **Install**。

10. 在 **Installed Operators** 页面中，选择 **openshift-operators-redhat** 项目。等待 OpenShift Elasticsearch Operator 的 **InstallSucceeded** 状态再继续。

### 4.1.4. 安装 Red Hat OpenShift distributed tracing Platform Operator

您可以通过 OperatorHub 安装 Red Hat OpenShift distributed tracing Platform Operator。

默认情况下，Operator 安装在 **openshift-operators** 项目中。

**先决条件**

- 访问 OpenShift Container Platform web 控制台。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 **dedicated-admin** 角色的帐户。
- 如果需要持久性存储，则必须在安装 Red Hat OpenShift distributed tracing Platform Operator 前安装 OpenShift Elasticsearch Operator。

**流程**

1. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform web 控制台。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 **dedicated-admin** 角色的帐户。

2. 进入 **Operators → OperatorHub**。

3. 通过在搜索字段中输入分布式追踪平台来搜索 Red Hat OpenShift **distributed tracing Platform** Operator。

4. 选择 **Red Hat OpenShift distributed tracing platform** Operator，它由红帽提供，显示 Operator 的信息。

5. 点 **Install**。
6. 对于 **Install Operator** 页面中的 **Update 频道**，在发布新版本时，选择 **stable** 来自动更新 Operator。
7. 接受默认的 **All namespaces on the cluster (default)**。这会在默认的 **openshift-operators** 项目中安装 Operator，并使其可以被集群中的所有项目使用。
8. 接受默认的 **Automatic** 批准策略。

**注意**

如果您接受此默认值，Operator Lifecycle Manager (OLM) 会在有新版 Operator 可用时自动升级此 Operator 的运行实例。

如果选择**手动**更新，OLM 会在有新版 Operator 可用时创建一个更新请求。要将 Operator 更新至新版本，必须以集群管理员身份手动批准更新请求。**Manual** 批准策略要求集群管理员手动批准 Operator 安装和订阅。

9. 点 **Install**。
10. 导航到 **Operators → Installed Operators**。
11. 在 **Installed Operators** 页面中，选择 **openshift-operators** 项目。等待 Red Hat OpenShift distributed tracing Platform Operator 的 **Succeeded** 状态继续。

## 4.2. 配置

**重要**

Red Hat OpenShift distributed tracing Platform (Jaeger) 是一个已弃用的功能。弃用的功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。

有关 OpenShift Container Platform 中已弃用或删除的主要功能的最新列表，请参阅 OpenShift Container Platform 发行注记中 *已弃用和删除的功能*部分。

Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 使用自定义资源定义(CRD)文件来定义创建和部署分布式追踪平台(Jaeger)资源时要使用的架构和配置设置。您可以安装默认配置或修改该文件。

如果您已将分布式追踪平台作为 Red Hat OpenShift Service Mesh 的一部分安装，您可以作为 ServiceMeshControlPlane 的一部分执行基本配置，但为了完全控制，您必须配置 Jaeger CR，然后在 ServiceMeshControlPlane 中引用您的分布式追踪配置文件。

Red Hat OpenShift distributed tracing Platform (Jaeger)有预定义的部署策略。您可以在自定义资源文件中指定一个部署策略。当您创建分布式追踪平台(Jaeger)实例时，Operator 会使用此配置文件创建部署所需的对象。

**Jaeger 自定义资源文件显示部署策略**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
```

```
  name: MyConfigFile
spec:
  strategy: production (1)
```

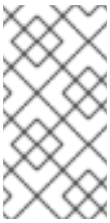
(1) 部署策略。

### 4.2.1. 支持的部署策略

Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 目前支持以下部署策略：

**allInOne**

- 此策略主要用于开发、测试和演示目的，它不用于生产环境。主要的后端组件 Agent、Collector 和 Query 服务都打包成单一可执行文件，默认为使用内存存储。



**注意**

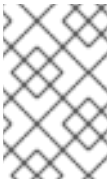
内存存储不是持久性的，这意味着如果分布式追踪平台(Jaeger)实例关闭、重启或被替换，您的 trace 数据将会丢失。此外，内存存储无法扩展，因为每个 Pod 都有自己的内存。对于持久性存储，您必须使用 **production** 或 **streaming** 策略，这些策略使用 Elasticsearch 作为默认存储。

**production**

production 策略主要用于生产环境，在生产环境中，对 trace 数据进行长期存储非常重要，同时需要更容易扩展和高度可用的构架。因此，每个后端组件都将单独部署。Agent 可以作为检测应用程序上的 sidecar 注入。Query 和 Collector 服务被配置为使用一个受支持的存储类型 - 当前为 Elasticsearch。可以根据性能和恢复能力的需要提供每个组件的多个实例。

**streaming**

streaming 策略旨在通过提供在 Collector 和 Elasticsearch 后端存储之间有效处的流传输功能来增强 production 策略。这样做的好处是在高负载情况下降低后端存储压力，并允许其他 trace 后处理功能直接从流传输平台 (AMQ Streams/ Kafka) 中利用实时 span 数据。

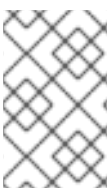


**注意**

- streaming 策略需要额外的 AMQ Streams 订阅。
- 目前 IBM Z® 不支持 streaming 部署策略。

### 4.2.2. 从 Web 控制台部署分布式追踪平台默认策略

自定义资源定义(CRD)定义部署 Red Hat OpenShift distributed tracing 平台实例时使用的配置。默认 CR 名为 **jaeger-all-in-one-inmemory**，它配置为使用最少资源，以确保您可以在默认的 OpenShift Container Platform 安装中成功安装它。您可以使用此默认配置创建使用 **AllInOne** 部署策略的 Red Hat OpenShift distributed tracing 平台(Jaeger)实例，或者您可以定义自己的自定义资源文件。



**注意**

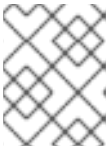
内存存储不是持久性的。如果 Jaeger pod 关闭、重启或被替换，您的 trace 数据将会丢失。对于持久性存储，您必须使用 **production** 或 **streaming** 策略，这些策略使用 Elasticsearch 作为默认存储。

**先决条件**

- 已安装 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。
- 您已查看了如何自定义部署的说明。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

**步骤**

1. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform web 控制台。
2. 创建一个新项目，如 **tracing-system**。



**注意**

如果您要作为 Service Mesh 的一部分安装，则分布式追踪平台资源必须与 **ServiceMeshControlPlane** 资源位于同一个命名空间中，如 **istio-system**。

   a. 进入 **Home → Projects**。
   b. 点击 **Create Project**。
   c. 在 **Name** 字段中输入 **tracing-system**。
   d. 点 **Create**。

3. 导航到 **Operators → Installed Operators**。
4. 如有必要，从 **Project** 菜单中选择 **tracing-system**。您可能需要等待一些时间，让 Operator 复制到新项目中。
5. 点 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。在 **Details** 标签页中的 **Provided APIs** 下，Operator 提供了一个单个链接。
6. 在 **Jaeger** 下，点 **Create Instance**。
7. 在 **Create Jaeger** 页面上，要使用默认值进行安装，请点 **Create** 来创建分布式追踪平台(Jaeger)实例。
8. 在 **Jaegers** 页面上，点分布式追踪平台(Jaeger)实例的名称，如 **jaeger-all-in-one-inmemory**。
9. 在 **Jaeger Details** 页面上，点击 **Resources** 选项卡。等待 pod 的状态变为"Running"再继续操作。

#### 4.2.2.1. 通过 CLI 部署分布式追踪平台默认策略

按照以下步骤，从命令行创建分布式追踪平台(Jaeger)实例。

**先决条件**

- 已安装并验证 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。
- 您已查看了如何自定义部署的说明。
- 您可以访问与 OpenShift Container Platform 版本匹配的 OpenShift CLI(**oc**)。

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

**流程**

1. 运行以下命令，以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform CLI：

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:8443
```

2. 运行以下命令，创建一个名为 **tracing-system** 的新项目：

```
$ oc new-project tracing-system
```

3. 创建一个名为 **jaeger.yaml** 的自定义资源文件，其中包含以下文本：

**示例 Jaeger-all-in-one.yaml**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```

4. 运行以下命令来部署分布式追踪平台(Jaeger)：

```
$ oc create -n tracing-system -f jaeger.yaml
```

5. 在安装过程中运行以下命令来监控 pod 的进度：

```
$ oc get pods -n tracing-system -w
```

安装过程完成后，输出类似以下示例：

```
NAME                                        READY   STATUS    RESTARTS   AGE
jaeger-all-in-one-inmemory-cdff7897b-qhfdx  2/2     Running   0          24s
```

### 4.2.3. 从 Web 控制台部署分布式追踪平台生产策略

**production** 部署策略主要用于生产环境，它需要更具扩展性和高度可用的架构，在此情况下，对 trace 数据进行长期存储非常重要。

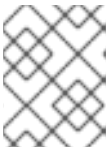
**先决条件**

- 已安装 OpenShift Elasticsearch Operator。
- 已安装 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。
- 您已查看了如何自定义部署的说明。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

**步骤**

1. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform web 控制台。

2. 创建一个新项目，如 **tracing-system**。



**注意**

如果您要作为 Service Mesh 的一部分安装，则分布式追踪平台资源必须与 **ServiceMeshControlPlane** 资源位于同一个命名空间中，如 **istio-system**。

   a. 浏览至 **Home → Project**。

   b. 点击 **Create Project**。

   c. 在 **Name** 字段中输入 **tracing-system**。

   d. 点 **Create**。

3. 导航到 **Operators → Installed Operators**。

4. 如有必要，从 **Project** 菜单中选择 **tracing-system**。您可能需要等待一些时间，让 Operator 复制到新项目中。

5. 点 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。在 **Overview** 选项卡上的 **Provided APIs** 下，Operator 提供了单个链接。

6. 在 **Jaeger** 下，点 **Create Instance**。

7. 在 **Create Jaeger** 页面上，将默认的 **all-in-one** YAML 文本替换为您的生产环境的 YAML 配置，例如：

**使用 Elasticsearch 的示例 jaeger-production.yaml 文件**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-production
  namespace:
spec:
  strategy: production
  ingress:
    security: oauth-proxy
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 3
      redundancyPolicy: SingleRedundancy
    esIndexCleaner:
      enabled: true
      numberOfDays: 7
      schedule: 55 23 * * *
    esRollover:
      schedule: '*/30 * * * *'
```

8. 点 **Create** 创建分布式追踪平台(Jaeger)实例。

9. 在 **Jaegers** 页面上，点分布式追踪平台(Jaeger)实例的名称，如 **jaeger-prod-elasticsearch**。

10. 在 **Jaeger Details** 页面上，点击 **Resources** 选项卡。等到所有 Pod 的状态变为“Running”再继续操作。

#### 4.2.3.1. 通过 CLI 部署分布式追踪平台生产策略

按照以下步骤，从命令行创建分布式追踪平台(Jaeger)实例。

**先决条件**

- 已安装 OpenShift Elasticsearch Operator。
- 已安装 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。
- 您已查看了如何自定义部署的说明。
- 您可以访问与 OpenShift Container Platform 版本匹配的 OpenShift CLI(**oc**)。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

**流程**

1. 运行以下命令，以具有 **cluster-admin** 角色的用户身份登录到 OpenShift CLI (**oc**)：

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:8443
```

2. 运行以下命令，创建一个名为 **tracing-system** 的新项目：

```
$ oc new-project tracing-system
```

3. 创建一个名为 **jaeger-production.yaml** 的自定义资源文件，其中包含上一步中的示例文件文本。
4. 运行以下命令来部署分布式追踪平台(Jaeger)：

```
$ oc create -n tracing-system -f jaeger-production.yaml
```

5. 在安装过程中运行以下命令来监控 pod 的进度：

```
$ oc get pods -n tracing-system -w
```

安装过程完成后，您会看到类似以下示例的输出：

```
NAME                                                  READY   STATUS    RESTARTS   AGE
elasticsearch-cdm-jaegersystemjaegerproduction-1-6676cf568gwhlw   2/2     Running   0
10m
elasticsearch-cdm-jaegersystemjaegerproduction-2-bcd4c8bf5l6g6w   2/2     Running   0
10m
elasticsearch-cdm-jaegersystemjaegerproduction-3-844d6d9694hhst   2/2     Running   0
10m
jaeger-production-collector-94cd847d-jwjlj                1/1     Running   3          8m32s
jaeger-production-query-5cbfbd499d-tv8zf                  3/3     Running   3          8m32s
```

### 4.2.4. 从 Web 控制台部署分布式追踪平台流策略

**streaming** 部署策略主要用于生产环境，在生产环境中需要更具扩展性和高度可用的架构，其中，对 trace 数据进行长期存储非常重要。

**streaming** 策略提供了位于 Collector 和 Elasticsearch 存储之间的流功能。这在高负载情况下降低了存储压力，并允许其他 trace 后处理功能直接从 Kafka streaming 平台利用实时 span 数据。

**注意**

streaming 策略需要额外的 AMQ Streams 订阅。如果您没有 AMQ Streams 订阅，请联络您的销售代表以了解更多信息。

**注意**

目前 IBM Z® 不支持 streaming 部署策略。

**先决条件**

- 已安装 AMQ Streams Operator。如果使用 1.4.0 或更高版本，您可以使用自助置备。否则，您必须创建 Kafka 实例。
- 已安装 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。
- 您已查看了如何自定义部署的说明。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

**步骤**

1. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform web 控制台。
2. 创建一个新项目，如 **tracing-system**。

**注意**

如果您要作为 Service Mesh 的一部分安装，则分布式追踪平台资源必须与 **ServiceMeshControlPlane** 资源位于同一个命名空间中，如 **istio-system**。

   a. 浏览至 **Home** → **Project**。
   b. 点击 **Create Project**。
   c. 在 **Name** 字段中输入 **tracing-system**。
   d. 点 **Create**。
3. 导航到 **Operators** → **Installed Operators**。
4. 如有必要，从 **Project** 菜单中选择 **tracing-system**。您可能需要等待一些时间，让 Operator 复制到新项目中。
5. 点 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。在 **Overview** 选项卡上的 **Provided APIs** 下，Operator 提供了单个链接。
6. 在 **Jaeger** 下，点 **Create Instance**。

7. 在 **Create Jaeger** 页面上，将默认的 **all-in-one** YAML 文本替换为您的流传输 YAML 配置，例如：

**示例 Jaeger-streaming.yaml 文件**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-streaming
spec:
  strategy: streaming
  collector:
    options:
      kafka:
        producer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092 (1)
  storage:
    type: elasticsearch
  ingester:
    options:
      kafka:
        consumer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
```

(1) 如果没有定义代理，AMQStreams 1.4.0+ self-provisions Kafka。

8. 点 **Create** 创建分布式追踪平台(Jaeger)实例。

9. 在 **Jaegers** 页面上，点分布式追踪平台(Jaeger)实例的名称，如 **jaeger-streaming**。

10. 在 **Jaeger Details** 页面上，点击 **Resources** 选项卡。等到所有 Pod 的状态变为"Running"再继续操作。

#### 4.2.4.1. 通过 CLI 部署分布式追踪平台流策略

按照以下步骤，从命令行创建分布式追踪平台(Jaeger)实例。

**先决条件**

- 已安装 AMQ Streams Operator。如果使用 1.4.0 或更高版本，您可以使用自助置备。否则，您必须创建 Kafka 实例。
- 已安装 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。
- 您已查看了如何自定义部署的说明。
- 您可以访问与 OpenShift Container Platform 版本匹配的 OpenShift CLI(**oc**)。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 运行以下命令，以具有 **cluster-admin** 角色的用户身份登录到 OpenShift CLI (**oc**)：

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:8443
```

2. 运行以下命令，创建一个名为 **tracing-system** 的新项目：

```
$ oc new-project tracing-system
```

3. 创建一个名为 **jaeger-streaming.yaml** 的自定义资源文件，其中包含上一步中的示例文件文本。

4. 运行以下命令来部署 Jaeger：

```
$ oc create -n tracing-system -f jaeger-streaming.yaml
```

5. 在安装过程中运行以下命令来监控 pod 的进度：

```
$ oc get pods -n tracing-system -w
```

安装过程完成后，您应该看到类似以下示例的输出：

```
NAME                                                  READY   STATUS    RESTARTS   AGE
elasticsearch-cdm-jaegersystemjaegerstreaming-1-697b66d6fcztcnn   2/2     Running   0
5m40s
elasticsearch-cdm-jaegersystemjaegerstreaming-2-5f4b95c78b9gckz   2/2     Running   0
5m37s
elasticsearch-cdm-jaegersystemjaegerstreaming-3-7b6d964576nnz97   2/2     Running   0
5m5s
jaeger-streaming-collector-6f6db7f99f-rtcfm                       1/1     Running   0          80s
jaeger-streaming-entity-operator-6b6d67cc99-4lm9q                 3/3     Running   2
2m18s
jaeger-streaming-ingester-7d479847f8-5h8kc                        1/1     Running   0          80s
jaeger-streaming-kafka-0                                          2/2     Running   0          3m1s
jaeger-streaming-query-65bf5bb854-ncnc7                           3/3     Running   0          80s
jaeger-streaming-zookeeper-0                                      2/2     Running   0          3m39s
```

### 4.2.5. 验证部署

#### 4.2.5.1. 访问 Jaeger 控制台

要访问 Jaeger 控制台，您必须安装 Red Hat OpenShift Service Mesh 或 Red Hat OpenShift distributed tracing 平台，并安装、配置和部署 Red Hat OpenShift distributed tracing Platform (Jaeger)。

安装过程会创建路由来访问 Jaeger 控制台。

如果您知道 Jaeger 控制台的 URL，您可以直接访问它。如果您不知道 URL，请使用以下指示：

**从 Web 控制台的步骤**

1. 以具有 cluster-admin 权限的用户身份登录到 OpenShift Container Platform web 控制台。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 **dedicated-admin** 角色的帐户。

2. 进入 **Networking → Routes**。

3. 在 **Routes** 页面中，从 **Namespace** 菜单中选择 control plane 项目，如 **tracing-system**。**Location** 列显示每个路由的链接地址。

4. 如有必要，使用过滤器来查找 **jaeger** 路由。单击路由 **位置** 以启动控制台。

5. 单击 **Log In With OpenShift**。

**通过 CLI 操作的步骤**

1. 运行以下命令，以具有 **cluster-admin** 角色的用户身份登录 OpenShift Container Platform CLI。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 **dedicated-admin** 角色的帐户。

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. 要使用命令行查询路由详情，请输入以下命令。在本例中，**trace-system** 是 control plane 命名空间。

```
$ export JAEGER_URL=$(oc get route -n tracing-system jaeger -o jsonpath='{.spec.host}')
```

3. 启动浏览器并进入 **https://<JAEGER_URL>**，其中 **<JAEGER_URL>** 是您在上一步中发现的路由。

4. 使用您用于访问 OpenShift Container Platform 控制台的相同用户名和密码登录。

5. 如果您已将服务添加到服务网格中并生成了 trace，您可以使用过滤器和 **Find Traces** 按钮搜索 trace 数据。
   如果您要验证控制台安装，则不会显示 trace 数据。

## 4.2.6. 自定义部署

### 4.2.6.1. 部署最佳实践

- Red Hat OpenShift distributed tracing 平台实例名称必须是唯一的。如果您有多个 Red Hat OpenShift distributed tracing 平台 (Jaeger) 实例并使用 sidecar 注入的代理，则 Red Hat OpenShift distributed tracing 平台 (Jaeger) 实例应具有唯一的名称，注入注解应该明确指定追踪数据的名称。

- 如果您有多租户实现，且租户由命名空间分开，请将 Red Hat OpenShift distributed tracing Platform (Jaeger)实例部署到每个租户命名空间中。

有关配置持久性存储的详情，请参考 了解持久性存储 以及您选择的存储选项的适当配置主题。

### 4.2.6.2. 分布式追踪默认配置选项

Jaeger 自定义资源(CR)定义创建分布式追踪平台 (Jaeger) 资源时要使用的架构和设置。您可以修改这些参数以根据您的业务需求自定义分布式追踪平台 (Jaeger) 实施。

**Jaeger CR 的通用 YAML 示例**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: name
spec:
  strategy: <deployment_strategy>
  allInOne:
    options: {}
```

```
    resources: {}
  agent:
    options: {}
    resources: {}
  collector:
    options: {}
    resources: {}
  sampling:
    options: {}
  storage:
    type:
    options: {}
  query:
    options: {}
    resources: {}
  ingester:
    options: {}
    resources: {}
  options: {}
```

**表 4.1. Jaeger 参数**

| 参数 | 描述 | 值 | 默认值 |
|---|---|---|---|
| **apiVersion：** | 创建对象时要使用的 API 版本。 | **jaegertracing.io/v1** | **jaegertracing.io/v1** |
| **kind：** | 定义要创建的 Kubernetes 对象的种类。 | **jaeger** | |
| **metadata：** | 有助于唯一标识对象的数据，包括 **name** 字符串、**UID** 和可选 **namespace**。 | | OpenShift Container Platform 会自动生成 **UID** 并使用创建对象的项目名称完成 **namespace**。 |
| **name：** | 对象的名称。 | 分布式追踪平台(Jaeger)实例的名称。 | **jaeger-all-in-one-inmemory** |
| **spec：** | 要创建的对象的规格。 | 包含分布式追踪平台(Jaeger)实例的所有配置参数。当需要所有 Jaeger 组件的通用定义时，会在 **spec** 节点下定义它。当该定义与单个组件相关时，它将放置在 **spec/<component>** 节点下。 | N/A |
| **strategy：** | Jaeger 部署策略 | **allInOne**、**production** 或 **streaming** | **allInOne** |

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| **allInOne：** | 因为 **allInOne** 镜像在单个 pod 中部署了 Agent、Collector、Query、Ingester 和 Jaeger UI，所以此部署的配置必须在 **allInOne** 参数下嵌套组件配置。 | | |
| **agent：** | 定义代理的配置选项。 | | |
| **collector：** | 定义 Jaeger Collector 的配置选项。 | | |
| **sampling：** | 定义用于追踪的抽样策略的配置选项。 | | |
| **storage：** | 定义存储的配置选项。所有与存储相关的选项都必须放在 **storage** 下，而不是放在 **allInOne** 或其他组件选项下。 | | |
| **query：** | 定义 Query 服务的配置选项。 | | |
| **ingester：** | 定义 Ingester 服务的配置选项。 | | |

以下示例 YAML 是使用默认设置创建 Red Hat OpenShift distributed tracing Platform (Jaeger)部署所需的最小 YAML。

**最低要求示例 dist-tracing-all-in-one.yaml**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```

#### 4.2.6.3. Jaeger Collector 配置选项

Jaeger Collector 组件负责接收 tracer 捕获的 span，在使用 **production** 策略时将其写入持久性 Elasticsearch 存储，或使用 **streaming** 策略时将其写入 AMQ Streams。

Collector 是无状态的，因此很多 Jaeger Collector 实例可以并行运行。除了 Elasticsearch 集群的位置，收集器几乎不需要任何配置。

**表 4.2. Operator 用来定义 Jaeger Collector 的参数**

| 参数 | 描述 | 值 |
| --- | --- | --- |
| collector:<br>replicas: | 指定要创建的 Collector 副本数。 | 整数，如 **5**。 |

**表 4.3. 传递给 Collector 的配置参数**

| 参数 | 描述 | 值 |
| --- | --- | --- |
| spec:<br>collector:<br>options: {} | 定义 Jaeger Collector 的配置选项。 | |
| options:<br>collector:<br>num-workers: | 从队列中拉取的 worker 数量。 | 整数，如 **50**。 |
| options:<br>collector:<br>queue-size: | Collector 队列的大小。 | 整数，如 **2000**。 |
| options:<br>kafka:<br>producer:<br>topic: jaeger-spans | **topic** 参数标识收集器用来生成消息的 Kafka 配置以及要使用消息的 ingester。 | producer 的标签。 |
| options:<br>kafka:<br>producer:<br>brokers: my-cluster-kafka-brokers.kafka:9092 | 标识 Collector 用来生成消息的 Kafka 配置。如果没有指定代理，并且安装了 AMQ Streams 1.4.0+，Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 将自助置备 Kafka。 | |
| options:<br>log-level: | Collector 的日志记录级别。 | 可能的值有：**debug**、**info**、**warn**、**error**、**fatal**、**panic**。 |

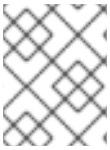
| 参数 | 描述 | 值 |
| --- | --- | --- |
| options:<br>  otlp:<br>  enabled: true<br>  grpc:<br>  host-port: 4317<br>  max-connection-age: 0s<br>  max-connection-age-grace: 0s<br>  max-message-size: 4194304<br>  tls:<br>  enabled: false<br>  cert: /path/to/cert.crt<br>  cipher-suites: "TLS_AES_256_GCM_SHA384,TLS_CHACHA20_POLY1305_SHA256"<br>  client-ca: /path/to/cert.ca<br>  reload-interval: 0s<br>  min-version: 1.2<br>  max-version: 1.3 | 要接受 OTLP/gRPC，请明确启用 **otlp**。所有其他选项都是可选的。 | |

| 参数 | 描述 | 值 |
| --- | --- | --- |
| options:<br>  otlp:<br>  enabled: true<br>  http:<br>  cors:<br>  allowed-headers: [<header-name>[, <header-name>]*]<br>  allowed-origins: *<br>  host-port: 4318<br>  max-connection-age: 0s<br>  max-connection-age-grace: 0s<br>  max-message-size: 4194304<br>  read-timeout: 0s<br>  read-header-timeout: 2s<br>  idle-timeout: 0s<br>  tls:<br>  enabled: false<br>  cert: /path/to/cert.crt<br>  cipher-suites: "TLS_AES_256_GCM_SHA384,TLS_CHACHA20_POLY1305_SHA256"<br>  client-ca: /path/to/cert.ca<br>  reload-interval: 0s<br>  min-version: 1.2<br>  max-version: 1.3 | 要接受 OTLP/HTTP，请明确启用 **otlp**。所有其他选项都是可选的。 | |

#### 4.2.6.4. 分布式追踪抽样配置选项

Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 可用于定义抽样策略，以提供给配置为使用远程 sampler 的 tracer。

虽然生成了所有 trace，但只有几个会被抽样。对某个 trace 进行抽样会标记该 trace 用于进一步处理和存储。



**注意**

如果一个 trace 由 Envoy 代理启动，则不会相关，因为抽样决定是在那里做出的。只有在应用程序使用客户端启动 trace 时，Jaeger 抽样决定才相关。

当服务收到不包含 trace 上下文的请求时，客户端会启动新的 trace，为它分配一个随机 trace ID，并根据当前安装的抽样策略做出抽样决定。抽样决定被传播到 trace 中的所有后续请求，以便其他服务不会再次做出抽样决定。

分布式追踪平台(Jaeger)库支持以下抽样：

- **Probabilistic（概率）** - sampler 做出一个随机抽样决定，其抽样的概率等于 **sampling.param** 属性的值。例如，使用 **sampling.param=0.1** 代表大约为 10 个 trace 抽样 1 次。
- **Rate Limiting（速率限制）** - sampler 使用泄漏存储桶速率限制器来确保 trace 使用某种恒定速率进行抽样。例如，使用 **sampling.param=2.0** 抽样请求，速率为每秒 2 个 trace。

**表 4.4. Jaeger 抽样选项**

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| spec:<br>sampling:<br>options: {}<br>default_strategy:<br><br>service_strategy: | 定义用于追踪的抽样策略的配置选项。 | | 如果没有提供配置，Collector 会返回默认的概率抽样策略，所有服务都为 0.001(0.1%)概率。 |
| default_strategy:<br>type:<br>service_strategy:<br>type: | 要使用的抽样策略。请参阅上述描述。 | 有效值是 **probabilistic** 和 **ratelimiting**。 | **probabilistic** |
| default_strategy:<br>param:<br>service_strategy:<br>param: | 所选抽样策略的参数。 | 小数值和整数值 (0, .1, 1, 10) | 1 |

这个示例定义了一种概率性的默认抽样策略，trace 实例被抽样的几率为 50%。

**概率抽样示例**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: with-sampling
spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 0.5
      service_strategies:
        - service: alpha
          type: probabilistic
          param: 0.8
          operation_strategies:
            - operation: op1
              type: probabilistic
              param: 0.2
```

```
        - operation: op2
          type: probabilistic
          param: 0.4
      - service: beta
        type: ratelimiting
        param: 5
```

如果没有用户提供的配置，分布式追踪平台(Jaeger)将使用以下设置：

**默认抽样**

```
spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 1
```

#### 4.2.6.5. 分布式追踪存储配置选项

您可以在 **spec.storage** 下为 Collector、Ingester 和 Query 服务配置存储。可以根据性能和恢复能力的需要提供每个组件的多个实例。

**表 4.5. Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 用来定义分布式追踪存储的一般存储参数**

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| `spec:`<br>`  storage:`<br>`  type:` | 要在部署中使用的存储类型。 | **memory** 或 **elasticsearch**。内存存储仅适用于开发、测试、演示和验证概念环境，因在关闭 pod 时，数据不会保留。对于生产环境，分布式追踪平台(Jaeger)支持 Elasticsearch 进行持久性存储。 | **内存** |
| `storage:`<br>`  secretname:` | secret 名称，例如 **tracing-secret**。 | | N/A |
| `storage:`<br>`  options: {}` | 定义存储的配置选项。 | | |

**表 4.6. Elasticsearch 索引清理参数**

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| storage:<br>esIndexCleaner:<br>enabled: | 当使用 Elasticsearch 存储时，默认会创建一个任务来清理索引中的旧 trace。这个参数用于启用或禁用索引清理任务。 | **true/ false** | **true** |
| storage:<br>esIndexCleaner:<br>numberOfDays: | 删除索引前等待的天数。 | 整数值 | **7** |
| storage:<br>esIndexCleaner:<br>schedule: | 为 Elasticsearch 索引的清理频率定义调度。 | Cron 表达式 | "55 23 * * *" |

#### 4.2.6.5.1. 自动置备 Elasticsearch 实例

当您部署 Jaeger 自定义资源时，Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 会使用 OpenShift Elasticsearch Operator 根据自定义资源文件的 **storage** 部分中提供的配置创建 Elasticsearch 集群。如果设置了以下配置，Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 将置备 Elasticsearch：

- **spec.storage:type** 设置为 **elasticsearch**
- **spec.storage.elasticsearch.doNotProvision** 设置为 **false**
- **spec.storage.options.es.server-urls** 没有被定义，即没有连接到 OpenShift Elasticsearch Operator 未置备的 Elasticsearch 实例。

在置备 Elasticsearch 时，Red Hat OpenShift distributed tracing platform (Jaeger) Operator 会将 Elasticsearch 自定义资源名称设置为 Jaeger 自定义资源的 **spec.storage.elasticsearch.name** 的 **name** 值。如果没有为 **spec.storage.elasticsearch.name** 指定一个值，Operator 会使用 **elasticsearch**。

**限制**

- 每个命名空间只能有一个带有自助置备 Elasticsearch 实例的分布式追踪平台(Jaeger)。Elasticsearch 集群旨在专用于单个分布式追踪平台(Jaeger)实例。
- 每个命名空间只能有一个 Elasticsearch。

**注意**

如果您已经安装了 Elasticsearch 作为 OpenShift Logging 的一部分，Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 可使用已安装的 OpenShift Elasticsearch Operator 来置备存储。

以下配置参数用于一个 *自置备的* Elasticsearch 实例，这是由 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 使用 OpenShift Elasticsearch Operator 创建的实例。在配置文件中，您可以在 **spec:storage:elasticsearch** 下为自助置备 Elasticsearch 指定配置选项。

**表 4.7. Elasticsearch 资源配置参数**

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| elasticsearch:<br>properties:<br>doNotProvision: | 用于指定 Red Hat OpenShift distributed tracing platform (Jaeger) Operator 是否应该置备 Elasticsearch 实例。 | **true**/**false** | **true** |
| elasticsearch:<br>properties:<br>name: | Elasticsearch 实例的名称。Red Hat OpenShift distributed tracing platform (Jaeger) Operator 使用此参数中指定的 Elasticsearch 实例连接到 Elasticsearch。 | string | **elasticsearch** |
| elasticsearch:<br>nodeCount: | Elasticsearch 节点数量。对于高可用性，需要至少 3 个节点。不要只使用 2 个节点，因为可能会出现"脑裂"问题。 | 整数值。例如，概念验证 = 1，最小部署 = 3 | 3 |
| elasticsearch:<br>resources:<br>requests:<br>cpu: | 根据您的环境配置，请求的 CPU 数量。 | 以内核数或 millicores 指定，例如 200m, 0.5, 1。例如，概念证明 = 500m，最小部署 =1 | 1 |
| elasticsearch:<br>resources:<br>requests:<br>memory: | 根据您的环境配置，可用于请求的内存。 | 以字节为单位指定，例如 200Ki, 50Mi, 5Gi。例如，概念证明 = 1Gi，最小部署 = 16Gi* | 16Gi |
| elasticsearch:<br>resources:<br>limits:<br>cpu: | 根据您的环境配置，CPU 数量的限值。 | 以内核数或 millicores 指定，例如 200m, 0.5, 1。例如，概念证明 = 500m，最小部署 =1 | |

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| elasticsearch:<br>resources:<br>limits:<br>memory: | 根据您的环境配置，可用的内存限值。 | 以字节为单位指定，例如 200Ki, 50Mi, 5Gi。例如，概念证明 = 1Gi，最小部署 = 16Gi* | |
| elasticsearch:<br><br>redundancyPolicy: | 数据复制策略定义如何在集群中的数据节点之间复制 Elasticsearch 分片：如果没有指定，Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 会自动根据节点数量决定最合适的复制。 | **ZeroRedundancy**（无副本分片）、**SingleRedundancy**（一个副本分片）、**MultipleRedundancy**（每个索引分散于一半的 Data 节点）、**FullRedundancy**（每个索引在集群中的每个 Data 节点上完全复制）。 | |
| elasticsearch:<br><br>useCertManagement: | 用于指定分布式追踪平台 (Jaeger)是否应使用 Elasticsearch Operator 的证书管理功能。此功能添加到 OpenShift Container Platform 4.7 中的 {logging-title} 5.2 中，它是新 Jaeger 部署的首选设置。 | **true**/**false** | **true** |

通过这个设置可以使每个 Elasticsearch 节点使用较低内存进行操作，但对于生产环境部署，不建议这样做。对于生产环境，您应该默认为每个 pod 分配不少于 16Gi 内存，但最好为每个 pod 最多分配 64Gi 内存。

**生产环境存储示例**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 3
      resources:
        requests:
          cpu: 1
          memory: 16Gi
        limits:
          memory: 16Gi
```

**具有持久性存储的存储示例：**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 1
      storage: 1
        storageClassName: gp2
        size: 5Gi
      resources:
        requests:
          cpu: 200m
          memory: 4Gi
        limits:
          memory: 4Gi
      redundancyPolicy: ZeroRedundancy
```

1 持久性存储配置。在本例中，AWS **gp2** 的大小为 **5Gi**。如果没有指定值，则分布式追踪平台(Jaeger)将使用 **emptyDir**。OpenShift Elasticsearch Operator 置备 **PersistentVolumeClaim** 和 **PersistentVolume**，它们不会在分布式追踪平台(Jaeger)实例中删除。如果创建具有相同名称和命名空间的分布式追踪平台(Jaeger)实例，则可以挂载同一卷。

#### 4.2.6.5.2. 连接到现有 Elasticsearch 实例

您可以使用现有 Elasticsearch 集群进行分布式追踪平台存储。现有的 Elasticsearch 集群（也称为 *外部* Elasticsearch 实例）是由 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 或 Elasticsearch Operator 安装的实例。

当您部署 Jaeger 自定义资源时，如果设置了以下配置，Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 将不会置备 Elasticsearch：

- **spec.storage.elasticsearch.doNotProvision** 设置为 **true**
- **spec.storage.options.es.server-urls** 有一个值
- **spec.storage.elasticsearch.name** 具有一个值，或者 Elasticsearch 实例名称是 **elasticsearch**。

Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 使用 **spec.storage.elasticsearch.name** 中指定的 Elasticsearch 实例连接到 Elasticsearch。

**限制**

- 您无法将 OpenShift Container Platform 日志记录 Elasticsearch 实例与分布式追踪平台(Jaeger)共享或重复使用。Elasticsearch 集群旨在专用于单个分布式追踪平台(Jaeger)实例。

以下配置参数适用于已经存在的 Elasticsearch 实例，也称为 *外部* Elasticsearch 实例。在本例中，您可以在自定义资源文件中的 **spec:storage:options:es** 下为 Elasticsearch 指定配置选项。

**表 4.8. 常规 ES 配置参数**

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| es:<br>server-urls: | Elasticsearch 实例的 URL。 | Elasticsearch 服务器的完全限定域名。 | **http://elasticsearch.<namespace>.svc:9200** |
| es:<br>max-doc-count: | 从 Elasticsearch 查询返回的最大文档数量。这也适用于聚合。如果同时设置了 **es.max-doc-count** 和 **es.max-num-spans**，Elasticsearch 将使用两者中的较小的值。 | | 10000 |
| es:<br>max-num-spans: | [**已弃用** - 将在以后的版本中删除，使用 **es.max-doc-count** 代替。] 在 Elasticsearch 中每个查询每次抓取的最大 span 数量。如果同时设置了 **es.max-num-spans** 和 **es.max-doc-count**，Elasticsearch 将使用两者中的较小的值。 | | 10000 |
| es:<br>max-span-age: | Elasticsearch 中 span 的最大查询。 | | 72h0m0s |
| es:<br>sniffer: | Elasticsearch 的侦察器配置。客户端使用侦察过程自动查找所有节点。默认禁用此选项。 | **true**/ **false** | **false** |
| es:<br>sniffer-tls-enabled: | 在监控 Elasticsearch 集群时启用 TLS 的选项。客户端使用侦察过程自动查找所有节点。默认禁用 | **true**/ **false** | **false** |
| es:<br>timeout: | 用于查询的超时。当设为零时，则没有超时。 | | 0s |

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| es:<br>username: | Elasticsearch 所需的用户名。如果指定，基本身份验证也会加载 CA。另请参阅 **es.password**。 | | |
| es:<br>password: | Elasticsearch 所需的密码。另请参阅 **es.username**。 | | |
| es:<br>version: | 主要的 Elasticsearch 版本。如果没有指定，则该值将从 Elasticsearch 中自动探测到。 | | 0 |

**表 4.9. ES 数据复制参数**

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| es:<br>num-replicas: | Elasticsearch 中每个索引的副本数。 | | 1 |
| es:<br>num-shards: | Elasticsearch 中每个索引的分片数量。 | | 5 |

**表 4.10. ES 索引配置参数**

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| es:<br>create-index-templates: | 设置为 **true** 时，应用程序启动时自动创建索引模板。手动安装模板时，设置为 **false**。 | **true**/ **false** | **true** |
| es:<br>index-prefix: | 分布式追踪平台(Jaeger)索引的可选前缀。例如，将其设置为 "production" 会创建名为 "production-tracing-*" 的索引。 | | |

**表 4.11. ES 批量处理器配置参数**

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| es:<br>bulk:<br>actions: | 在批量处理器决定向磁盘提交更新前可添加到队列的请求数。 | | 1000 |
| es:<br>bulk:<br>flush-interval: | **提交批量请求的时间**. 要禁用批量处理器清除间隔，请将其设置为零。 | | 200ms |
| es:<br>bulk:<br>size: | 在批量处理器决定提交更新之前，批量请求可以处理的字节数。 | | 5000000 |
| es:<br>bulk:<br>workers: | 可以接收并将批量请求提交 Elasticsearch 的 worker 数量。 | | 1 |

**表 4.12. ES TLS 配置参数**

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| es:<br>tls:<br>ca: | 用于验证远程服务器的 TLS 证书颁发机构(CA)文件的路径。 | | 默认将使用系统信任存储。 |
| es:<br>tls:<br>cert: | TLS 证书文件的路径，用来识别此进程到远程服务器。 | | |
| es:<br>tls:<br>enabled: | 与远程服务器对话时启用传输层安全(TLS)。默认禁用此选项。 | **true/ false** | **false** |
| es:<br>tls:<br>key: | TLS 私钥文件的路径，用来识别此进程到远程服务器。 | | |

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| es:<br>tls:<br>server-name: | 覆盖远程服务器证书中预期的 TLS 服务器名称。 | | |
| es:<br>token-file: | 包含 bearer 令牌的文件的路径。如果指定该标志，该标志也会载入认证机构（CA）文件。 | | |

**表 4.13. ES 归档配置参数**

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| es-archive:<br>bulk:<br>actions: | 在批量处理器决定向磁盘提交更新前可添加到队列的请求数。 | | 0 |
| es-archive:<br>bulk:<br>flush-interval: | **提交批量请求的时间**. 要禁用批量处理器清除间隔，请将其设置为零。 | | 0s |
| es-archive:<br>bulk:<br>size: | 在批量处理器决定提交更新之前，批量请求可以处理的字节数。 | | 0 |
| es-archive:<br>bulk:<br>workers: | 可以接收并将批量请求提交 Elasticsearch 的 worker 数量。 | | 0 |
| es-archive:<br>create-index-templates: | 设置为 **true** 时，应用程序启动时自动创建索引模板。手动安装模板时，设置为 **false**。 | **true**/ **false** | **false** |
| es-archive:<br>enabled: | 启用额外的存储。 | **true**/ **false** | **false** |

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| es-archive: index-prefix: | 分布式追踪平台(Jaeger)索引的可选前缀。例如，将其设置为 "production" 会创建名为 "production-tracing-*" 的索引。 | | |
| es-archive: max-doc-count: | 从 Elasticsearch 查询返回的最大文档数量。这也适用于聚合。 | | 0 |
| es-archive: max-num-spans: | [**已弃用** - 将在以后的版本中删除，使用 **es-archive.max-doc-count** 替代。] Elasticsearch 中的每个查询一次获取的最大 span 数量。 | | 0 |
| es-archive: max-span-age: | Elasticsearch 中 span 的最大查询。 | | 0s |
| es-archive: num-replicas: | Elasticsearch 中每个索引的副本数。 | | 0 |
| es-archive: num-shards: | Elasticsearch 中每个索引的分片数量。 | | 0 |
| es-archive: password: | Elasticsearch 所需的密码。另请参阅 **es.username**。 | | |
| es-archive: server-urls: | 以逗号分隔的 Elasticsearch 服务器列表。必须指定为完全限定的 URL，例如 **http://localhost:9200**。 | | |
| es-archive: sniffer: | Elasticsearch 的侦察器配置。客户端使用侦察过程自动查找所有节点。默认禁用此选项。 | **true**/ **false** | **false** |

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| es-archive:<br>sniffer-tls-<br>enabled: | 在监控 Elasticsearch 集群时启用 TLS 的选项。客户端使用侦察过程自动查找所有节点。默认禁用此选项。 | **true/ false** | **false** |
| es-archive:<br>timeout: | 用于查询的超时。当设为零时，则没有超时。 | | 0s |
| es-archive:<br>tls:<br>ca: | 用于验证远程服务器的 TLS 证书颁发机构(CA)文件的路径。 | | 默认将使用系统信任存储。 |
| es-archive:<br>tls:<br>cert: | TLS 证书文件的路径，用来识别此进程到远程服务器。 | | |
| es-archive:<br>tls:<br>enabled: | 与远程服务器对话时启用传输层安全(TLS)。默认禁用此选项。 | **true/ false** | **false** |
| es-archive:<br>tls:<br>key: | TLS 私钥文件的路径，用来识别此进程到远程服务器。 | | |
| es-archive:<br>tls:<br>server-name: | 覆盖远程服务器证书中预期的 TLS 服务器名称。 | | |
| es-archive:<br>token-file: | 包含 bearer 令牌的文件的路径。如果指定该标志，该标志也会载入认证机构（CA）文件。 | | |
| es-archive:<br>username: | Elasticsearch 所需的用户名。如果指定，基本身份验证也会加载 CA。请参阅 **es-archive.password**。 | | |

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| es-archive:<br>version: | 主要的 Elasticsearch 版本。如果没有指定，则该值将从 Elasticsearch 中自动探测到。 | | 0 |

**使用卷挂载的存储示例**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200
        index-prefix: my-prefix
        tls:
          ca: /es/certificates/ca.crt
    secretName: tracing-secret
  volumeMounts:
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public
```

以下示例显示了使用从存储在 secret 中的卷和用户/密码挂载了 TLS CA 证书的外部 Elasticsearch 集群的 Jaeger CR。

**外部 Elasticsearch 示例**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200 (1)
        index-prefix: my-prefix
        tls: (2)
          ca: /es/certificates/ca.crt
    secretName: tracing-secret (3)
```

```
volumeMounts: 4
  - name: certificates
    mountPath: /es/certificates/
    readOnly: true
volumes:
  - name: certificates
    secret:
      secretName: quickstart-es-http-certs-public
```

1 在默认命名空间中运行的 Elasticsearch 服务 URL。

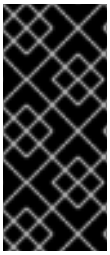
2 TLS 配置。在这种情况下，只有 CA 证书，但在使用 mutual TLS 时，它也可以包含 es.tls.key 和 es.tls.cert。

3 定义环境变量 ES_PASSWORD 和 ES_USERNAME 的 Secret。由 kubectl create secret generic tracing-secret --from-literal=ES_PASSWORD=changeme --from-literal=ES_USERNAME=elastic 创建

4 被挂载到所有存储组件的卷挂载和卷。

#### 4.2.6.6. 使用 Elasticsearch 管理证书

您可以使用 OpenShift Elasticsearch Operator 创建和管理证书。使用 OpenShift Elasticsearch Operator 管理证书还可让您使用多个 Jaeger Collector 的单一 Elasticsearch 集群。



**重要**

使用 Elasticsearch 管理证书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。

从版本 2.4 开始，Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 使用 Elasticsearch 自定义资源中的以下注解将证书创建委派给 OpenShift Elasticsearch Operator：

- **logging.openshift.io/elasticsearch-cert-management: "true"**
- **logging.openshift.io/elasticsearch-cert.jaeger-<shared-es-node-name>: "user.jaeger"**
- **logging.openshift.io/elasticsearch-cert.curator-<shared-es-node-name>: "system.logging.curator"**

其中 **<shared-es-node-name>** 是 Elasticsearch 节点的名称。例如，如果您创建一个名为 **custom-es** 的 Elasticsearch 节点，您的自定义资源可能类似以下示例。

**显示注解的 Elasticsearch CR 示例**

```
apiVersion: logging.openshift.io/v1
kind: Elasticsearch
metadata:
  annotations:
    logging.openshift.io/elasticsearch-cert-management: "true"
    logging.openshift.io/elasticsearch-cert.jaeger-custom-es: "user.jaeger"
```

```
    logging.openshift.io/elasticsearch-cert.curator-custom-es: "system.logging.curator"
  name: custom-es
spec:
  managementState: Managed
  nodeSpec:
    resources:
      limits:
        memory: 16Gi
      requests:
        cpu: 1
        memory: 16Gi
  nodes:
    - nodeCount: 3
      proxyResources: {}
      resources: {}
      roles:
        - master
        - client
        - data
      storage: {}
  redundancyPolicy: ZeroRedundancy
```

**先决条件**

- 安装了 Red Hat OpenShift Service Mesh Operator。
- {logging-title} 安装在集群中的默认配置。
- Elasticsearch 节点和 Jaeger 实例必须部署到同一命名空间中。例如，**tracing-system**。

您可以通过在 Jaeger 自定义资源中将 **spec.storage.elasticsearch.useCertManagement** 设置为 **true** 来启用证书管理。

**示例显示 useCertManagement**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      name: custom-es
      doNotProvision: true
      useCertManagement: true
```

在置备 Elasticsearch 时，Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 将 Elasticsearch 自定义资源的 **name** 设置为 Jaeger 自定义资源的 **spec.storage.elasticsearch.name** 的值。

证书由 OpenShift Elasticsearch Operator 和 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 注入证书。

#### 4.2.6.7. 查询**配置**选项

Query 是一个从存储中检索 trace 并托管用户界面来显示它们的服务。

**表 4.14. Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 用来定义 Query 的参数**

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| spec:<br>query:<br>replicas: | 指定要创建的 Query 副本数。 | 整数，如 **2**。 | |

**表 4.15. 传递给 Query 的配置参数**

| 参数 | 描述 | 值 | 默认值 |
| --- | --- | --- | --- |
| spec:<br>query:<br>options: {} | 定义 Query 服务的配置选项。 | | |
| options:<br>log-level: | Query 的日志记录级别。 | 可能的值有：**debug**、**info**、**warn**、**error**、**fatal**、**panic**。 | |
| options:<br>query:<br>base-path: | 所有 jaeger-query HTTP 路由的基本路径都可设置为非 root 值，例如，**/jaeger** 将导致所有 UI URL 以 **/jaeger** 开头。当在反向代理后面运行 Jaeger-query 时，这很有用。 | /<path> | |

**示例 Query 配置**

```
apiVersion: jaegertracing.io/v1
kind: "Jaeger"
metadata:
  name: "my-jaeger"
spec:
  strategy: allInOne
  allInOne:
    options:
      log-level: debug
      query:
        base-path: /jaeger
```

#### 4.2.6.8. Ingester **配置**选项

Ingester 是一个从 Kafka 主题读取并写入 Elasticsearch 存储后端的服务。如果您使用 **allInOne** 或 **production** 部署策略，则不需要配置 Ingester 服务。

**表 4.16. 传递给 Ingester 的 Jaeger 参数**

| 参数 | 描述 | 值 |
| --- | --- | --- |
| `spec:`<br>`  ingester:`<br>`  options: {}` | 定义 Ingester 服务的配置选项。 | |
| `options:`<br>`  deadlockInterval:` | 指定 Ingester 在终止前必须等待消息的时间间隔（以秒为单位）。默认情况下，死锁时间间隔被禁用（设置为 **0**），以避免在系统初始化过程中没有信息到达 Ingester。 | 分钟和秒，例如 **1m0s**。默认值为 **0**。 |
| `options:`<br>`  kafka:`<br>`  consumer:`<br>`  topic:` | **topic** 参数标识收集器用来生成消息的 Kafka 配置，并使用 Ingester。 | consumer 的标签例如，**jaeger-spans**. |
| `options:`<br>`  kafka:`<br>`  consumer:`<br>`  brokers:` | Ingester 用来使用消息的 Kafka 配置的标识。 | 代理的标签，如 **my-cluster-kafka-brokers.kafka:9092**。 |
| `options:`<br>`  log-level:` | Ingester 的日志记录级别。 | 可能的值有：**debug**,**info**,**warn**,**error**,**fatal**,**dpanic**,**panic**. |

**流传输 Collector 和 Ingester 示例**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-streaming
spec:
  strategy: streaming
  collector:
    options:
      kafka:
        producer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
  ingester:
    options:
      kafka:
```

```
      consumer:
        topic: jaeger-spans
        brokers: my-cluster-kafka-brokers.kafka:9092
    ingester:
      deadlockInterval: 5
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: http://elasticsearch:9200
```

### 4.2.7. 注入 sidecar

Red Hat OpenShift distributed tracing Platform (Jaeger)依赖于应用程序 pod 中的代理 sidecar 来提供代理。Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 可以将 Agent sidecar 注入部署工作负载。您可以使用自动 sidecar 注入功能或手动管理它。

#### 4.2.7.1. 自动注入 sidecar

Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 可以将 Jaeger Agent sidecar 注入部署工作负载。要启用 sidecar 自动注入，将 **sidecar.jaegertracing.io/inject** 注解设置为字符串 **true** 或通过运行 **$ oc get jaegers** 返回的分布式追踪平台(Jaeger)实例名称。当您指定 **true** 时，与部署相同的命名空间必须只有一个分布式追踪平台(Jaeger)实例。否则，Operator 无法决定使用哪个分布式追踪平台(Jaeger)实例。部署中的特定分布式追踪平台(Jaeger)实例名称的优先级高于其命名空间中应用的 **true**。

以下片段演示了一个简单的应用程序，它将注入一个 sidecar，代理指向同一命名空间中可用的单个分布式追踪平台(Jaeger)实例：

**自动 sidecar 注入示例**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    "sidecar.jaegertracing.io/inject": "true" 1
spec:
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: acme/myapp:myversion
```

1 设置为字符串 **true** 或 Jaeger 实例名称。

当 sidecar 被注入后，可以在 **localhost** 上的默认位置访问代理。

#### 4.2.7.2. 手动注入 sidecar

Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 只能将 Jaeger Agent sidecar 注入 Deployment 工作负载。对于 **Deployment** 以外的控制器类型，如 **StatefulSets`and `DaemonSets**，您可以在规格中手动定义 Jaeger 代理 sidecar。

以下片段显示了您可以在 Jaeger Agent sidecar 的 containers 部分中包含的手动定义：

**StatefulSet 的 sidecar 定义示例**

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: example-statefulset
  namespace: example-ns
  labels:
    app: example-app
spec:

    spec:
      containers:
        - name: example-app
          image: acme/myapp:myversion
          ports:
            - containerPort: 8080
              protocol: TCP
        - name: jaeger-agent
          image: registry.redhat.io/distributed-tracing/jaeger-agent-rhel7:<version>
           # The agent version must match the Operator version
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 5775
              name: zk-compact-trft
              protocol: UDP
            - containerPort: 5778
              name: config-rest
              protocol: TCP
            - containerPort: 6831
              name: jg-compact-trft
              protocol: UDP
            - containerPort: 6832
              name: jg-binary-trft
              protocol: UDP
            - containerPort: 14271
              name: admin-http
              protocol: TCP
          args:
            - --reporter.grpc.host-port=dns:///jaeger-collector-headless.example-ns:14250
            - --reporter.type=grpc
```

然后，代理可以在 localhost 上的默认位置访问。

## 4.3. 升级

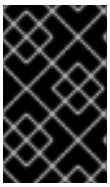
**重要**

Red Hat OpenShift distributed tracing Platform (Jaeger) 是一个已弃用的功能。弃用的功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。

有关 OpenShift Container Platform 中已弃用或删除的主要功能的最新列表，请参阅 OpenShift Container Platform 发行注记中*已弃用和删除的功能*部分。

Operator Lifecycle Manager (OLM) 能够控制集群中 Operator 的安装、升级和基于角色的访问控制 (RBAC)。OLM 在 OpenShift Container Platform 中默认运行。OLM 会查询可用的 Operator 以及已安装的 Operator 的升级。

在更新过程中，Red Hat OpenShift distributed tracing Platform Operator 将受管分布式追踪平台实例升级到与 Operator 关联的版本。每当安装新版本的 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 时，由 Operator 管理的所有分布式追踪平台(Jaeger)应用程序实例都会升级到 Operator 的版本。例如，在将 Operator 从 1.10 升级到 1.11 后，Operator 扫描运行分布式追踪平台 (Jaeger)实例并将其升级到 1.11。



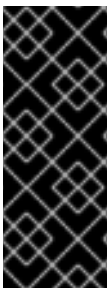
**重要**

如果您还没有更新 OpenShift Elasticsearch Operator，如 更新 OpenShift Logging 所述，请在更新 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 前完成该更新。

### 4.3.1. 其他资源

- Operator Lifecycle Manager 概念和资源
- 更新安装的 Operator
- 更新 OpenShift Logging

## 4.4. 删除



**重要**

Red Hat OpenShift distributed tracing Platform (Jaeger) 是一个已弃用的功能。弃用的功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。

有关 OpenShift Container Platform 中已弃用或删除的主要功能的最新列表，请参阅 OpenShift Container Platform 发行注记中*已弃用和删除的功能*部分。

从 OpenShift Container Platform 集群中删除 Red Hat OpenShift distributed tracing 平台的步骤如下：

1. 关闭任何 Red Hat OpenShift distributed tracing 平台 pod。
2. 删除任何 Red Hat OpenShift distributed tracing 平台实例。
3. 删除 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。
4. 删除 OpenTelemetry Operator 的红帽构建。

### 4.4.1. 使用 Web 控制台删除分布式追踪平台(Jaeger)实例

您可以在 web 控制台的 **Administrator** 视图中删除分布式追踪平台(Jaeger)实例。

**警告**

当删除使用内存存储的实例时，所有数据都会不可避免地丢失。当 Red Hat OpenShift distributed tracing Platform (Jaeger)实例被删除时，存储在持久性存储中的数据不会被删除。

**先决条件**

- 以集群管理员身份使用 **cluster-admin** 角色登录到 web 控制台。

**流程**

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 导航到 **Operators → Installed Operators**。
3. 从 **Project** 菜单中选择 Operators 安装的项目名称，如 **openshift-operators**。
4. 点 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。
5. 点 **Jaeger** 标签页。
6. 点击您要删除  的实例旁的 Options 菜单，然后选择 **Delete Jaeger**。
7. 在确认信息中，点击 **Delete**。

### 4.4.2. 使用 CLI 删除分布式追踪平台(Jaeger)实例

您可以在命令行中删除分布式追踪平台(Jaeger)实例。

**先决条件**

- 集群管理员具有 **cluster-admin** 角色的活跃 OpenShift CLI (**oc**) 会话。

  **提示**

  - 确保您的 OpenShift CLI (**oc**) 版本为最新版本，并与您的 OpenShift Container Platform 版本匹配。
  - 运行 **oc login**:

    ```
    $ oc login --username=<your_username>
    ```

**流程**

1. 运行以下命令，使用 OpenShift CLI (**oc**) 登录：

```
$ oc login --username=<NAMEOFUSER>
```

2. 要显示分布式追踪平台(Jaeger)实例，请运行以下命令：

```
$ oc get deployments -n <jaeger-project>
```

例如，

```
$ oc get deployments -n openshift-operators
```

Operator 的名称具有后缀 **-operator**。以下示例显示了两个 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 和四个分布式追踪平台(Jaeger)实例：

```
$ oc get deployments -n openshift-operators
```

您将看到类似如下的输出：

```
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
elasticsearch-operator   1/1     1            1           93m
jaeger-operator          1/1     1            1           49m
jaeger-test              1/1     1            1           7m23s
jaeger-test2             1/1     1            1           6m48s
tracing1                 1/1     1            1           7m8s
tracing2                 1/1     1            1           35m
```

3. 要删除分布式追踪平台(Jaeger)的实例，请运行以下命令：

```
$ oc delete jaeger <deployment-name> -n <jaeger-project>
```

例如：

```
$ oc delete jaeger tracing2 -n openshift-operators
```

4. 要验证删除过程，请再次运行 **oc get deployments** 命令：

```
$ oc get deployments -n <jaeger-project>
```

例如：

```
$ oc get deployments -n openshift-operators
```

您将看到生成的输出类似以下示例：

```
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
elasticsearch-operator   1/1     1            1           94m
jaeger-operator          1/1     1            1           50m
jaeger-test              1/1     1            1           8m14s
jaeger-test2             1/1     1            1           7m39s
tracing1                 1/1     1            1           7m59s
```

### 4.4.3. 删除 Red Hat OpenShift distributed tracing Platform Operator

**流程**

1. 按照 从集群中删除 Operator 中的说明操作， 以删除 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator。
2. 可选： 删除 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 后，删除 OpenShift Elasticsearch Operator。