OpenShift Container Platform 4.19

分布式追踪

在 OpenShift Container Platform 中配置和使用分布式追踪

Last Updated: 2025-12-11

OpenShift Container Platform 4.19 分布式追踪

在 OpenShift Container Platform 中配置和使用分布式追踪

Legal Notice

Copyright © 2025 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

使用分布式追踪存储、分析和视觉化微服务事务，通过 OpenShift Container Platform 中的分布式系统传递。

Table of Contents

第 1 章 RED HAT OPENSHIFT DISTRIBUTED TRACING PLATFORM 3.8 发行注记 3
1.1. 关于此版本 3
1.2. 修复的问题 3
1.3. 获取支持 3

第 2 章 关于 DISTRIBUTED TRACING PLATFORM 5
2.1. 分布式追踪中的关键概念 5
2.2. RED HAT OPENSHIFT DISTRIBUTED TRACING PLATFORM 功能 5
2.3. RED HAT OPENSHIFT DISTRIBUTED TRACING PLATFORM 架构 6

第 3 章 安装 DISTRIBUTED TRACING PLATFORM 7
3.1. 安装 TEMPO OPERATOR 7
3.2. 对象存储设置 9
3.3. 配置权限和租户. 22
3.4. 安装 TEMPOSTACK 实例 27
3.5. 安装 TEMPOMONOLITHIC 实例 33
3.6. 其他资源 40

第 4 章 配置 DISTRIBUTED TRACING PLATFORM 41
4.1. 配置后端存储 41
4.2. TEMPOSTACK 配置参数简介 41
4.3. 查询配置选项 44
4.4. 配置 UI 46
4.5. 在 JAEGER UI 中配置 MONITOR 选项卡 46
4.6. 配置接收器 TLS 50
4.7. 配置查询 RBAC 52
4.8. 使用污点和容限 54
4.9. 配置监控和警报 54

第 5 章 DISTRIBUTED TRACING PLATFORM 故障排除 56
5.1. 从命令行收集诊断数据 56

第 6 章 升级 57
6.1. 其他资源 57

第 7 章 删除 DISTRIBUTED TRACING PLATFORM 58
7.1. 使用 WEB 控制台删除 58
7.2. 使用 CLI 删除 58
7.3. 其他资源 59

第 1 章 RED HAT OPENSHIFT DISTRIBUTED TRACING PLATFORM 3.8 发行注记

1.1. 关于此版本

Distributed Tracing Platform 3.8 通过 Tempo Operator 0.19.0 提供，它基于开源 Grafana Tempo 2.9.0。

注意

只支持在文档中包括的功能。没有包括在文档中的功能不被支持。如果您需要对某个功能的帮助，请联系红帽支持。

1.2. 修复的问题

解决了影响 Tempo pod 的 TLS 证书的问题

在此次更新之前，Tempo pod 会因为内部 TLS 证书被续订而停止进行通信。在这个版本中，Tempo pod 会在证书被续订后自动重启。
TRACING-5622

Tempo 查询不再会在获取 trace JSON 时失败

在这个更新 之前，点 Jaeger UI 中的**Trace** 并刷新页，或者从 Tempo 查询前端访问 **Trace → Trace Timeline → Trace JSON**，可能会导致 Tempo 查询 pod 失败并显示 EOF 错误。在这个版本中，这个问题已解决。
TRACING-5483

注意

某些链接的 Jira 只能通过红帽凭证访问。

1.3. 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。

警告

弃用的 Red Hat OpenShift distributed Tracing Platform (Jaeger) 3.5 是红帽支持的 Red Hat OpenShift distributed Tracing Platform (Jaeger) 的最后一个发行版本。

所有对已弃用 Red Hat OpenShift distributed Tracing Platform (Jaeger) 3.5 的支持和维护在 2025 年 11 月 3 日终止。

如果您仍然使用 Red Hat OpenShift distributed Tracing Platform (Jaeger)，您必须迁移到红帽构建的 OpenTelemetry Operator 和 Tempo Operator 用于分布式追踪集合和存储。如需更多信息，请参阅 红帽构建的 OpenTelemetry 文档中的迁移、红帽构建的 OpenTelemetry 文档中的安装，以及 Red Hat OpenShift Distributed Tracing Platform 文档中的安装。

如需更多信息，请参阅 OpenShift 中的红帽知识库解决方案 Jaeger Deprecation 和 Removal。

第 2 章 关于 DISTRIBUTED TRACING PLATFORM

2.1. 分布式追踪中的关键概念

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 Platform 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。端到端的 trace 由一个或多个 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。

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 Distributed Tracing Platform ，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 Distributed Tracing Platform 与 OpenShift Container Platform 的其它相关组件相结合：

- 红帽构建的 OpenTelemetry 用于将 trace 转发到 TempoStack 实例
- Cluster Observability Operator (COO) 的 Distributed tracing UI 插件

其他资源

- Red Hat build of OpenTelemetry
- 分布式追踪 UI 插件

2.2. RED HAT OPENSHIFT DISTRIBUTED TRACING PLATFORM 功能

Red Hat OpenShift distributed Tracing Platform 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 Distributed Tracing Platform 数据。
- 高可伸缩性 - Distributed Tracing Platform 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。

2.3. 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**- 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。
  - **查询 Frontend** - Query Frontend 划分了传入的查询的搜索空间，并将查询发送到 Querier。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。

第 3 章 安装 DISTRIBUTED TRACING PLATFORM

安装 Distributed Tracing Platform 涉及以下步骤：

1. 安装 Tempo Operator。
2. 设置受支持的对象存储并为对象存储凭证创建 secret。
3. 配置权限和租户.
4. 根据您的用例，安装您选择的部署：
   - Microservices-mode **TempoStack** 实例
   - Monolithic-mode **TempoMonolithic** 实例

3.1. 安装 TEMPO OPERATOR

您可以使用 Web 控制台或命令行安装 Tempo Operator。

3.1.1. 使用 Web 控制台安装 Tempo Operator

您可从 OpenShift Container Platform Web 控制台安装 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 Distributed Tracing Platform 中。在安装 Distributed Tracing Platform (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. 在 Web 控制台中，搜索 **Tempo Operator**。

   提示

   在 OpenShift Container Platform 4.19 或更早版本中，进入 **Operators → OperatorHub**。

   在 OpenShift Container Platform 4.20 或更高版本中，进入 **Ecosystem → Software Catalog**。

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**。

3.1.2. 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 Distributed Tracing Platform 中。在安装 Distributed Tracing Platform (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
```

3.2. 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。

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。

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 **bucket: <bucket_name> # requires an ObjectBucketClaim** **endpoint: https://s3.openshift-storage.svc** **access_key_id: <data_foundation_access_key_id>** **access_key_secret: <data_foundation_access_key_secret>**
MinIO
请参阅 MinIO Operator。 **name: tempostack-dev-minio # example** **bucket: <minio_bucket_name> # MinIO documentation** **endpoint: <minio_bucket_endpoint>** **access_key_id: <minio_access_key_id>** **access_key_secret: <minio_access_key_secret>**
Amazon S3
name: tempostack-dev-s3 # example **bucket: <s3_bucket_name> # Amazon S3 documentation** **endpoint: <s3_bucket_endpoint>** **access_key_id: <s3_access_key_id>** **access_key_secret: <s3_access_key_secret>**
带有安全令牌服务(STS)的 Amazon S3

存储供应商
name: tempostack-dev-s3 # example **bucket: <s3_bucket_name> # Amazon S3 documentation** **region: <s3_region>** **role_arn: <s3_role_arn>**
Microsoft Azure Blob Storage
name: tempostack-dev-azure # example **container: <azure_blob_storage_container_name> # Microsoft Azure documentation** **account_name: <azure_blob_storage_account_name>** **account_key: <azure_blob_storage_account_key>**
Google Cloud Storage 上的 Google Cloud Storage
name: tempostack-dev-gcs # example **bucketname: <google_cloud_storage_bucket_name> # requires a bucket created in a Google Cloud project** **key.json: <path/to/key.json> # requires a service account in the bucket's GCP project for GCP authentication**

3.2.1. 使用安全令牌服务设置 Amazon S3 存储

您可以使用安全令牌服务 (STS) 和 AWS 命令行界面(AWS CLI) 设置 Amazon S3 存储。另外，您还可以使用 Cloud Credential Operator (CCO)。

重要

将 Distributed Tracing Platform 与 Amazon S3 存储和 STS 一起使用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，请参阅以下链接：

- 技术预览功能支持范围

先决条件

- 已安装 AWS CLI 的最新版本。
- 如果要使用 CCO，已在集群中安装并配置了 CCO。

流程

1. 创建 AWS S3 存储桶。

2. 为 AWS Identity and Access Management (AWS IAM) 策略创建以下 **trust.json** 文件，用于设置 AWS IAM 角色之间的信任关系，您要在下一步中创建的，以及 **TempoStack** 或 **TempoMonolithic** 实例的服务帐户：

   trust.json

```
{
   "Version": "2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Principal": {
         "Federated": "arn:aws:iam::<aws_account_id>:oidc-provider/<oidc_provider>" (1)
       },
       "Action": "sts:AssumeRoleWithWebIdentity",
       "Condition": {
         "StringEquals": {
           "<oidc_provider>:sub": [
             "system:serviceaccount:<openshift_project_for_tempo>:tempo-
<tempo_custom_resource_name>" (2)
             "system:serviceaccount:<openshift_project_for_tempo>:tempo-
<tempo_custom_resource_name>-query-frontend"
           ]
         }
       }
     }
   ]
}
```

   (1) 在 OpenShift Container Platform 上配置的 OpenID Connect (OIDC) 供应商。

   (2) 您要在其中创建 **TempoStack** 或 **TempoMonolithic** 实例的命名空间。将 **<tempo_custom_resource_name>** 替换为您在 **TempoStack** 或 **TempoMonolithic** 自定义资源中定义的 **metadata** 名称。

   提示

   您还可以运行以下命令来获取 OIDC 供应商的值：

```
$ oc get authentication cluster -o json | jq -r '.spec.serviceAccountIssuer' | sed
's~http[s]*://~~g'
```

3. 通过附加创建的 **trust.json** 策略文件来创建 AWS IAM 角色。您可以运行以下命令来完成此操作：

```
$ aws iam create-role \
    --role-name "tempo-s3-access" \
    --assume-role-policy-document "file:///tmp/trust.json" \
    --query Role.Arn \
```

```
    --output text
```

4. 将 AWS IAM 策略附加到创建的 AWS IAM 角色。您可以运行以下命令来完成此操作：

```
$ aws iam attach-role-policy \
    --role-name "tempo-s3-access" \
    --policy-arn "arn:aws:iam::aws:policy/AmazonS3FullAccess"
```

5. 如果不使用 CCO，请跳过这一步。如果使用 CCO，请为 Tempo Operator 配置云供应商环境。您可以运行以下命令来完成此操作：

```
$ oc patch subscription <tempo_operator_sub> \ (1)
        -n <tempo_operator_namespace> \ (2)
        --type='merge' -p '{"spec": {"config": {"env": [{"name": "ROLEARN", "value": """
<role_arn>"""}]}}}' (3)
```

(1) Tempo Operator 订阅的名称。

(2) Tempo Operator 的命名空间。

(3) AWS STS 需要将 **ROLEARN** 环境变量添加到 Tempo Operator 子系统中。作为 **<role_arn>** 值，添加在第 3 步中创建的 AWS IAM 角色的 Amazon Resource Name (ARN)。

6. 在 OpenShift Container Platform 中，使用键创建一个对象存储 secret，如下所示：

```
apiVersion: v1
kind: Secret
metadata:
  name: <secret_name>
stringData:
  bucket: <s3_bucket_name>
  region: <s3_region>
  role_arn: <s3_role_arn>
type: Opaque
```

7. 创建对象存储 secret 时，按如下所示 Distributed Tracing Platform 实例的相关自定义资源：

TempoStack 自定义资源示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
  namespace: <namespace>
spec:
# ...
  storage:
    secret: (1)
      name: <secret_name>
      type: s3
      credentialMode: token-cco (2)
# ...
```

–

1. 上一步中创建的 secret。
2. 如果不使用 CCO，请省略这一行。如果使用 CCO，请使用 **token-cco** 值添加此参数。

TempoMonolithic 自定义资源示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic
metadata:
  name: <name>
  namespace: <namespace>
spec:
# ...
  storage:
    traces:
      backend: s3
      s3:
        secret: <secret_name> (1)
        credentialMode: token-cco (2)
# ...
```

1. 上一步中创建的 secret。
2. 如果不使用 CCO，请省略这一行。如果使用 CCO，请使用 **token-cco** 值添加此参数。

其他资源

- AWS Identity and Access Management 文档 (AWS 文档)
- AWS 命令行界面文档 (AWS 文档)
- 配置 OpenID Connect 身份提供程序
- 使用 Amazon 资源名称(ARN) 识别 AWS 资源 (AWS 文档)

3.2.2. 使用安全令牌服务设置 Azure 存储

您可以使用 Azure 命令行界面 (Azure CLI) 使用安全令牌服务 (STS) 设置 Azure 存储。

重要

将 Distributed Tracing Platform 与 Azure 存储和 STS 一起使用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，请参阅以下链接：

- 技术预览功能支持范围

先决条件

- 已安装 Azure CLI 的最新版本。
- 您已创建了 Azure 存储帐户。
- 您已创建了 Azure blob 存储容器。

流程

1. 运行以下命令来创建 Azure 管理的身份：

```
$ az identity create \
  --name <identity_name> \ (1)
  --resource-group <resource_group> \ (2)
  --location <region> \ (3)
  --subscription <subscription_id> (4)
```

(1) 为受管身份选择的名称。

(2) 您要创建身份的 Azure 资源组。

(3) Azure 区域，它必须与资源组相同。

(4) Azure 订阅 ID。

2. 为 OpenShift Container Platform 服务帐户创建一个联邦身份凭证，供 Distributed Tracing Platform 的所有组件使用，但 Query Frontend 除外。您可以运行以下命令来完成此操作：

```
$ az identity federated-credential create \ (1)
  --name <credential_name> \ (2)
  --identity-name <identity_name> \
  --resource-group <resource_group> \
  --issuer <oidc_provider> \ (3)
  --subject <tempo_service_account_subject> \ (4)
  --audiences <audience> (5)
```

(1) 联邦身份凭证允许 OpenShift Container Platform 服务帐户作为 Azure 管理的身份进行身份验证，而无需存储 secret 或使用 Azure 服务主体身份。

(2) 为联邦凭证选择的名称。

(3) 集群的 OpenID Connect (OIDC) 供应商的 URL。

(4) 集群的服务帐户主题，格式为：**system:serviceaccount:<namespace>:tempo-<tempostack_instance_name>**。

(5) 预期的受众，用于验证为联邦身份凭证发布的令牌。这通常设置为 **api://AzureADTokenExchange**。

提示

您可以运行以下命令来获取集群的 OpenID Connect (OIDC) 签发者的 URL：

```
$ oc get authentication cluster -o json | jq -r .spec.serviceAccountIssuer
```

3. 为 OpenShift Container Platform 服务帐户创建一个联邦身份凭证，供 Distributed Tracing Platform 的 Query Frontend 组件使用。您可以运行以下命令来完成此操作：

```
$ az identity federated-credential create \ (1)
  --name <credential_name>-frontend \ (2)
  --identity-name <identity_name> \
  --resource-group <resource_group> \
  --issuer <cluster_issuer> \
  --subject <tempo_service_account_query_frontend_subject> \ (3)
  --audiences <audience> | jq
```

1. 联邦身份凭证允许 OpenShift Container Platform 服务帐户作为 Azure 管理的身份进行身份验证，而无需存储 secret 或使用 Azure 服务主体身份。
2. 为前端联邦的身份凭证选择的名称。
3. 集群的服务帐户主题，格式为：**system:serviceaccount:<namespace>:tempo-<tempostack_instance_name>**。

4. 将 Storage Blob Data Contributor 角色分配给创建的 Azure 管理身份的 Azure 服务主体身份。您可以运行以下命令来完成此操作：

```
$ az role assignment create \
  --assignee <assignee_name> \ (1)
  --role "Storage Blob Data Contributor" \
  --scope "/subscriptions/<subscription_id>
```

1. 您在第 1 步中创建的 Azure 管理身份的 Azure 服务主体身份。

提示

您可以运行以下命令来获取 **<assignee_name>** 值：

```
$ az ad sp list --all --filter "servicePrincipalType eq 'ManagedIdentity'" | jq -r --arg idName
<identity_name> '.[] | select(.displayName == $idName) | .appId'`
```

5. 获取您在第 1 步中创建的 Azure 管理身份的客户端 ID：

```
CLIENT_ID=$(az identity show \
  --name <identity_name> \ (1)
  --resource-group <resource_group> \ (2)
  --query clientId \
  -o tsv)
```

1. 复制并粘贴第 1 步中的 **<identity_name>** 值。
2. 复制并粘贴第 1 步中的 **<resource_group>** 值。

6. 为 Azure 工作负载身份联邦(WIF)创建 OpenShift Container Platform secret。您可以运行以下命令来完成此操作：

```
$ oc create -n <tempo_namespace> secret generic azure-secret \
  --from-literal=container=<azure_storage_azure_container> \ (1)
  --from-literal=account_name=<azure_storage_azure_accountname> \ (2)
  --from-literal=client_id=<client_id> \ (3)
  --from-literal=audience=<audience> \ (4)
  --from-literal=tenant_id=<tenant_id> (5)
```

1. Azure Blob Storage 容器的名称。
2. Azure Storage 帐户的名称。
3. 您在上一步中获取的受管身份的客户端 ID。
4. 可选：默认为 **api://AzureADTokenExchange**。
5. Azure 租户 ID。

7. 创建对象存储 secret 时，按如下所示 Distributed Tracing Platform 实例的相关自定义资源：

TempoStack 自定义资源示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
  namespace: <namespace>
spec:
# ...
  storage:
    secret: (1)
      name: <secret_name>
      type: azure
# ...
```

1. 上一步中创建的 secret。

TempoMonolithic 自定义资源示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic
metadata:
  name: <name>
  namespace: <namespace>
spec:
```

```
# ...
  storage:
    traces:
      backend: azure
      azure:
        secret: <secret_name> (1)
# ...
```

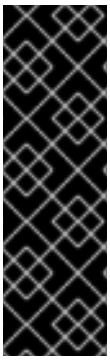
1. 上一步中创建的 secret。

其他资源

- 在 Linux 上安装 Azure CLI (Azure 文档)

3.2.3. 使用安全令牌服务设置 Google Cloud 存储

您可以使用 Google Cloud CLI 使用安全令牌服务 (STS) 设置 Google Cloud Storage (GCS)。



重要

使用带有 GCS 和 STS 的 Distributed Tracing Platform 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，请参阅以下链接：

- 技术预览功能支持范围

先决条件

- 已安装 Google Cloud CLI 的最新版本。

流程

1. 在 Google Cloud 上创建 GCS 存储桶。
2. 使用 Google Identity and Access Management (IAM) 创建或重复使用服务帐户：

```
SERVICE_ACCOUNT_EMAIL=$(gcloud iam service-accounts create
<iam_service_account_name> \ (1)
   --display-name="Tempo Account" \
   --project <project_id>  \ (2)
   --format='value(email)' \
   --quiet)
```

1. Google Cloud 上服务帐户的名称。
2. Google Cloud 上服务帐户的项目 ID。

3. 将所需的 Google Cloud 角色绑定到项目级别创建的服务帐户。您可以运行以下命令来完成此操作：

```
$ gcloud projects add-iam-policy-binding <project_id> \
    --member "serviceAccount:$SERVICE_ACCOUNT_EMAIL" \
    --role "roles/storage.objectAdmin"
```

4. 检索与集群关联的 Google Cloud Workload Identity Pool 的 **POOL_ID** 值。如何检索这个值取决于您的环境，因此命令只是示例：

```
$ OIDC_ISSUER=$(oc get authentication.config cluster -o
jsonpath='{.spec.serviceAccountIssuer}') \
&&
  POOL_ID=$(echo "$OIDC_ISSUER" | awk -F'/' '{print $NF}' | sed 's/-oidc$//')
```

5. 添加 IAM 策略绑定。您可以运行以下命令来完成此操作：

```
$ gcloud iam service-accounts add-iam-policy-binding "$SERVICE_ACCOUNT_EMAIL" \ (1)
  --role="roles/iam.workloadIdentityUser" \
  --
member="principal://iam.googleapis.com/projects/<project_number>/locations/global/workloadIc
entityPools/<pool_id>/subject/system:serviceaccount:<tempo_namespace>:tempo-
<tempo_name>" \
  --project=<project_id> \
  --quiet \
&&
  gcloud iam service-accounts add-iam-policy-binding "$SERVICE_ACCOUNT_EMAIL" \
  --role="roles/iam.workloadIdentityUser" \
  --
member="principal://iam.googleapis.com/projects/<project_number>/locations/global/workloadIc
entityPools/<pool_id>/subject/system:serviceaccount:<tempo_namespace>:tempo-
<tempo_name>-query-frontend" \
  --project=<project_id> \
  --quiet
&&
  gcloud storage buckets add-iam-policy-binding "gs://$BUCKET_NAME" \
  --role="roles/storage.admin" \
  --member="serviceAccount:$SERVICE_ACCOUNT_EMAIL" \
  --condition=None
```

(1) **$SERVICE_ACCOUNT_EMAIL** 是第 2 步中的命令的输出。

6. 为存储 secret 的 **key.json** 密钥创建一个凭据文件，供 **TempoStack** 自定义资源使用。您可以运行以下命令来完成此操作：

```
$ gcloud iam workload-identity-pools create-cred-config \

"projects/<project_number>/locations/global/workloadIdentityPools/<pool_id>/providers/<provide
r_id>" \
    --service-account="$SERVICE_ACCOUNT_EMAIL" \
    --credential-source-file=/var/run/secrets/storage/serviceaccount/token \ (1)
    --credential-source-type=text \
    --output-file=<output_file_path> (2)
```

1. **credential-source-file** 参数必须始终指向 **/var/run/secrets/storage/serviceaccount/token** 路径，因为 Operator 从这个路径挂载令牌。
2. 保存输出文件的路径。

7. 运行以下命令来获取正确的受众：

```
$ gcloud iam workload-identity-pools providers describe "$PROVIDER_NAME" --
format='value(oidc.allowedAudiences[0])'
```

8. 运行以下命令，为 Distributed Tracing Platform 创建存储 secret。

```
$ oc -n <tempo_namespace> create secret generic gcs-secret \
  --from-literal=bucketname="<bucket_name>" \ (1)
  --from-literal=audience="<audience>" \ (2)
  --from-file=key.json=<output_file_path> (3)
```

1. Google Cloud Storage 的存储桶名称。
2. 您在上一步获得的受众。
3. 您在第 6 步中创建的凭据文件。

9. 创建对象存储 secret 时，按如下所示 Distributed Tracing Platform 实例的相关自定义资源：

TempoStack 自定义资源示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: <name>
  namespace: <namespace>
spec:
# ...
  storage:
    secret: (1)
      name: <secret_name>
      type: gcs
# ...
```

1. 上一步中创建的 secret。

TempoMonolithic 自定义资源示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic
metadata:
  name: <name>
  namespace: <namespace>
spec:
```

```
# ...
  storage:
    traces:
      backend: gcs
      gcs:
        secret: <secret_name> (1)
# ...
```

(1) 上一步中创建的 secret。

其他资源

- 安装 gcloud CLI (Google Cloud 文档)
- 服务帐户概述 (Google Cloud 文档)

3.2.4. 设置 IBM Cloud Object Storage

您可以使用 OpenShift CLI (**oc**) 设置 IBM Cloud Object Storage。

先决条件

- 已安装最新版本的 OpenShift CLI (**oc**)。如需更多信息，请参阅*配置：CLI 工具*中的"使用 OpenShift CLI 入门"。
- 已安装最新版本的 IBM Cloud Command Line Interface (**ibmcloud**)。如需更多信息，请参阅 *IBM Cloud Docs* 中的"开始使用 IBM Cloud CLI"。
- 您已配置了 IBM Cloud Object Storage。如需更多信息，请参阅 *IBM Cloud Docs* 中的"删除计划并创建实例"。
  - 您有一个 IBM Cloud Platform 帐户。
  - 您已订购了一个 IBM Cloud Object Storage 计划。
  - 您已创建了 IBM Cloud Object Storage 实例。

流程

1. 在 IBM Cloud 上，创建一个对象存储存储桶。
2. 在 IBM Cloud 中，运行以下命令来创建用于连接到对象存储存储桶的服务密钥：

```
$ ibmcloud resource service-key-create <tempo_bucket> Writer \
  --instance-name <tempo_bucket> --parameters '{"HMAC":true}'
```

3. 在 IBM Cloud 中，运行以下命令来使用存储桶凭证创建 secret：

```
$ oc -n <namespace> create secret generic <ibm_cos_secret> \
  --from-literal=bucket="<tempo_bucket>" \
  --from-literal=endpoint="<ibm_bucket_endpoint>" \
  --from-literal=access_key_id="<ibm_bucket_access_key>" \
  --from-literal=access_key_secret="<ibm_bucket_secret_key>"
```

4. 在 OpenShift Container Platform 中，使用键创建一个对象存储 secret，如下所示：

```
apiVersion: v1
kind: Secret
metadata:
  name: <ibm_cos_secret>
stringData:
  bucket: <tempo_bucket>
  endpoint: <ibm_bucket_endpoint>
  access_key_id: <ibm_bucket_access_key>
  access_key_secret: <ibm_bucket_secret_key>
type: Opaque
```

5. 在 OpenShift Container Platform 中，设置 **TempoStack** 自定义资源中的 storage 部分，如下所示：

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
# ...
spec:
# ...
  storage:
    secret:
      name: <ibm_cos_secret> (1)
      type: s3
# ...
```

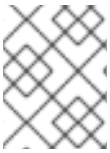
(1) 包含 IBM Cloud Storage 访问和 secret 密钥的 secret 名称。

其他资源

- OpenShift CLI 入门
- IBM Cloud CLI 入门（IBM Cloud 文档）
- 选择计划并创建实例 (IBM Cloud Docs)
- IBM Cloud Object Storage 入门：开始之前（IBM Cloud Docs）

3.3. 配置权限和租户.

在安装 **TempoStack** 或 **TempoMonolithic** 实例之前，您必须定义一个或多个租户并配置其读写访问权限。您可以使用 Kubernetes 基于角色的访问控制(RBAC)的集群角色和集群角色绑定来配置这样的授权设置。默认情况下，没有用户授予读取或写入权限。如需更多信息，请参阅"配置租户的读取权限"和"配置租户的写入权限"。



注意

红帽构建的 OpenTelemetry 收集器可以使用带有 RBAC 的服务帐户来编写数据，将 trace 数据发送到 **TempoStack** 或 **TempoMonolithic** 实例。

表 3.2. 认证和授权

组件	Tempo Gateway 服务	OpenShift OAuth	TokenReview API	SubjectAccess Review API
身份验证	X	X	X	
授权	X			X

3.3.1. 为租户配置读取权限

您可以从 web 控制台或命令行的 **Administrator** 视图为租户配置读取权限。

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. 使用您选择的值添加到 **TempoStack** 自定义资源 (CR)，通过添加 **tenantName** 和 **tenantId** 参数来定义租户：

   TempoStack CR 中的租户示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: redmetrics
spec:
# ...
  tenants:
    mode: openshift
    authentication:
      - tenantName: dev (1)
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa" (2)
# ...
```

   (1) 用户选择的 **tenantName** 值。

   (2) 用户选择的 **tenantId** 值。

2. 将租户添加到具有读取(**get**)权限的集群角色，以读取 trace。

   ClusterRole 资源中的 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)
```

1. 在本示例中列出租户 **dev** 和 **prod**，这通过上一步中的 **tenantName** 参数来定义。
2. 为列出的租户启用读取操作。

3. 通过为上一步中的集群角色定义集群角色绑定，授予经过身份验证的用户对 trace 数据具有读取权限。

ClusterRoleBinding 资源中的 RBAC 配置示例

```
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 (1)
```

1. 授予所有经过身份验证的用户的 trace 数据读取权限。

3.3.2. 为租户配置写入权限

您可以从 web 控制台或命令行的 **Administrator** 视图为租户配置写入权限。

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。
- 已安装 OpenTelemetry Collector，并将其配置为使用具有权限的授权服务帐户。如需更多信息，请参阅 Red Hat build of OpenTelemetry 文档中的"自动创建所需的 RBAC 资源"。

流程

1. 创建用于 OpenTelemetry Collector 的服务帐户。

```
apiVersion: v1
kind: ServiceAccount
```

```
metadata:
  name: otel-collector
  namespace: <project_of_opentelemetry_collector_instance>
```

2. 将租户添加到具有 write (**create**)权限的集群角色中来编写 trace。

ClusterRole 资源中的 RBAC 配置示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tempostack-traces-write
rules:
  - apiGroups:
      - 'tempo.grafana.com'
    resources: (1)
      - dev
    resourceNames:
      - traces
    verbs:
      - 'create' (2)
```

1. 列出租户。
2. 启用写入操作。

3. 通过定义集群角色绑定来附加 OpenTelemetry Collector 服务帐户，为 OpenTelemetry Collector 授予写入权限。

ClusterRoleBinding 资源中的 RBAC 配置示例

```
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 (1)
    namespace: otel
```

1. 上一步中创建的服务帐户。客户端在导出 trace 数据时使用它。

4. 配置 **OpenTelemetryCollector** 自定义资源，如下所示：
   - 将 **bearertokenauth** 扩展和有效令牌添加到追踪管道服务。
   - 在 **otlp**/**otlphttp** exporters 中添加租户名称，作为 **X-Scope-OrgID** 标头。
   - 使用有效的证书颁发机构文件启用 TLS。

OpenTelemetry CR 配置示例

```
apiVersion: opentelemetry.io/v1beta1
kind: OpenTelemetryCollector
metadata:
  name: cluster-collector
  namespace: <project_of_tempostack_instance>
spec:
  mode: deployment
  serviceAccount: otel-collector (1)
  config: |
    extensions:
      bearertokenauth: (2)
        filename: "/var/run/secrets/kubernetes.io/serviceaccount/token" (3)
    exporters:
      otlp/dev: (4)
        endpoint: sample-gateway.tempo.svc.cluster.local:8090
        tls:
          insecure: false
          ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt" (5)
        auth:
          authenticator: bearertokenauth
        headers:
          X-Scope-OrgID: "dev" (6)
      otlphttp/dev: (7)
        endpoint: https://sample-gateway.
<project_of_tempostack_instance>.svc.cluster.local:8080/api/traces/v1/dev
        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] (8)

# ...
```

1. 配置具有写入权限的服务帐户。
2. Bearer Token 扩展，以使用服务帐户令牌。
3. 服务帐户令牌。客户端将令牌发送到追踪管道服务，作为 bearer 令牌标头。
4. 指定 OTLP gRPC Exporter (**otlp/dev**)或 OTLP HTTP Exporter (**otlphttp/dev**)。
5. 使用有效的服务 CA 文件启用 TLS。
6. 带有租户名称的标头。
7. 指定 OTLP gRPC Exporter (**otlp/dev**)或 OTLP HTTP Exporter (**otlphttp/dev**)。

8 您在 CR 的 **exporters** 部分指定的导出器。

其他资源

- 自动创建所需的 RBAC 资源

3.4. 安装 TEMPOSTACK 实例

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

3.4.1. 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 Distributed Tracing Platform 中。在安装 Distributed Tracing Platform (Tempo) 前，您必须通过受支持的供应商选择和设置对象存储。

- 您已定义一个或多个租户，并配置了读写权限。如需更多信息，请参阅"配置租户的读取权限"和"配置租户的写入权限"。

流程

1. 前往 **Home** → **Projects** → **Create Project**，为后续步骤中创建的 **TempoStack** 实例创建一个您选择的允许项目。不允许以 **openshift-** 前缀开头的项目名称。
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**
>
> 您可以在同一集群上的不同项目中创建多个 **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)：

AWS S3 和 MinIO 存储和两个租户的 TempoStack CR 示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack (1)
metadata:
  name: simplest
  namespace: <permitted_project_of_tempostack_instance> (2)
spec: (3)
  storage: (4)
    secret: (5)
      name: <secret_name> (6)
      type: <secret_provider> (7)
  storageSize: <value>Gi (8)
  resources: (9)
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  tenants:
    mode: openshift (10)
    authentication: (11)
      - tenantName: dev (12)
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa" (13)
      - tenantName: prod
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfb"
  template:
    gateway:
      enabled: true (14)
    queryFrontend:
      jaegerQuery:
        enabled: true (15)
```

(1) 此 CR 创建一个 **TempoStack** 部署，它被配置为通过 HTTP 和 OpenTelemetry 协议(OTLP)接收 Jaeger Thrift。

2. 为 **TempoStack** 部署选择的项目。不允许以 **openshift-** 前缀开头的项目名称。
3. 红帽只支持 Red Hat OpenShift distributed Tracing Platform 文档中的提供的自定义资源选项。
4. 指定用于存储 trace 的存储。
5. 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。
6. secret 的 **metadata** 部分中的 **name** 字段的值。例如：**minio**。
7. 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）。例如：**s3**。
8. Tempo Write-Ahead Logging (WAL) 的持久性卷声明的大小。默认值为 **10Gi**。例如：**1Gi**。
9. 可选。
10. 该值必须是 **openshift**。
11. 租户列表。
12. 租户名称，用作 **X-Scope-OrgId** HTTP 标头的值。
13. 租户的唯一标识符。在 **TempoStack** 部署的生命周期内必须是唯一的。Distributed Tracing Platform 使用此 ID 为对象存储中的对象添加前缀。您可以重复使用 UUID 或 **tempoName** 字段的值。
14. 启用执行身份验证和授权的网关。
15. 公开 Jaeger UI，它通过路由 **http://<gateway_ingress>/api/traces/v1/<tenant_name>/search** 对数据进行视觉化。

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。

3.4.2. 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 Distributed Tracing Platform 中。在安装 Distributed Tracing Platform (Tempo) 前，您必须通过受支持的供应商选择和设置对象存储。

- 您已定义一个或多个租户，并配置了读写权限。如需更多信息，请参阅"配置租户的读取权限"和"配置租户的写入权限"。

流程

1. 运行以下命令，为您要在后续步骤中创建的 **TempoStack** 实例创建您选择的允许项目：

   ```
   $ oc apply -f - << EOF
   apiVersion: project.openshift.io/v1
   kind: Project
   metadata:
     name: <permitted_project_of_tempostack_instance> (1)
   EOF
   ```

   (1) 不允许以 **openshift-** 前缀开头的项目名称。

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. 在您为它创建的项目中创建 **TempoStack** 实例：

注意

您可以在同一集群上的不同项目中创建多个 **TempoStack** 实例。

a. 自定义 **TempoStack** 自定义资源(CR)：

AWS S3 和 MinIO 存储和两个租户的 TempoStack CR 示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack (1)
metadata:
  name: simplest
  namespace: <permitted_project_of_tempostack_instance> (2)
spec: (3)
  storage: (4)
    secret: (5)
      name: <secret_name> (6)
      type: <secret_provider> (7)
  storageSize: <value>Gi (8)
  resources: (9)
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  tenants:
    mode: openshift (10)
    authentication: (11)
      - tenantName: dev (12)
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa" (13)
      - tenantName: prod
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfb"
```

```
template:
  gateway:
    enabled: true (14)
  queryFrontend:
    jaegerQuery:
      enabled: true (15)
```

1. 此 CR 创建一个 **TempoStack** 部署，它被配置为通过 HTTP 和 OpenTelemetry 协议 (OTLP)接收 Jaeger Thrift。
2. 为 **TempoStack** 部署选择的项目。不允许以 **openshift-** 前缀开头的项目名称。
3. 红帽只支持 Red Hat OpenShift distributed Tracing Platform 文档中的提供的自定义资源选项。
4. 指定用于存储 trace 的存储。
5. 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。
6. secret 的 **metadata** 部分中的 **name** 字段的值。例如：**minio**。
7. 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）。例如：**s3**。
8. Tempo Write-Ahead Logging (WAL) 的持久性卷声明的大小。默认值为 **10Gi**。例如：**1Gi**。
9. 可选。
10. 该值必须是 **openshift**。
11. 租户列表。
12. 租户名称，用作 **X-Scope-OrgId** HTTP 标头的值。
13. 租户的唯一标识符。在 **TempoStack** 部署的生命周期内必须是唯一的。Distributed Tracing Platform 使用此 ID 为对象存储中的对象添加前缀。您可以重复使用 UUID 或 **tempoName** 字段的值。
14. 启用执行身份验证和授权的网关。
15. 公开 Jaeger UI，它通过路由 **http://<gateway_ingress>/api/traces/v1/<tenant_name>/search** 对数据进行视觉化。

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 components** 的状态是否为 **Running**，并且 **conditions** 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 是否都在运行：

```
$ 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。

3.5. 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，请参阅以下链接：

- 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 实例支持存储 trace in-memory storage、持久性卷或对象存储。

在单体模式中进行临时部署是小型部署、演示和测试的首选方式。

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。

3.5.1. 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，请参阅以下链接：

- 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。
- 您已定义一个或多个租户，并配置了读写权限。如需更多信息，请参阅"配置租户的读取权限"和"配置租户的写入权限"。

流程

1. 前往 **Home** → **Projects** → **Create Project**，为后续步骤中创建的 **TempoMonolithic** 实例创建您选择的允许项目。不允许以 **openshift-** 前缀开头的项目名称。
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 Distributed Tracing Platform 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

您可以在同一集群上的不同项目中创建多个 **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)。

TempoMonolithic CR 示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic (1)
metadata:
  name: <metadata_name>
  namespace: <permitted_project_of_tempomonolithic_instance> (2)
spec: (3)
  storage: (4)
    traces:
      backend: <supported_storage_type> (5)
      size: <value>Gi (6)
      s3: (7)
        secret: <secret_name> (8)
    tls: (9)
      enabled: true
      caName: <ca_certificate_configmap_name> (10)
  jaegerui:
    enabled: true (11)
    route:
      enabled: true (12)
  resources: (13)
    total:
      limits:
        memory: <value>Gi
        cpu: <value>m
  multitenancy:
    enabled: true
    mode: openshift
    authentication: (14)
      - tenantName: dev (15)
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa" (16)
      - tenantName: prod
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfb"
```

1. 此 CR 使用 OTLP 协议中的 trace ingestion 创建一个 **TempoMonolithic** 部署。
2. 为 **TempoMonolithic** 部署选择的项目。不允许以 **openshift-** 前缀开头的项目名称。
3. 红帽只支持 Red Hat OpenShift distributed Tracing Platform 文档中的提供的自定义资源选项。
4. 指定用于存储 trace 的存储。

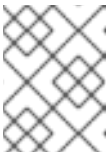
5. 用于存储 trace 的存储类型：内存存储、持久性卷或对象存储。持久性卷的值是 **pv**。对象存储接受的值是 **s3**、**gcs** 或 **azure**，具体取决于使用的对象存储类型。**tmpfs** 内存中
6. 内存大小：对于内存存储，这意味着 **tmpfs** 卷的大小，默认值为 **2Gi**。对于持久性卷，这意味着持久性卷声明的大小，默认值为 **10Gi**。对于对象存储，这意味着 Tempo Write-Ahead Logging (WAL)的持久性卷声明的大小，默认值为 **10Gi**。
7. 可选：对于对象存储，对象存储的类型。接受的值包括 **s3**、**gcs** 和 **azure**，具体取决于使用的对象存储类型。
8. 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"。
9. 可选。
10. 可选：包含 CA 证书的 **ConfigMap** 对象的名称。
11. 公开 Jaeger UI，它通过路由 **http://<gateway_ingress>/api/traces/v1/<tenant_name>/search** 对数据进行视觉化。
12. 启用为 Jaeger UI 创建路由。
13. 可选。
14. 列出租户。
15. 租户名称，用作 **X-Scope-OrgId** HTTP 标头的值。
16. 租户的唯一标识符。在 **TempoMonolithic** 部署的整个生命周期内必须是唯一的。此 ID 将作为前缀添加到对象存储中的对象。您可以重复使用 UUID 或 **tempoName** 字段的值。

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**，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 **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。

3.5.2. 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，请参阅以下链接：

- 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>
    ```

- 您已定义一个或多个租户，并配置了读写权限。如需更多信息，请参阅"配置租户的读取权限"和"配置租户的写入权限"。

流程

1. 运行以下命令，为您要在后续步骤中创建的 **TempoMonolithic** 实例创建您选择的允许项目：

   ```
   $ oc apply -f - << EOF
   apiVersion: project.openshift.io/v1
   kind: Project
   metadata:
     name: <permitted_project_of_tempomonolithic_instance> (1)
   EOF
   ```

   (1) 不允许以 **openshift-** 前缀开头的项目名称。

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 Distributed Tracing Platform 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. 在您为它创建的项目中创建 **TempoMonolithic** 实例。

提示

您可以在同一集群上的不同项目中创建多个 **TempoMonolithic** 实例。

a. 自定义 **TempoMonolithic** 自定义资源 (CR)。

TempoMonolithic CR 示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoMonolithic (1)
metadata:
  name: <metadata_name>
  namespace: <permitted_project_of_tempomonolithic_instance> (2)
spec: (3)
  storage: (4)
    traces:
      backend: <supported_storage_type> (5)
      size: <value>Gi (6)
      s3: (7)
        secret: <secret_name> (8)
    tls: (9)
      enabled: true
```

```
    caName: <ca_certificate_configmap_name> (10)
jaegerui:
  enabled: true (11)
  route:
    enabled: true (12)
resources: (13)
  total:
    limits:
      memory: <value>Gi
      cpu: <value>m
multitenancy:
  enabled: true
  mode: openshift
  authentication: (14)
    - tenantName: dev (15)
      tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa" (16)
    - tenantName: prod
      tenantId: "1610b0c3-c509-4592-a256-a1871353dbfb"
```

1. 此 CR 使用 OTLP 协议中的 trace ingestion 创建一个 **TempoMonolithic** 部署。
2. 为 **TempoMonolithic** 部署选择的项目。不允许以 **openshift-** 前缀开头的项目名称。
3. 红帽只支持 Red Hat OpenShift distributed Tracing Platform 文档中的提供的自定义资源选项。
4. 指定用于存储 trace 的存储。
5. 用于存储 trace 的存储类型：内存存储、持久性卷或对象存储。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。**tmpfs** 内存中存储的默认值为 **memory**，它仅适用于开发、测试、演示和概念验证环境，因为在 pod 关闭时数据不会被保留。
6. 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 Write-Ahead Logging (WAL)的持久性卷声明的大小，默认值为 **10Gi**。
7. 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。
8. 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"。
9. 可选。
10. 可选：包含 CA 证书的 **ConfigMap** 对象的名称。
11. 公开 Jaeger UI，它通过路由 **http://<gateway_ingress>/api/traces/v1/<tenant_name>/search** 对数据进行视觉化。
12. 启用为 Jaeger UI 创建路由。
13. 可选。

14. 列出租户。
15. 租户名称，用作 **X-Scope-OrgId** HTTP 标头的值。
16. 租户的唯一标识符。在 **TempoMonolithic** 部署的整个生命周期内必须是唯一的。此 ID 将作为前缀添加到对象存储中的对象。您可以重复使用 UUID 或 **tempoName** 字段的值。

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**，并且 **conditions** 为 **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 **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。

3.6. 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

第 4 章 配置 DISTRIBUTED TRACING PLATFORM

Tempo Operator 使用自定义资源定义(CRD)文件来定义创建和部署 Distributed Tracing Platform 资源的架构和配置设置。您可以安装默认配置或修改该文件。

4.1. 配置后端存储

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。

4.2. TEMPOSTACK 配置参数简介

TempoStack 自定义资源(CR)定义用于创建 Distributed Tracing Platform 资源的架构和设置。您可以根据您的业务需求，修改这些参数以自定义您的实现。

TempoStack CR 示例

```
apiVersion: tempo.grafana.com/v1alpha1 (1)
kind: TempoStack (2)
metadata: (3)
  name: <name> (4)
spec: (5)
  storage: {} (6)
  resources: {} (7)
  replicationFactor: 1 (8)
  retention: (9)
    global:
      traces: 48h
    perTenant: {}
  template:
      distributor: {} (10)
      ingester: {} (11)
      compactor: {} (12)
      querier: {} (13)
      queryFrontend: {} (14)
      gateway: {} (15)
  limits: (16)
    global:
      ingestion: {} (17)
      query: {} (18)
  observability: (19)
    grafana: {}
    metrics: {}
    tracing: {}
  search: {} (20)
  managementState: managed (21)
```

1. 创建对象时要使用的 API 版本。
2. 定义要创建的 Kubernetes 对象的种类。
3. 唯一标识对象的数据，包括 **name** 字符串，**UID**, 和可选的 **namespace**。OpenShift Container

4. TempoStack 实例的名称。
5. 包含 TempoStack 实例的所有配置参数。当需要一个适用于所有 Tempo 组件的通用定义时，在 **spec** 部分中定义它。当定义只与单个组件相关时，将其放在 **spec.template.<component>** 部分中。
6. 存储在实例部署中指定。有关实例的存储选项的信息，请参阅安装页。
7. 为 Tempo 容器定义计算资源。
8. 在接受范围之前，必须确认经销商中的数据数量的整数值。
9. 保留 trace 的配置选项。默认值为 **48h**。
10. Tempo **distributor** 组件的配置选项。
11. Tempo **ingester** 组件的配置选项。
12. Tempo **compactor** 组件的配置选项。
13. Tempo **querier** 组件的配置选项。
14. Tempo **query-frontend** 组件的配置选项。
15. Tempo **gateway** 组件的配置选项。
16. 摄入（ingestion）和查询（query）率限制。
17. 定义摄入率限制。
18. 定义查询率限制。
19. 配置操作对象以处理遥测数据。
20. 配置搜索功能。
21. 定义此 CR 是否由 Operator 管理。默认值为 **managed**。

表 4.1. TempoStack CR 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**	

参数	描述	值	默认值
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。		
retention:	保留 trace 的配置选项。		
storage：	定义存储的配置选项。		
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 查询前端的配置选项。		

参数	描述	值	默认值
template.gateway:	Tempo 网关的配置选项。		

其他资源

- 安装 TempoStack 实例
- 安装 TempoMonolithic 实例

4.3. 查询配置选项

Distributed Tracing Platform 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 4.2. 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 4.3. 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

参数	描述	值
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)
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:9092**。	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
```

其他资源

- 了解污点和容限

4.4. 配置 UI

您可以使用 Cluster Observability Operator (COO) 的 Distributed tracing UI 插件作为 Red Hat OpenShift distributed Tracing Platform 的用户界面 (UI)。有关安装和使用 Distributed tracing UI 插件的更多信息，请参阅 *Cluster Observability Operator* 中的"Distributed tracing UI 插件"。

其他资源

- 分布式追踪 UI 插件

4.5. 在 JAEGER UI 中配置 MONITOR 选项卡

您可以在 OpenShift Container Platform Web 控制台的 **Monitor** 选项卡中通过 Jaeger Console 提取请求率、错误和持续时间(RED)指标，并通过 Jaeger Console 进行视觉化。指标来自 OpenTelemetry Collector 中的 span，由 Prometheus 从 Collector 中提取，您可以在用户工作负载监控堆栈中部署。Jaeger UI 从 Prometheus 端点查询这些指标，并视觉化它们。

先决条件

- 您已为 Distributed Tracing Platform 配置了权限和租户。如需更多信息，请参阅"配置权限和租户"。

流程

1. 在 **OpenTelemetry Collector** 的 OpenTelemetryCollector 自定义资源中，启用 Spanmetrics Connector (**spanmetrics**)，它将从 trace 派生指标，并以 Prometheus 格式导出指标。

   span RED 的 OpenTelemetryCollector 自定义资源示例

   ```
   apiVersion: opentelemetry.io/v1beta1
   kind: OpenTelemetryCollector
   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 (5)

      otlp:
        auth:
          authenticator: bearertokenauth
        endpoint: tempo-redmetrics-gateway.mynamespace.svc.cluster.local:8090
        headers:
          X-Scope-OrgID: dev
        tls:
          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt
          insecure: false

    extensions:
      bearertokenauth:
        filename: /var/run/secrets/kubernetes.io/serviceaccount/token

    service:
      extensions:
      - bearertokenauth
      pipelines:
        traces:
          receivers: [otlp]
          exporters: [otlp, spanmetrics] (6)
        metrics:
          receivers: [spanmetrics] (7)
          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. 默认丢弃资源属性。
6. Spanmetrics 连接器在 trace 管道中被配置为 exporter。
7. Spanmetrics 连接器在 metrics 管道中配置为接收器。

2. 在 **TempoStack** 自定义资源中，启用 **Monitor** 选项卡，并将 Prometheus 端点设置为 Thanos querier 服务，以从用户定义的监控堆栈查询数据。

带有启用 Monitor 选项卡的 TempoStack 自定义资源示例

```
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: redmetrics
spec:
  storage:
    secret:
      name: minio-test
      type: s3
  storageSize: 1Gi
  tenants:
    mode: openshift
    authentication:
      - tenantName: dev
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfa"
  template:
    gateway:
      enabled: true
    queryFrontend:
      jaegerQuery:
        monitorTab:
          enabled: true (1)
          prometheusEndpoint: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
(2)

          redMetricsNamespace: "" (3)

# ...
```

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. 可选： Jaeger 查询检索 Prometheus 指标的指标命名空间。只有在您使用比 0.109.0 更早的 OpenTelemetry Collector 版本时，才包含这一行。如果您使用 OpenTelemetry Collector 版本 0.109.0 或更高版本，请省略这一行。

3. 可选：使用带有警报规则的 **spanmetrics** 连接器生成的 span RED 指标。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 4.4. 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**
span_name	操作的名称。	• / • **/customer**
span_kind	标识服务器、客户端、消息传递或内部操作。	• **SPAN_KIND_SERVER** • **SPAN_KIND_CLIENT** • **SPAN_KIND_PRODUCER** • **SPAN_KIND_CONSUMER** • **SPAN_KIND_INTERNAL**

PrometheusRule 自定义资源示例，当前端服务于 2000ms 内没有提供 95% 时 SLO 定义警报规则

```
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。

其他资源

- 配置权限和租户.

4.6. 配置接收器 TLS

TempoStack 或 TempoMonolithic 实例的自定义资源支持使用用户提供的证书或 OpenShift 的服务证书为接收器配置 TLS。

4.6.1. TempoStack 实例的接收器 TLS 配置

您可以在 secret 中提供 TLS 证书，或使用 OpenShift Container Platform 生成的服务证书。

- 要在 secret 中提供 TLS 证书，请在 **TempoStack** 自定义资源中配置它。

注意

启用的 Tempo Gateway 不支持此功能。

TLS 用于接收器并在 secret 中使用用户提供的证书

```
apiVersion: tempo.grafana.com/v1alpha1
kind:  TempoStack
# ...
spec:
# ...
  template:
    distributor:
      tls:
        enabled: true (1)
        certName: <tls_secret> (2)
        caName: <ca_name> (3)
# ...
```

1. 在 Tempo Distributor 上启用 TLS。
2. 包含您提前应用的 **tls.key** 密钥和 **tls.crt** 证书的 secret。
3. 可选：配置映射中的 CA 来启用 mutual TLS 身份验证 (mTLS)。

- 另外，您可以使用 OpenShift Container Platform 生成的服务证书。

注意

此功能不支持 mutual TLS 身份验证 (mTLS)。

用于接收器的 TLS 并使用 OpenShift Container Platform 生成的服务证书

```
apiVersion: tempo.grafana.com/v1alpha1
kind:  TempoStack
# ...
spec:
# ...
  template:
    distributor:
```

```
      tls:
        enabled: true (1)
# ...
```

1. 在 Tempo Distributor 为 TLS 进行足够配置。

其他资源

- 了解服务用证书
- 服务 CA 证书

4.6.2. TempoMonolithic 实例的接收器 TLS 配置

您可以在 secret 中提供 TLS 证书，或使用 OpenShift Container Platform 生成的服务证书。

- 要在 secret 中提供 TLS 证书，请在 **TempoMonolithic** 自定义资源中配置它。

> **注意**
>
> 启用的 Tempo Gateway 不支持此功能。

TLS 用于接收器并在 secret 中使用用户提供的证书

```
apiVersion: tempo.grafana.com/v1alpha1
kind:  TempoMonolithic
# ...
  spec:
# ...
  ingestion:
    otlp:
      grpc:
        tls:
          enabled: true (1)
          certName: <tls_secret> (2)
          caName: <ca_name> (3)
# ...
```

1. 在 Tempo Distributor 上启用 TLS。
2. 包含您提前应用的 **tls.key** 密钥和 **tls.crt** 证书的 secret。
3. 可选：配置映射中的 CA 来启用 mutual TLS 身份验证 (mTLS)。

- 另外，您可以使用 OpenShift Container Platform 生成的服务证书。

> **注意**
>
> 此功能不支持 mutual TLS 身份验证 (mTLS)。

用于接收器的 TLS 并使用 OpenShift Container Platform 生成的服务证书

```
apiVersion: tempo.grafana.com/v1alpha1
kind:  TempoMonolithic
# ...
  spec:
# ...
  ingestion:
    otlp:
      grpc:
        tls:
          enabled: true
      http:
        tls:
          enabled: true (1)
# ...
```

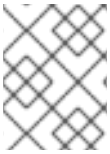
(1) 在 Tempo Distributor 为 TLS 的最小配置。

其他资源

- 了解服务用证书
- 服务 CA 证书

4.7. 配置查询 RBAC

作为管理员，您可以设置查询基于角色的访问控制(RBAC)，以便根据您授予其权限的命名空间过滤用户的 span 属性。



注意

当您启用查询 RBAC 时，用户仍然可以从所有命名空间访问 trace，**service.name** 和 **k8s.namespace.name** 属性对所有用户可见。

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. 在 **TempoStack** 自定义资源(CR) 中启用多租户和查询 RBAC，例如：

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

```
metadata:
  name: simplest
  namespace: chainsaw-multitenancy
spec:
  storage:
    secret:
      name: minio
      type: s3
  storageSize: 1Gi
  resources:
    total:
      limits:
        memory: 2Gi
        cpu: 2000m
  tenants:
    mode: openshift
    authentication:
      - tenantName: dev
        tenantId: "1610b0c3-c509-4592-a256-a1871353dbfb"
  template:
    gateway:
      enabled: true (1)
      rbac:
        enabled: true (2)
    queryFrontend:
      jaegerQuery:
        enabled: false (3)
```

1. 始终设置为 **true**。
2. 始终设置为 **true**。
3. 始终设置为 **false**。

2. 创建集群角色和集群角色绑定，为目标用户授予访问您在 **TempoStack** CR 中指定的租户的权限，例如：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tempo-dev-read
rules:
- apiGroups: [tempo.grafana.com]
  resources: [dev] (1)
  resourceNames: [traces]
  verbs: [get]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tempo-dev-read
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tempo-dev-read
```

```
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated (2)
```

1. **TempoStack** CR 中的租户名称。
2. 表示所有通过身份验证的 OpenShift 用户。

3. 为目标用户授予项目读取属性的权限。您可以运行以下命令来完成此操作：

```
$ oc adm policy add-role-to-user view <username> -n <project>
```

4.8. 使用污点和容限

要将 TempoStack pod 调度到专用节点上，请参阅 如何在 OpenShift 4 中使用 nodeSelector 和 tolerations 在 infra 节点上部署不同的 TempoStack 组件。

4.9. 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。

4.9.1. 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。

其他资源

- 为用户定义的项目启用监控

4.9.2. 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**。

第 5 章 DISTRIBUTED TRACING PLATFORM 故障排除

您可以使用各种故障排除方法诊断和修复 TempoStack 或 TempoMonolithic 实例中的问题。

5.1. 从命令行收集诊断数据

在提交问题单时，向红帽提供包含有关集群的诊断信息会很有帮助。您可以使用 **oc adm must-gather** 工具为各种类型的资源收集诊断数据，如 **TempoStack** 或 **TempoMonolithic**，以及创建的资源，如 **Deployment**、**Pod** 或 **ConfigMap**。**oc adm must-gather** 工具会创建一个收集此数据的新 pod。

流程

- 从您要保存收集的数据的目录中，运行 **oc adm must-gather** 命令来收集数据：

```
$ oc adm must-gather --image=ghcr.io/grafana/tempo-operator/must-gather -- \
/usr/bin/must-gather --operator-namespace <operator_namespace> (1)
```

1. 安装 Operator 的默认命名空间是 **openshift-tempo-operator**。

验证

- 验证新目录是否已创建并包含收集的数据。

第 6 章 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。

6.1. 其他资源

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

第 7 章 删除 DISTRIBUTED TRACING PLATFORM

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

1. 关闭所有 Distributed Tracing Platform pod。
2. 删除任何 TempoStack 实例。
3. 删除 Tempo Operator。

7.1. 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。

7.2. 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>
```

7.3. 其他资源

- 从集群中删除 Operator
- OpenShift CLI 入门