Red Hat Streams for Apache Kafka 2.7

OpenShift 概述中的 Apache Kafka 流

在 OpenShift Container Platform 中发现 AMQ Streams 2.7 的功能

Last Updated: 2024-05-29

Red Hat Streams for Apache Kafka 2.7 OpenShift 概述中的 Apache Kafka 流

在 OpenShift Container Platform 中发现 AMQ Streams 2.7 的功能

法律通告

摘要

了解 Kafka 组件的功能，以及如何使用 AMQ Streams 在 OpenShift 中部署和管理 Kafka。

目录

前言

对红帽文档提供反馈

我们感谢您对我们文档的反馈。

要改进，创建一个 JIRA 问题并描述您推荐的更改。提供尽可能多的详细信息，以便我们快速解决您的请求。

前提条件

- 您有红帽客户门户网站帐户。此帐户可让您登录到 Red Hat Jira Software 实例。
  如果您没有帐户，系统会提示您创建一个帐户。

流程

1. 点以下内容： Create issue。
2. 在 **Summary** 文本框中输入问题的简短描述。
3. 在 **Description** 文本框中提供以下信息：
   - 找到此问题的页面的 URL。
   - 有关此问题的详细描述。
     您可以将信息保留在任何其他字段中的默认值。
4. 添加 reporter 名称。
5. 点 **Create** 将 JIRA 问题提交到文档团队。

感谢您花时间来提供反馈。

第 1 章 主要特性

Apache Kafka 的流简化了在 OpenShift 集群中运行 Apache Kafka 的过程。

本指南旨在作为构建 Apache Kafka 的流了解的起点。本指南介绍了 Kafka 背后的一些关键概念，它是 Apache Kafka 的核心概念，解释 Kafka 组件的目的。概述了配置点，包括安全和监控 Kafka 的选项。Apache Kafka 的 Streams 发行版提供了部署和管理 Kafka 集群 的文件，以及配置和监控部署的示例文件。

描述了典型的 Kafka 部署，以及部署和管理 Kafka 的工具。

1.1. KAFKA 功能

Kafka 的底层数据流处理功能和组件架构可以提供：

- 微服务和其他应用，以极高吞吐量和低延迟共享数据
- 消息排序保证
- 从数据存储中递归/恢复消息，以重新构建应用程序状态
- 使用键值日志时，消息压缩以删除旧记录
- 集群配置中的水平可扩展性
- 复制数据以控制容错
- 保留大量数据以即时访问

1.2. KAFKA 用例

Kafka 的功能使其适合：

- 事件驱动的构架
- 事件提供，将更改捕获为作为事件日志的应用程序状态
- 消息代理
- 网站活动跟踪
- 通过指标数据进行操作监控
- 日志聚合和聚合
- 为分布式系统提交日志
- 流处理，以便应用程序实时响应数据

1.3. APACHE KAFKA 的流如何支持 KAFKA

Apache Kafka 的流为在 OpenShift 中运行 Kafka 提供容器镜像和操作器。Apache Kafka operator 的流专门构建有专家操作知识，以便在 OpenShift 上有效地管理 Kafka。

Operator 简化以下过程：

- 部署并运行 Kafka 集群
- 部署并运行 Kafka 组件
- 配置对 Kafka 的访问
- 保护对 Kafka 的访问
- 升级 Kafka
- 管理代理
- 创建和管理主题
- 创建和管理用户

第 2 章 KAFKA 的 APACHE KAFKA 部署流

提供了 Apache Kafka 组件，用于使用 Streams for Apache Kafka 发行版本部署到 OpenShift。Kafka 组件通常作为集群运行，以实现高可用性。

使用 Kafka 组件的典型部署可能包括：

- 代理节点的 **Kafka** 集群
- 复制 **ZooKeeper** 实例的 zookeeper 集群
- 用于外部数据连接的 **Kafka 连接** 集群
- **Kafka MirrorMaker** 集群在二级集群中镜像 Kafka 集群
- **Kafka Exporter** 来提取额外的 Kafka 指标数据以进行监控。
- **Kafka Bridge** 为 Kafka 集群发出基于 HTTP 的请求
- **Cruise Control** 在代理节点间重新平衡主题分区

并非所有组件都是必须的，但最少需要 Kafka 和 ZooKeeper。有些组件可以在没有 Kafka 的情况下部署，如 MirrorMaker 或 Kafka Connect。

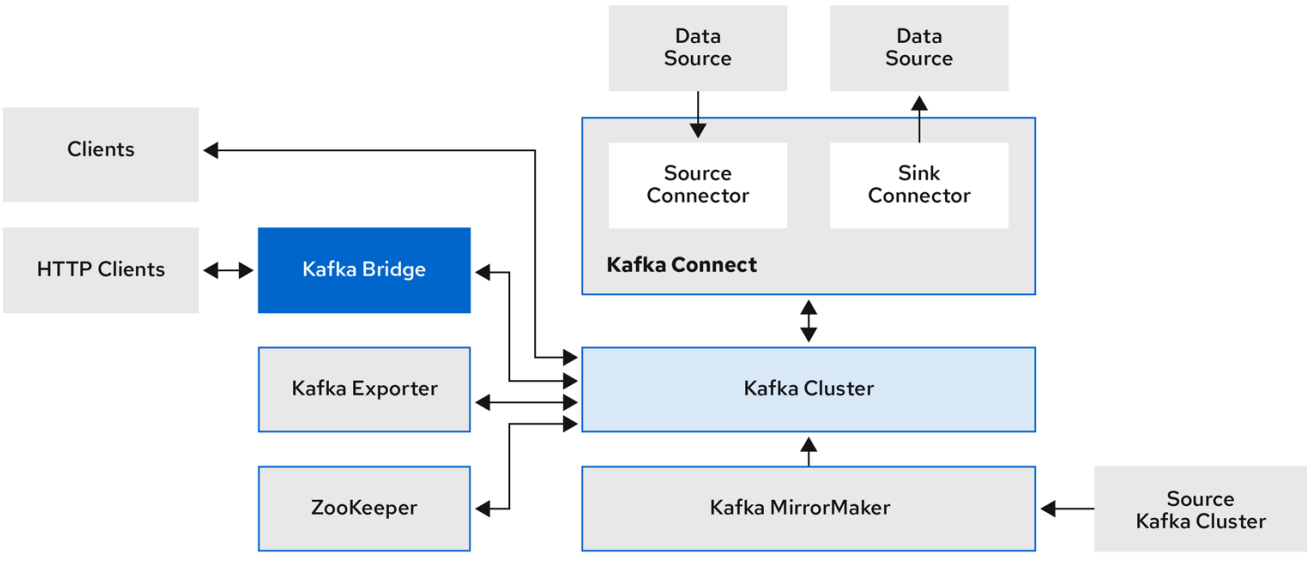
2.1. KAFKA 组件架构

Kafka 集群包含负责消息发送的代理。

ZooKeeper 用于集群管理。以 KRaft (Kafka Raft metadata)模式部署 Kafka 时，通过集成 Kafka 节点上的代理和控制器角色来简化集群管理，从而消除 ZooKeeper 的需求。Kafka 节点使用代理、控制器或两者的角色。使用节点池在 Apache Kafka 中配置了角色。

其他 Kafka 组件与 Kafka 集群交互，以执行特定的角色。

Kafka 组件交互



Apache ZooKeeper

Apache ZooKeeper 提供了一个集群协调服务，用于存储和跟踪代理和消费者的状态。Zookeeper 也用于控制器选举。如果使用 ZooKeeper，则 ZooKeeper 集群必须在运行 Kafka 前就绪。在 KRaft 模式中，不需要 ZooKeeper，因为协调由作为控制器运行的 Kafka 集群中管理。

Kafka Connect

Kafka Connect 是一个在 Kafka 代理和其他使用 *Connector* 插件系统间流传输数据的集成工具包。Kafka Connect 提供了一个框架，用于将 Kafka 与外部数据源或目标（如数据库）集成，如数据库，用于使用连接器导入或导出数据。连接器是提供所需的连接配置的插件。

- *source（源）*连接器将外部数据推送到 Kafka 中。
- *sink（接收器）*连接器从 Kafka 中提取数据
  外部数据会被转换并转换为适当的格式。

  您可以使用 **build** 配置部署 Kafka Connect，以使用您数据连接所需的连接器插件来自动构建容器镜像。

Kafka MirrorMaker

Kafka MirrorMaker 在两个 Kafka 集群或数据中心之间复制数据。
MirrorMaker 从源 Kafka 集群获取信息，并将其写入目标 Kafka 集群。

Kafka Bridge

Kafka Bridge 提供了一个 API，用于将基于 HTTP 的客户端与 Kafka 集群集成。

Kafka Exporter

Kafka Exporter 提取数据以 Prometheus 指标的形式提取，主要与偏移、消费者组、消费者和主题有关的数据。consumer lag 是写入分区的最后一条消息之间的延迟，以及当前由消费者从那个分区获取的消息之间的延迟

第 3 章 关于 KAFKA

Apache Kafka 是一个开源分布式发布订阅消息传递系统，用于容错实时数据源。

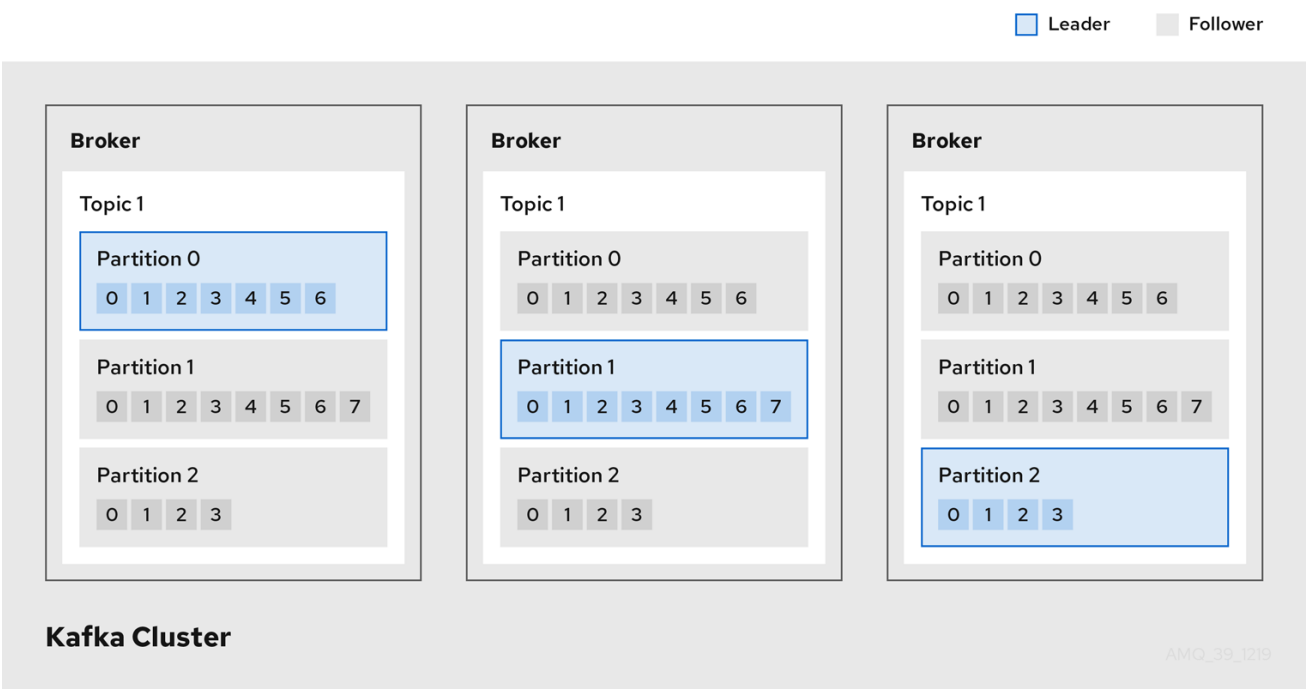
有关 Apache Kafka 的更多信息，请参阅 Apache Kafka 文档。

3.1. KAFKA 如何作为消息代理运行

要最大化您使用 Apache Kafka 的 Streams 的经验，您需要了解 Kafka 如何作为消息代理运行。

- Kafka 集群由多个节点组成。
- 作为代理运行的节点包含接收和存储数据的主题。
- 主题按分区分割，分区中写入数据。
- 分区在代理之间复制，以进行容错。

Kafka 代理和主题



Broker

代理(broker)编配存储并传递信息。

Topic

主题提供数据存储的目的地。每个主题都被分成一个或多个分区。

集群

一组代理实例。

分区

主题分区的数量由主题 *分区计数* 来定义。

分区领导

分区领导机处理主题的所有制作者请求。

分区跟踪器

分区遵循复制分区领导分区数据（可选）处理消费者请求。
主题使用 *复制因素* 来配置集群中各个分区的副本数。主题至少包含一个分区。

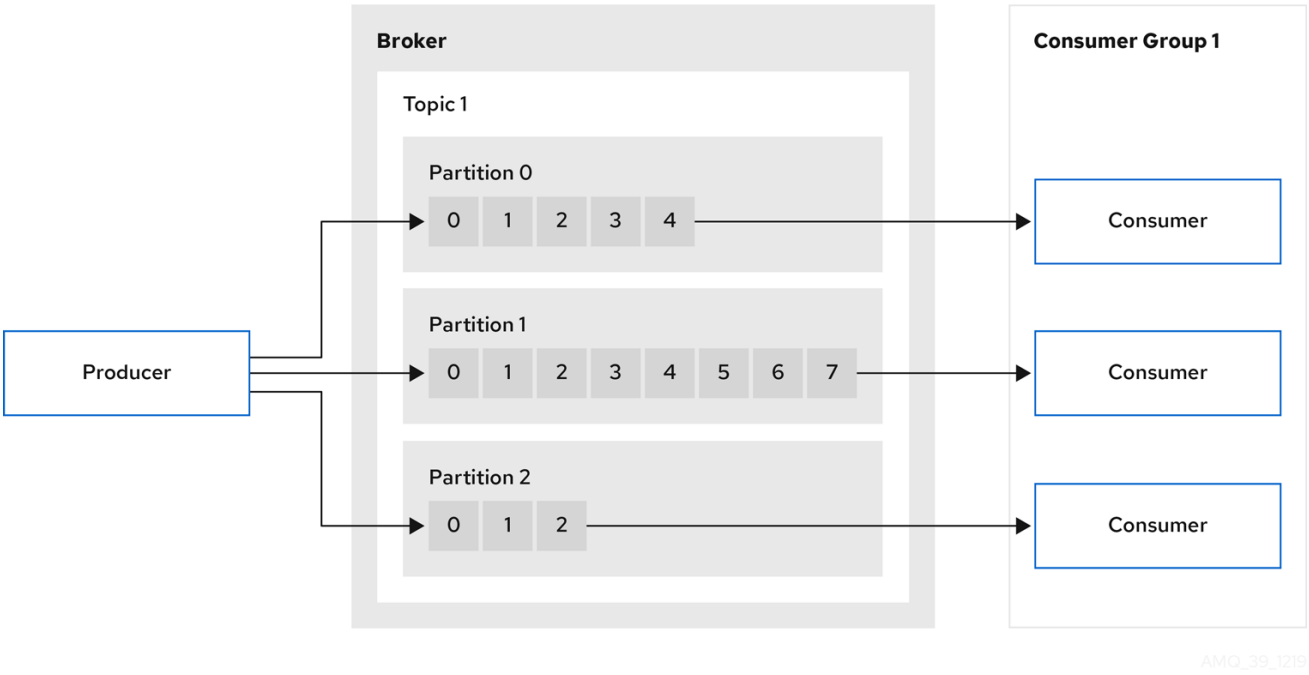
in-sync 副本的数量与领导数相同。配置定义有多少副本必须同步才能生成消息，确保只有在消息成功复制到副本分区后才会提交消息。这样，如果领导机失败，消息就不会丢失。

在 *Kafka 代理和主题* 图中，可以看到每个编号的分区都有一个领导分区，在复制的主题中有两个因素。

3.2. 生产者和消费者

生产者和消费者通过代理发送和接收消息（发布和订阅）。消息包含一个可选的 *键* 和 *值*，它包括了消息数据以及标头和相关的元数据。密钥用于识别消息的主题或消息的属性。消息在批处理中传输，并且记录包含标头和元数据，它们提供客户端过滤和路由相关的详细信息，如记录的时间戳和偏移位置。

生产者和消费者



制作者

制作者发送消息到代理主题，以写入分区的最终用户。消息由制作者按轮循方式写入分区，或根据消息键写入到特定分区。

消费者

消费者订阅一个主题，并根据主题、分区和偏移读取消息。

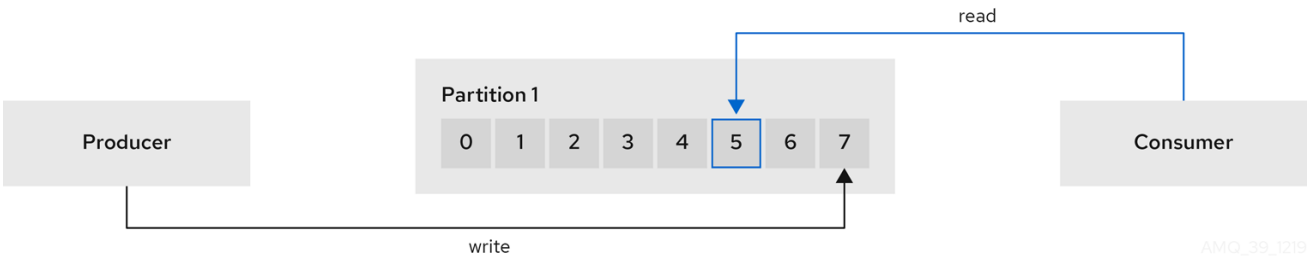
消费者组

用户组用来共享由来自给定主题的多个制作者生成的大型数据流。消费者使用 **group.id** 分组，允许消息分散到成员中。组中的消费者不会从同一分区读取数据，但可以从一个或多个分区接收数据。

Offsets

Offset 描述分区内消息的位置。给定分区中的每个消息都有一个唯一的偏移值，这有助于识别分区中消费者的位置，以跟踪已消耗的记录数。
已提交的偏移会写入偏移日志。**__consumer_offsets** 主题根据消费者组在提交偏移量上存储信息，这是最后一个偏移的位置。

生成和使用数据


read
Partition 1
0
1
2
3
4
5
6
7
Producer
Consumer
write
AMQ_39_1219

第 4 章 关于 KAFKA 连接

Kafka Connect 是一个在 Kafka 代理和其他系统间流传输数据的集成工具包。其他系统通常是外部数据源或目标，如数据库。

Kafka Connect 使用插件架构为连接器提供实施工件。插件允许连接到其他系统，并提供额外的配置来操作数据。插件包括连接器和其他组件，如数据转换器和转换。连接器使用特定类型的外部系统运行。每个连接器都定义了其配置架构。您提供到 Kafka Connect 的配置，以在 Kafka Connect 中创建连接器实例。然后，连接器实例定义了一组用于在系统之间移动数据的任务。

Apache Kafka 的流 *以分布式模式运行* Kafka Connect，在一个或多个 worker pod 间分布数据流任务。Kafka Connect 集群由一组 worker pod 组成。每个连接器都在单个 worker 上实例化。每个连接器由一个或多个在 worker 组分发的任务组成。在 worker 间分布允许高扩展管道。

worker 将数据从一个格式转换为适合源或目标系统的另一种格式。根据连接器实例的配置，worker 也可能应用转换（也称为 Single Message Transforms 或 SMT）。在信息被转换前，会对信息进行调整，如过滤某些数据。Kafka Connect 有一些内置转换，但在需要时可以通过插件来提供其他转换。

4.1. KAFKA 连接流数据

Kafka Connect 使用连接器实例来与其他系统集成以流传输数据。

Kafka Connect 在启动时加载现有连接器实例，并在 worker pod 间分配数据流任务和连接器配置。worker 为连接器实例运行任务。每个 worker 作为单独的 pod 运行，使 Kafka Connect 集群更具容错性。如果任务数量超过 worker，则 worker 会被分配多个任务。如果 worker 失败，其任务会自动分配给 Kafka Connect 集群中的活跃 worker。

流数据中使用的主要的 Kafka Connect 组件如下：

- 创建任务的连接器
- 移动数据的任务
- 用于运行任务的 worker
- 转换为操作数据
- 用于转换数据的转换器

4.1.1. 连接器

连接器可以是以下类型之一：

- 将数据推送到 Kafka 的源连接器
- 从 Kafka 中提取数据的接收器连接器

插件提供了 Kafka 连接到运行连接器实例的实施。连接器实例创建在 Kafka 中传输数据所需的任务。Kafka Connect 运行时编配在 worker pod 间分离所需工作的任务。

MirrorMaker 2 也使用 Kafka Connect 框架。在这种情况下，外部数据系统是另一个 Kafka 集群。MirrorMaker 2 的专用连接器管理源和目标 Kafka 集群之间的数据复制。

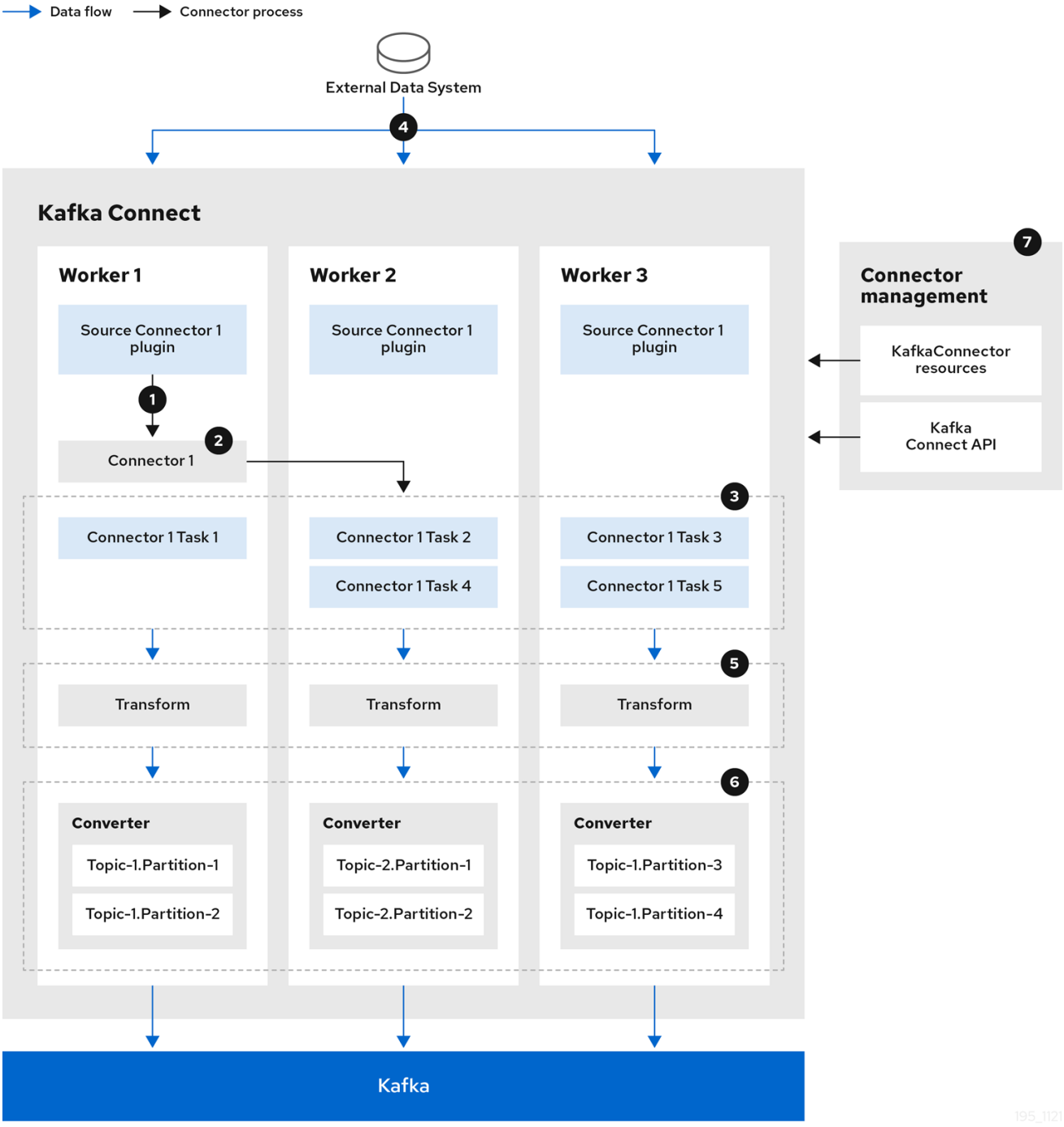
注意

除了 MirrorMaker 2 连接器外，Kafka 还提供了两个连接器作为示例：

- **FileStreamSourceConnector** 将来自 worker 文件系统上的文件的数据流到 Kafka，读取输入文件并将每行发送到给定的 Kafka 主题。
- **FileStreamSinkConnector** 将数据从 Kafka 流传输到 worker 文件系统，读取 Kafka 主题的消息并在输出文件中写入一行。

以下源连接器图显示了源连接器的进程流，该连接器从外部数据系统中流传输记录。Kafka Connect 集群可以同时运行源和接收器连接器。Worker 在集群的分布式模式下运行。worker 可以为多个连接器实例运行一个或多个任务。

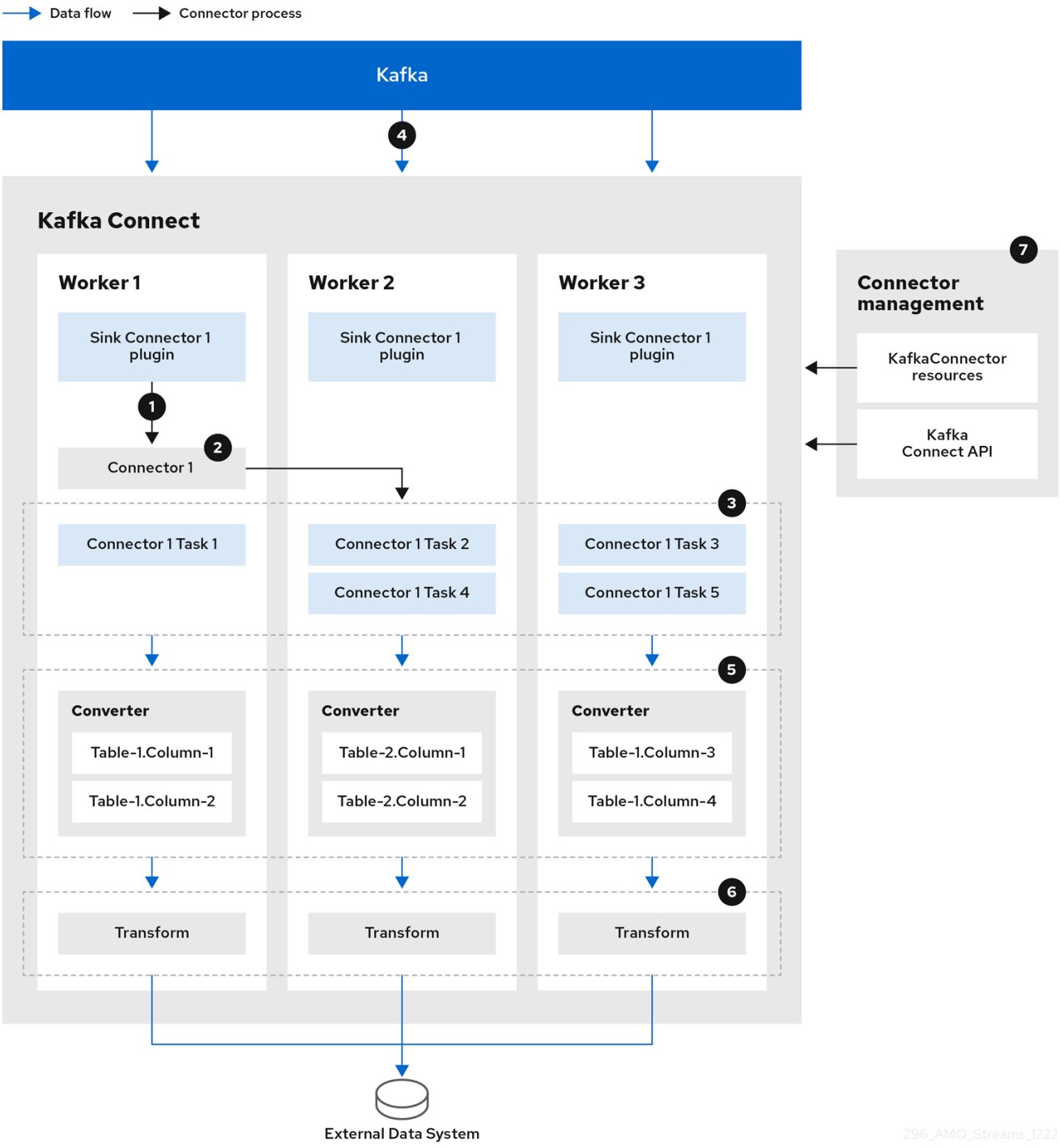
Source 连接器流数据到 Kafka



1. 插件为源连接器提供实施工件
2. 单个 worker 启动源连接器实例
3. 源连接器创建流数据的任务
4. 任务并行运行，以轮询外部数据系统和返回记录
5. 转换会调整记录，如过滤或重新标记它们
6. 转换器将记录置于适合 Kafka 的格式中
7. 源连接器使用 KafkaConnectors 或 Kafka Connect API 进行管理

以下接收器连接器图显示了将数据从 Kafka 流传输到外部数据系统时的流程流。

Kafka 中的接收器连接器流数据



1. 插件为接收器连接器提供实施工件
2. 单个 worker 启动接收器连接器实例
3. sink 连接器创建用于流传输数据的任务
4. 任务并行运行，以轮询 Kafka 和返回记录
5. 转换器将记录置于适合外部数据系统的格式
6. 转换会调整记录，如过滤或重新标记它们
7. sink 连接器使用 KafkaConnectors 或 Kafka Connect API 进行管理

4.1.2. 任务

Kafka Connect 运行时编排的数据传输被分成并行运行的任务。使用连接器实例提供的配置启动任务。Kafka Connect 将任务配置分发给 worker，从而实例化和执行任务。

- 源连接器任务轮询外部数据系统，并返回 worker 发送到 Kafka 代理的记录列表。
- 接收器连接器任务从 worker 接收 Kafka 记录，以写入外部数据系统。

对于接收器连接器，所创建的任务数量与被消耗的分区数量相关。对于源连接器，源数据如何分区由连接器定义。您可以通过在连接器配置中设置 **tasksMax** 来控制可以并行运行的最大任务数量。连接器可能会创建比最大设置更少的任务。例如，如果无法将源数据分成多个分区，则连接器可能会创建较少的任务。

注意

在 Kafka Connect 中，*分区*可能意味着一个主题分区，或外部系统中的*数据分片*。

4.1.3. Worker

worker 使用部署到 Kafka Connect 集群的连接器配置。配置存储在 Kafka Connect 使用的内部 Kafka 主题中。工作程序还可运行连接器及其任务。

Kafka Connect 集群包含一组具有相同 **group.id** 的 worker。ID 用于在 Kafka 中标识集群。该 ID 通过 **KafkaConnect** 资源在 worker 配置中分配。worker 配置还指定内部 Kafka Connect 主题的名称。主题存储连接器配置、偏移和状态信息。这些主题的组 ID 和名称对于 Kafka Connect 集群也必须是唯一的。

为 worker 分配一个或多个连接器实例和任务。以分布式部署的 Kafka Connect 具有容错和可扩展功能。如果 worker pod 失败，则会将其运行的任务重新分配给活跃 worker。您可以通过配置 **KafkaConnect** 资源中的 **replicas** 属性来添加到一组 worker pod。

4.1.4. 转换

Kafka Connect 可转换外部数据。单消息转换会将信息改为适合目标目的地的格式。例如，转换可能会插入或重命名字段。转换也可以过滤和路由数据。插件包含 worker 执行一个或多个转换所需的实施。

- 源连接器在将数据转换为 Kafka 支持的格式之前应用转换。
- sink 连接器会在将数据转换为适合外部数据系统的格式后进行转换。

一个转换包括封装在 JAR 文件中的一组 Java 类文件，用于包含在连接器插件中。Kafka Connect 提供了一组标准转换，但您也可以自行创建。

4.1.5. 转换器

当 worker 收到数据时，会使用转换器将数据转换为适当的格式。您可以在 **KafkaConnect** 资源中的 **worker** 配置中指定转换器。

Kafka Connect 可以从 Kafka 支持的格式（如 JSON 或 Avro）转换数据。它还支持构造数据的架构。如果您不将数据转换为结构化格式，则不需要启用架构。

注意

您还可以为特定连接器指定转换器，以覆盖适用于所有 worker 的一般 Kafka Connect worker 配置。

其他资源

- Apache Kafka 文档
- worker 的 Kafka 连接配置
- 使用 MirrorMaker 2 在 Kafka 集群间同步数据

第 5 章 KAFKA BRIDGE 接口

Kafka Bridge 提供了一个 RESTful 接口，它允许基于 HTTP 的客户端与 Kafka 集群交互。 它为客户端提供与 Apache Kafka 的流的 HTTP API 连接的优点，以便客户端生成和使用消息，而无需使用原生 Kafka 协议。

API 有两个主要资源 - **消费者（consumer）**和**主题（topic）** - 通过端点公开并可访问，以便与 Kafka 集群中的消费者和制作者交互。资源只与 Kafka Bridge 相关，而不是直接连接到 Kafka 的用户和制作者。

5.1. HTTP 请求

Kafka Bridge 支持对 Kafka 集群的 HTTP 请求，使用以下方法执行以下操作：

- 发送消息到主题。
- 从主题检索消息。
- 检索主题的分区列表。
- 创建和删除用户。
- 订阅消费者到主题，以便他们开始从这些主题接收信息。
- 检索消费者订阅的主题列表。
- 取消订阅消费者的主题.
- 为消费者分配分区。
- 提交使用者偏移列表。
- 定位分区，以便消费者开始收到来自第一或最后一个偏移位置的信息，或给定的偏移位置。

该方法提供 JSON 响应和 HTTP 响应代码错误处理。消息可以使用 JSON 或二进制格式发送。

其他资源

- 要查看 API 文档，包括请求和响应示例，请参阅使用 Apache Kafka Bridge 的 Streams 。

5.2. KAFKA BRIDGE 支持的客户端

您可以使用 Kafka Bridge 将*内部*和*外部* HTTP 客户端应用程序与 Kafka 集群集成。

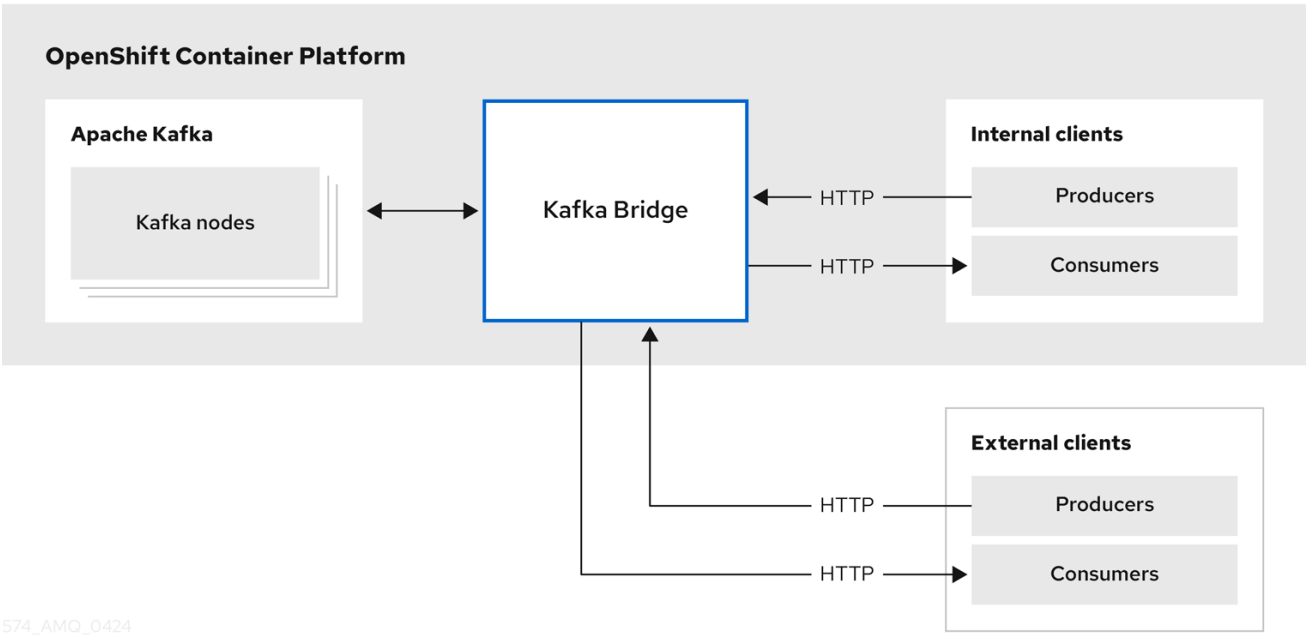
内部客户端

内部客户端是在与 Kafka Bridge 本身 *相同的* OpenShift 集群中运行的容器 HTTP 客户端。内部客户端可以访问 KafkaBridge 自定义资源中定义的主机和端口上的 **Kafka Bridge**。

外部客户端

外部客户端是 Kafka Bridge 在 其中部署和运行的 OpenShift 集群*外部*运行的 HTTP 客户端。外部客户端可以通过 OpenShift Route、负载均衡器服务或使用 Ingress 访问 Kafka 网桥。

HTTP 内部和外部客户端集成


OpenShift Container Platform
Apache Kafka
Kafka nodes
Kafka Bridge
HTTP
HTTP
Internal clients
Producers
Consumers
External clients
HTTP
HTTP
Producers
Consumers
574_AMQ_0424

第 6 章 APACHE KAFKA OPERATOR 的流

Operator 是一种打包、部署和管理 OpenShift 应用程序的方法。它们提供了一种扩展 Kubernetes API 的方法，并简化与特定应用程序关联的管理任务。

Apache Kafka operator 的流支持与 Kafka 部署相关的任务。Apache Kafka 自定义资源的流提供部署配置。这包括 Kafka 集群、主题、用户和其他组件的配置。利用自定义资源配置，Apache Kafka operator 的 Streams 在 OpenShift 环境中创建、配置和管理 Kafka 组件。使用 Operator 可减少人工干预并简化在 OpenShift 集群中管理 Kafka 的过程。

Apache Kafka 的流提供以下操作器来管理在 OpenShift 集群中运行的 Kafka 集群。

Cluster Operator

部署和管理 Apache Kafka 集群、Kafka Connect、Kafka MirrorMaker、Kafka Bridge、Kafka Exporter、Cruise Control 和 Entity Operator

Entity Operator

包括 Topic Operator 和 User Operator

Topic Operator

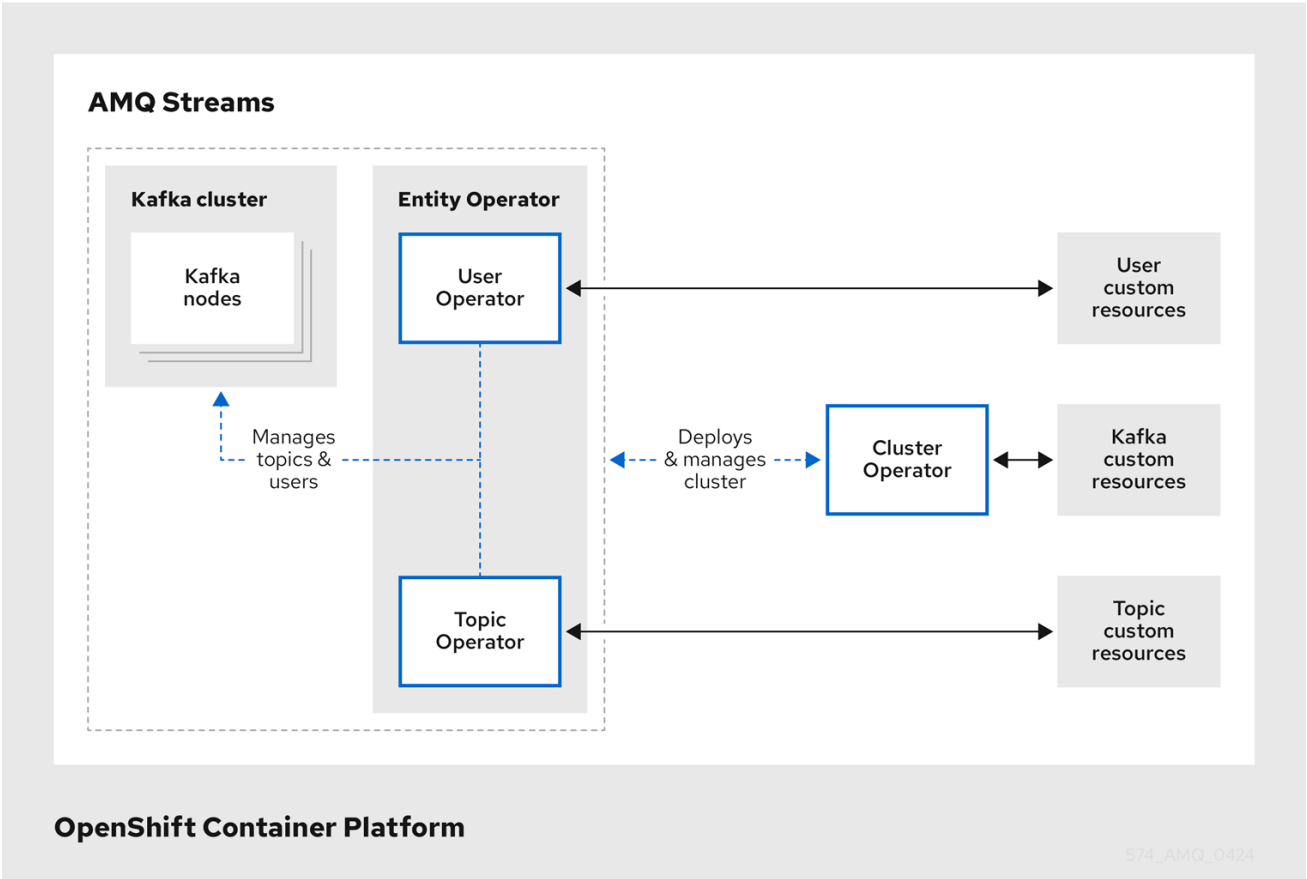
管理 Kafka 主题

User Operator

管理 Kafka 用户

Cluster Operator 可以在与 Kafka 集群同时部署 Topic Operator 和 User Operator 作为 **Entity Operator** 配置的一部分。

Streams for Apache Kafka 架构中的 Operator



6.1. CLUSTER OPERATOR

Apache Kafka 的 Streams 使用 Cluster Operator 来部署和管理集群。默认情况下，当您为 Apache Kafka 部署流时，会部署一个 Cluster Operator 副本。您可以使用领导选举机制添加副本，以便在出现问题时有其他 Cluster Operator 处于待机状态。

Cluster Operator 管理以下 Kafka 组件的集群：

- Kafka （包括 ZooKeeper、实体 Operator、Kafka Exporter 和 Cruise Control）
- Kafka Connect
- Kafka MirrorMaker
- Kafka Bridge

集群使用自定义资源部署。

例如，要部署 Kafka 集群：

- 在 OpenShift 集群中创建带有集群配置的 **Kafka** 资源。
- Cluster Operator 根据 Kafka 资源中声明的内容部署对应的 **Kafka** 集群。

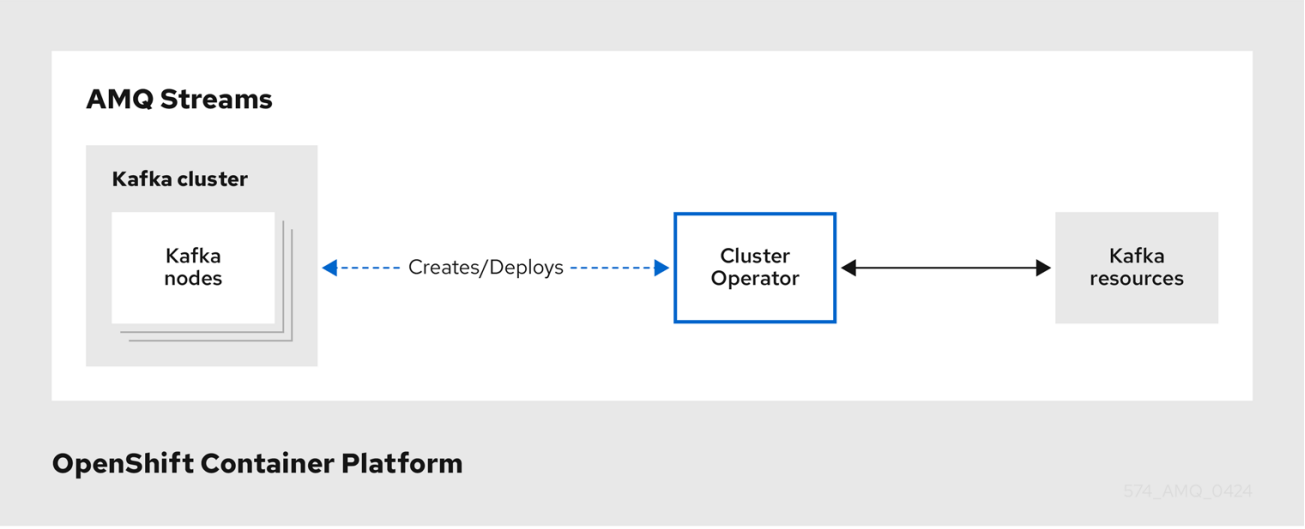
Cluster Operator 还可通过配置 **Kafka** 资源来部署 Apache Kafka operator 的以下流：

- 主题 Operator 通过 **KafkaTopic** 自定义资源提供 operator 风格主题管理
- 用户 Operator 通过 **KafkaUser** 自定义资源提供 operator 风格的用户管理

在部署时实体 Operator 中的主题 Operator 和用户 Operator 功能。

您可以使用 Cluster Operator 部署 Apache Kafka Drain Cleaner 的 Streams，以帮助 pod 驱除。通过为 Apache Kafka Drain Cleaner 部署流，您可以使用 Cluster Operator 来移动 Kafka pod 而不是 OpenShift。Apache Kafka Drain Cleaner 的流使用滚动更新注解来注解被驱除的 pod。该注解告知 Cluster Operator 执行滚动更新。

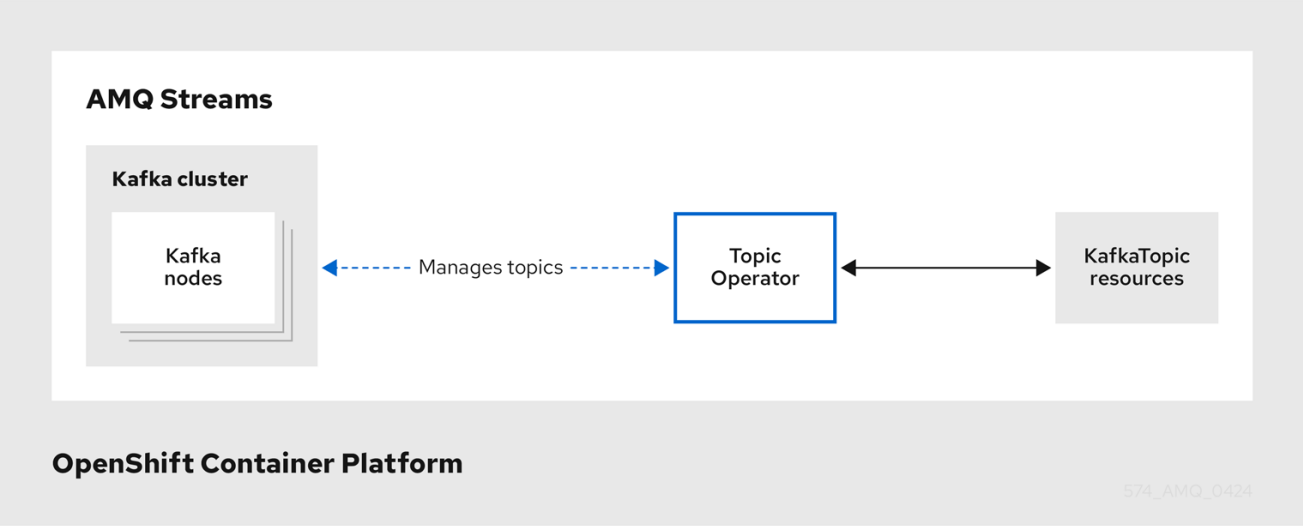
Cluster Operator 构架示例



6.2. TOPIC OPERATOR

主题 Operator 提供了通过 **KafkaTopic** 资源管理 Kafka 集群中的主题的方法。

主题 Operator 的架构示例



主题 Operator 通过监视 describe Kafka 主题的 **KafkaTopic** 资源来管理 Kafka 主题，并确保它们在 Kafka 集群中正确配置。

当创建、删除或更改 **KafkaTopic** 时，主题 Operator 对 Kafka 主题执行对应的操作。

您可以将 **KafkaTopic** 声明为应用程序部署的一部分，主题 Operator 会为您管理 Kafka 主题。

主题 Operator 以以下模式运行：

单向模式

单向模式意味着主题 Operator 通过 **KafkaTopic** 资源单独管理主题。这个模式不需要 ZooKeeper，并在 KRaft 模式中与 Apache Kafka 使用 Streams 兼容。

双向模式

双向模式意味着 Topic Operator 可以协调对 Kafka 集群的 **KafkaTopic** 资源的更改。这意味着，您可以通过 **KafkaTopic** 资源或在 Kafka 中直接更新主题，主题 Operator 可确保更新这两个源以反映更改。这个模式需要 ZooKeeper 用于集群管理。

主题 Operator 会维护有关*主题存储*中的每个主题的信息，该存储会持续与 Kafka 主题或 OpenShift **KafkaTopic** 自定义资源的更新同步。应用到本地内存主题存储的操作更新将保留到磁盘上的备份主题。

6.3. USER OPERATOR

User Operator 提供了通过 **KafkaUser** 资源管理 Kafka 集群中的用户的方法。

User Operator 通过监视用于描述 Kafka 用户的 **Kafka** 资源来管理 Kafka 集群的 Kafka 用户，并确保它们在 Kafka 集群中正确配置。

创建、删除或更改 **KafkaUser** 时，User Operator 对 Kafka 用户执行对应的操作。

您可以将 **KafkaUser** 资源声明为应用程序部署的一部分，User Operator 会为您管理 Kafka 用户。您可以为用户指定身份验证和授权机制。您还可以配置控制使用 Kafka 资源的 *用户配额* 以确保用户不单调访问代理。

创建用户时，用户凭证会在 **Secret** 中创建。您的应用需要使用用户及其凭据来进行身份验证，才能生成或消耗消息。

除了管理用于身份验证的凭据外，User Operator 还通过在 **KafkaUser** 声明中包含用户访问权限的描述来管理授权规则。

6.4. APACHE KAFKA OPERATOR 的 STREAMS 中的功能门

Apache Kafka operator 的流使用功能门来启用或禁用特定功能和功能。启用功能门会更改关联的 Operator 的行为，为 Apache Kafka 部署引入对应的功能。

功能门在 Operator 配置中设置，有三个成熟度阶段： alpha、beta 或 graduated。graduated 功能门已正式发布(GA)，并被永久启用的功能。

如需更多信息，请参阅 功能门。

第 7 章 KAFKA 配置

使用 Streams for Apache Kafka 将 Kafka 组件部署到 OpenShift 集群上，强烈建议您使用自定义资源进行配置。这些资源作为自定义资源定义(CRD)引入的 API 实例创建，后者扩展 OpenShift 资源。

CRD 作为描述 OpenShift 集群中自定义资源的配置说明，为部署中使用的每个 Kafka 组件提供 Apache Kafka 的流，以及用户和主题。CRD 和自定义资源被定义为 YAML 文件。Apache Kafka 发行版的 Streams 提供了 YAML 文件示例。

CRD 还允许 Apache Kafka 资源的 Streams 从原生 OpenShift 功能中受益，如 CLI 访问和配置验证。

在本节中，我们介绍了如何通过自定义资源配置 Kafka 组件，从常见配置点开始，然后了解与组件相关的重要配置注意事项。

Apache Kafka 的 Streams 提供了 示例配置文件，可在为部署构建自己的 Kafka 组件配置时作为起点。

7.1. 自定义资源

在安装 CRD 中添加新的自定义资源类型后，您可以根据规格创建资源实例。

Apache Kafka 组件的 Streams 自定义资源具有通用配置属性，这些属性在 **spec** 下定义。

在 Kafka 主题自定义资源中的这种片段中，**apiVersion** 和 **kind** 属性标识关联的 CRD。**spec** 属性显示定义主题的分区和副本数的配置。

Kafka 主题自定义资源

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: my-topic
  labels:
    strimzi.io/cluster: my-cluster
spec:
  partitions: 1
  replicas: 1
  # ...
```

有很多额外的配置选项可以合并到 YAML 定义中，一些常见和特定于特定组件的配置选项。

其他资源

- 使用 CustomResourceDefinitions 扩展 Kubernetes API

7.2. 常见配置

此处介绍了一些与资源通用的配置选项。在适当时，Security 和 metrics collection 可能也会采用。

Bootstrap 服务器

Bootstrap 服务器用于主机/端口连接到一个 Kafka 集群用于：

- Kafka Connect
- Kafka Bridge

- Kafka MirrorMaker 制作者和消费者

CPU 和内存资源

您可以为组件请求 CPU 和内存资源。限制（limits）指定给定容器可消耗的最大资源。Topic Operator 和 User Operator 的资源请求和限值在 **Kafka** 资源中设置。

日志记录

您为组件定义日志级别。可以使用配置映射直接（在线）或外部定义日志记录。

健康检查（Healthcheck）

HealthCheck 配置引入了*存活度*和*就绪度*探测，以知道何时重启容器（持续）以及容器是否可以接受流量（就绪状态）。

JVM 选项

JVM 选项提供最大和最小内存分配，以根据所运行的平台优化组件性能。

Pod 调度

Pod 调度使用 *关联性/反关联性*规则，以确定在将 pod 调度到某个节点的情况下。

显示常见配置的 YAML 示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-cluster
spec:
  # ...
  bootstrapServers: my-cluster-kafka-bootstrap:9092
  resources:
    requests:
      cpu: 12
      memory: 64Gi
    limits:
      cpu: 12
      memory: 64Gi
  logging:
    type: inline
    loggers:
      connect.root.logger.level: INFO
  readinessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 5
  livenessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 5
  jvmOptions:
    "-Xmx": "2g"
    "-Xms": "2g"
  template:
    pod:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
```

```
            - key: node-type
              operator: In
              values:
                - fast-network
# ...
```

7.3. KAFKA 集群配置

kafka 集群包含一个或多个代理。要使生产者和消费者能够访问代理中的主题，Kafka 配置必须定义如何存储数据，以及如何访问数据。您可以将 Kafka 集群配置为使用多个代理节点在机架之间运行。

7.3.1. Storage

Apache Kafka 的流支持以下存储配置选项来管理 Kafka 和 ZooKeeper 数据：

Ephemeral (推荐只在开发时使用)

临时存储在实例生命周期内存储数据。实例重启时数据会丢失。

持久性

持久性存储与独立于实例生命周期的长期数据存储相关。

JBOD (Just a Bunch Disks，仅适用于 Kafka)

JBOD 允许您使用多个磁盘将提交日志存储到每个代理中。

除了这些选项外，您还可以将 Apache Kafka 的 Streams 配置为使用分层存储。分层存储是 Kafka 中的 **早期访问** 功能，通过利用具有不同特征的并行使用存储类型为数据管理提供更大的灵活性。虽然其中一项基本存储选项必须配置为分层存储，但其支持组合，例如使用对象存储的块存储，以提高性能和可扩展性。

存储类型使用 **storage** 或 tier **Storage** 配置属性在自定义资源中指定：

- **存储配置** 类型：
  - **type: ephemeral**
  - **Type: persistent-claim** for persistent storage using Persistent Volume Claims (PVC)
  - **type: jbod** 用于 JBOD 存储
- **layeredStorage** 配置类型：
  - **类型：** 用于自定义分层存储的自定义

对于存储在磁盘上的 Kafka 和 ZooKeeper 数据，存储的文件系统格式必须是 XFS 或 EXT4。如果基础架构支持，可以增加现有 Kafka 集群使用的磁盘容量。

7.3.2. 监听器

侦听器配置客户端如何连接到 Kafka 集群。

通过在 Kafka 集群中为每个监听程序指定唯一名称和端口，您可以配置多个监听程序。

支持以下监听程序类型：

- 用于在 OpenShift 中访问的**内部监听程序**
- 用于在 OpenShift 外部进行访问的 **外部监听器**

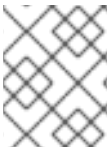
您可以为监听器启用 TLS 加密，并配置身份验证。

内部监听程序通过指定一个 **internal** 类型来公开 Kafka：

- 在同一 OpenShift 集群内连接的**内部**
- **cluster-ip** 使用针对每个代理的 **ClusterIP** 服务公开 Kafka

外部监听程序通过指定一个外部 类**型** 来公开 Kafka：

- 使用 OpenShift 路由和默认 HAProxy 路由器的**路由**
- **LoadBalancer** 使用负载均衡器服务
- **NodePort** 使用 OpenShift 节点上的端口
- **Ingress** 使用 OpenShift *Ingress* 和 Ingress NGINX Controller for Kubernetes



注意

使用 **cluster-ip** 类型可以添加您自己的访问机制。例如，您可以将监听程序与自定义 Ingress 控制器或 OpenShift 网关 API 搭配使用。

如果您使用 OAuth 2.0 进行基于令牌的身份验证，您可以将监听程序配置为使用授权服务器。

7.3.3. 机架感知

机架代表数据中心，或数据中心中的机架，或可用性区域。配置机架感知，以在机架之间分发 Kafka 代理 pod 和主题副本。使用 **rack** 属性启用机架感知，以指定 **topologyKey**。**topologyKey** 是分配给 OpenShift worker 节点的标签的名称，用于标识机架。Apache Kafka 的流会为每个 Kafka 代理分配一个机架 ID。Kafka 代理使用 ID 在机架间分布分区副本。您还可以指定 **RackAwareReplicaSelector** 选择器插件，以用于机架感知。该插件与代理和消费者的机架 ID 匹配，以便消息从最接近的副本消耗。要使用插件，消费者还必须启用机架感知。您可以在 Kafka Connect、M MirrorMaker 2 和 Kafka Bridge 中启用机架感知。

7.3.4. 显示 Kafka 配置的 YAML 示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
      - name: external1
        port: 9094
        type: route
        tls: true
```

```
      authentication:
        type: tls
  # ...
  storage:
    type: persistent-claim
    size: 10000Gi
  # ...
  rack:
    topologyKey: topology.kubernetes.io/zone
  config:
    replica.selector.class: org.apache.kafka.common.replica.RackAwareReplicaSelector
  # ...
```

7.4. KAFKA 节点池配置

节点池指的是 Kafka 集群中不同的 Kafka 节点组。通过使用节点池，节点可以在同一 Kafka 集群中有不同的配置。在节点池中没有指定的配置选项从 Kafka 配置继承。

您可以使用一个或多个节点池部署 Kafka 集群。节点池配置包括强制和可选设置。副本、角色和存储的配置是必需的。

如果使用 KRaft 模式，您可以指定节点池中所有节点作为代理、控制器或两者运行的角色。控制器和双角色特定于具体的 KRaft。如果您使用 ZooKeeper 用于集群管理的 Kafka 集群，则只能使用使用代理角色配置的节点池。

显示节点池配置的 YAML 示例

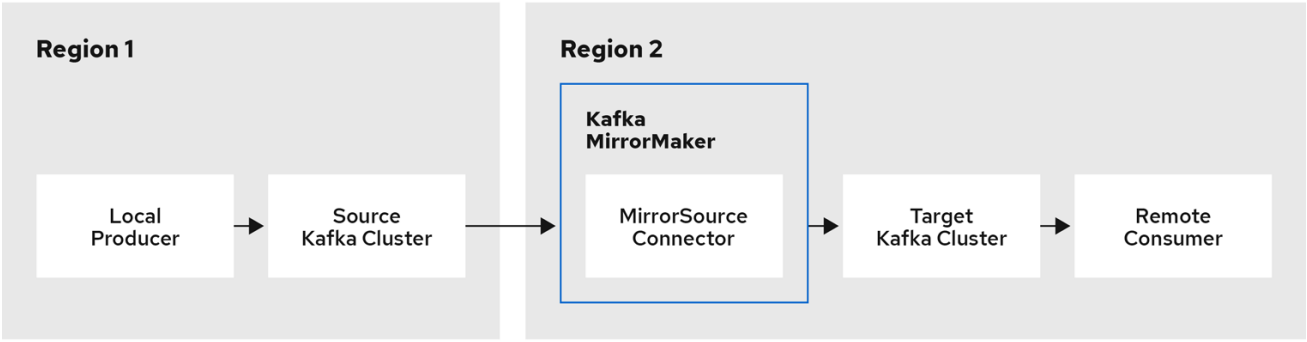
```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaNodePool
metadata:
  name: pool-a
  labels:
    strimzi.io/cluster: my-cluster
spec:
  replicas: 3
  roles:
    - broker
  storage:
    type: jbod
    volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
```

7.5. KAFKA MIRRORMAKER 2 配置

Kafka MirrorMaker 2 在两个或多个活跃 Kafka 集群之间复制数据，并在数据中心之间复制数据。要设置 MirrorMaker 2，源和目标（目标） Kafka 集群必须正在运行。

将数据从源集群*镜像 (mirror)*到目标集群的过程是异步的。每个 MirrorMaker 2 实例将数据从一个源集群镜像到一个目标集群。您可以使用多个 MirrorMaker 2 实例在任意数量的集群间镜像数据。

图 7.1. 在两个集群间复制



MirrorMaker 2 使用源和目标集群配置，如下所示：

- 用于从源集群消耗数据的源集群配置
- 将数据输出到目标集群的目标集群配置

主题和消费者组复制以逗号分隔的列表或正则表达式模式指定。

显示 MirrorMaker 2 配置的 YAML 示例

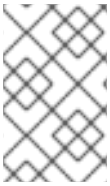
```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
  spec:
    version: 3.7.0
    connectCluster: "my-cluster-target"
    clusters:
    - alias: "my-cluster-source"
      bootstrapServers: my-cluster-source-kafka-bootstrap:9092
    - alias: "my-cluster-target"
      bootstrapServers: my-cluster-target-kafka-bootstrap:9092
    mirrors:
    - sourceCluster: "my-cluster-source"
      targetCluster: "my-cluster-target"
      sourceConnector: {}
    topicsPattern: ".*"
    groupsPattern: "group1|group2|group3"
```

7.6. KAFKA MIRRORMAKER 配置

Kafka MirrorMaker （也称为 MirrorMaker 1）使用制作者和消费者在集群间复制数据，如下所示：

- 消费者使用来自源集群的数据
- 生成者将数据输出到目标集群

消费者和制作者配置包括任何所需的身份验证和加密设置。**include** 属性定义要从源镜像到目标集群的主题。



注意

MirrorMaker 在 Kafka 3.0.0 中已弃用，并将在 Kafka 4.0.0 中删除。因此，用于部署 MirrorMaker 的 Apache Kafka **KafkaMirrorMaker** 自定义资源的 Streams 已被弃用。当使用 Kafka 4.0.0 时，**KafkaMirrorMaker** 资源将从 Apache Kafka 的 Streams 中删除。

关键的消费者配置

消费者组标识符

MirrorMaker consumer 的使用者组 ID，以便消耗的消息被分配给消费者组。

消费者流的数量

一个值，用于决定消费者组中的消费者数量，以并行使用消息。

偏移提交间隔

一个偏移的提交间隔，用于设置消耗和提交消息之间的时间。

关键的生成者配置

发送失败的取消选项

您可以定义消息发送失败是否忽略或 mirrorMaker 被终止并重新创建。

显示 MirrorMaker 配置的 YAML 示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker
metadata:
  name: my-mirror-maker
spec:
  # ...
  consumer:
    bootstrapServers: my-source-cluster-kafka-bootstrap:9092
    groupId: "my-group"
    numStreams: 2
    offsetCommitInterval: 120000
    # ...
  producer:
    # ...
    abortOnSendFailure: false
    # ...
  include: "my-topic|other-topic"
  # ...
```

7.7. KAFKA CONNECT 配置

使用 Apache Kafka 的 **KafkaConnect** 资源的 Streams 来快速轻松地创建新的 Kafka Connect 集群。

当使用 KafkaConnect 资源部署 **Kafka Connect** 时，您可以指定 bootstrap 服务器地址（在 **spec.bootstrapServers**中）以连接到 Kafka 集群。当服务器停机时，您可以指定多个地址。您还指定身份验证凭据和 TLS 客户端证书，以建立安全连接。

注意

Kafka 集群不需要由 Streams for Apache Kafka 管理，或部署到 OpenShift 集群。

您还可以使用 **KafkaConnect** 资源来指定以下内容：

- 构建包含插件的容器镜像的插件配置，以建立连接
- 属于 Kafka Connect 集群的 worker pod 配置
- 启用使用 **KafkaConnector** 资源管理插件的注解

Cluster Operator 管理使用 KafkaConnector 资源部署的 **Kafka Connect** 集群，以及利用 **KafkaConnector** 资源创建的连接器。

插件配置

插件提供创建连接器实例的实施。当插件实例化时，会为连接特定类型的外部数据系统提供配置。插件提供一组或者多个 JAR 文件，该文件定义了连接器和任务实施，以连接到指定类型数据源。许多外部系统的插件可用于 Kafka 连接。您还可以创建自己的插件。

配置描述了要发送到 Kafka Connect 的源输入数据和目标输出数据。对于源连接器，外部源数据必须引用要存储消息的特定主题。插件也可以包含转换数据所需的库和文件。

Kafka Connect 部署可以有一个或多个插件，但每个插件只能有一个版本。

您可以创建自定义 Kafka Connect 镜像，其中包括您选择的插件。您可以通过两种方式创建镜像：

- 自动使用 Kafka Connect 配置
- 使用 Dockerfile 和 Kafka 容器镜像作为基础镜像手动

要自动创建容器镜像，您可以使用 KafkaConnect 资源的 **build** 属性指定要添加到 **Kafka Connect** 集群中的插件。Apache Kafka 的 Streams 会自动下载插件工件，并将插件工件添加到新容器镜像中。

插件配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  # ...
  build: 1
    output: 2
      type: docker
      image: my-registry.io/my-org/my-connect-cluster:latest
      pushSecret: my-registry-credentials
    plugins: 3
      - name: my-connector
        artifacts:
          - type: tgz
            url: https://<plugin_download_location>.tgz
            sha512sum: <checksum_to_verify_the_plugin>
      # ...
  # ...
```

1 构建自动使用插件构建容器镜像的配置属性。

2 推送新镜像的容器 registry 的配置。**output** 属性描述镜像的类型和名称，以及包含访问容器 registry 所需的凭证的 secret 名称。

3 要添加到新容器镜像的插件及其构件列表。**plugins** 属性描述工件的类型以及下载工件的 URL。每个插件必须配置至少一个工件。另外，您可以指定 SHA-512 校验和来验证工件，然后再解包。

如果使用 Dockerfile 构建镜像，您可以使用 Apache Kafka 的最新容器镜像的 Streams 作为基础镜像来添加插件配置文件。

显示手动添加插件配置示例

```
FROM registry.redhat.io/amq-streams/kafka-37-rhel8:2.7.0
USER root:root
COPY ./my-plugins/ /opt/kafka/plugins/
USER 1001
```

worker 的 Kafka Connect 集群配置

您可以在 **KafkaConnect** 资源的 **config** 属性中指定 worker 的配置。

分布式 Kafka Connect 集群有一个组 ID 和一组内部配置主题。

- **group.id**
- **offset.storage.topic**
- **config.storage.topic**
- **status.storage.topic**

Kafka Connect 集群默认使用这些属性的值配置。Kafka Connect 集群无法共享组 ID 或主题名称，因为它将创建错误。如果使用多个不同的 Kafka Connect 集群，则每个创建的 Kafka Connect 集群的 worker 必须是唯一的。

每个 Kafka Connect 集群使用的连接器名称也必须是唯一的。

在以下示例中，指定了 JSON 转换器。Kafka Connect 使用的内部 Kafka 主题设置了复制因素。对于生产环境，至少应有 3 个。在创建主题后更改复制因素将无效。

worker 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
# ...
spec:
  config:
    # ...
    group.id: my-connect-cluster 1
    offset.storage.topic: my-connect-cluster-offsets 2
    config.storage.topic: my-connect-cluster-configs 3
    status.storage.topic: my-connect-cluster-status 4
    key.converter: org.apache.kafka.connect.json.JsonConverter 5
    value.converter: org.apache.kafka.connect.json.JsonConverter 6
    key.converter.schemas.enable: true 7
    value.converter.schemas.enable: true 8
```

```
    config.storage.replication.factor: 3 9
    offset.storage.replication.factor: 3 10
    status.storage.replication.factor: 3 11
  # ...
```

1. Kafka 中的 Kafka Connect 集群 ID。每个 Kafka Connect 集群都必须是唯一的。
2. 存储连接器偏移的 Kafka 主题。每个 Kafka Connect 集群都必须是唯一的。
3. 存储连接器和任务状态配置的 Kafka 主题。每个 Kafka Connect 集群都必须是唯一的。
4. 存储连接器和任务状态更新的 Kafka 主题。每个 Kafka Connect 集群都必须是唯一的。
5. 转换程序，将消息密钥转换为 Kafka 中存储的 JSON 格式。
6. 转换程序，将消息值转换为 Kafka 中存储的 JSON 格式。
7. 为将消息键转换为结构化 JSON 格式的 schema。
8. 为将消息值转换为结构化 JSON 格式的 schema。
9. 存储连接器偏移的 Kafka 主题的复制因素。
10. 存储连接器和任务状态配置的 Kafka 主题的复制因素。
11. 存储连接器和任务状态更新的 Kafka 主题的复制因素。

连接器的 KafkaConnector 管理

在将插件添加到用于部署中的 worker pod 的容器镜像后，您可以使用 Streams for Apache Kafka 的 **KafkaConnector** 自定义资源或 Kafka Connect API 来管理连接器实例。您还可以使用这些选项创建新的连接器实例。

KafkaConnector 资源提供了一种 OpenShift 原生的方法来管理 Cluster Operator 连接器。要使用 **KafkaConnector** 资源管理连接器，您必须在 **KafkaConnect** 自定义资源中指定注解。

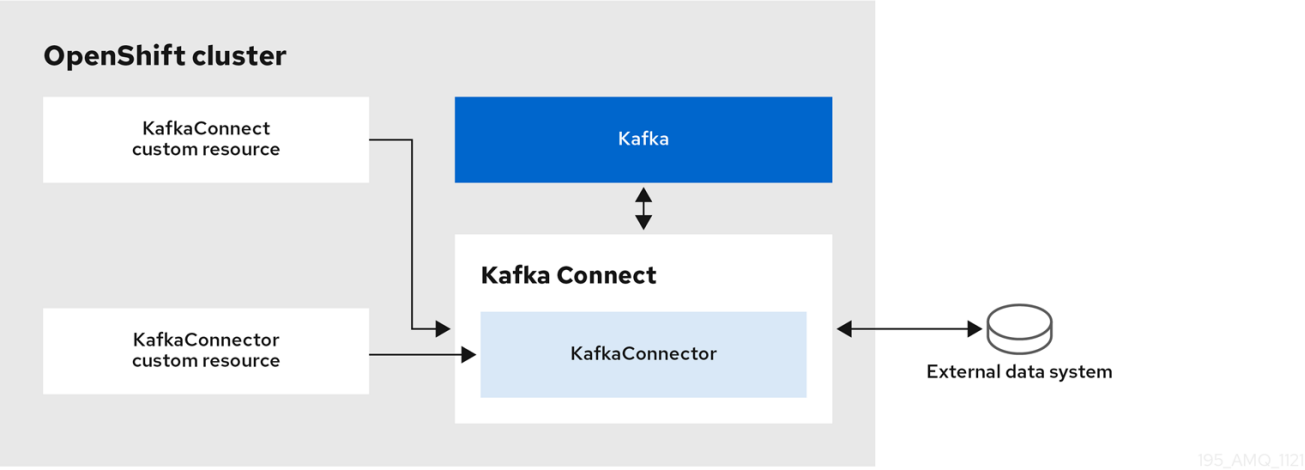
启用 KafkaConnectors 的注解

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true"
  # ...
```

将 **use-connector-resources** 设置为 **true** 可启用 KafkaConnectors 创建、删除和重新配置连接器。

如果在 **KafkaConnect** 配置中启用了 **use-connector-resources**，则必须使用 **KafkaConnector** 资源来定义和管理连接器。**KafkaConnector** 资源被配置为连接到外部系统。它们部署到与 Kafka Connect 集群和与外部数据系统交互的 Kafka 集群相同的 OpenShift 集群。

Kafka 组件在同一个 OpenShift 集群中包含



配置指定连接器实例如何连接到外部数据系统，包括任何身份验证。您还需要陈述要监视的数据。对于源连接器，您可能会在配置中提供数据库名称。您还可以通过指定目标主题名称指定数据在 Kafka 中的位置。

使用 **tasksMax** 指定最大任务数。例如，带有 **tasksMax: 2** 的源连接器可能会将源数据导入两个任务。

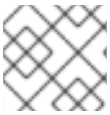
KafkaConnector 源连接器配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-source-connector 1
  labels:
    strimzi.io/cluster: my-connect-cluster 2
spec:
  class: org.apache.kafka.connect.file.FileStreamSourceConnector 3
  tasksMax: 2 4
  autoRestart: 5
    enabled: true
  config: 6
    file: "/opt/kafka/LICENSE" 7
    topic: my-topic 8
    # ...
```

1. **KafkaConnector** 资源的名称，用作连接器的名称。使用对 OpenShift 资源有效的任何名称。
2. 在其中创建连接器实例的 Kafka Connect 集群的名称。连接器必须部署到它们所链接的 Kafka Connect 集群相同的命名空间中。
3. 连接器类的全名。这应该存在于 Kafka Connect 集群使用的镜像中。
4. 连接器可创建的最大 Kafka Connect 任务数量。
5. 启用自动重启失败的连接器和任务。默认情况下，重启数量是无限的，但您可以使用 **maxRestarts** 属性设置自动重启次数的最大值。
6. 连接器配置 作为键值对。
7. 外部数据文件的位置。在本例中，我们将 **FileStreamSourceConnector** 配置为从 **/opt/kafka/LICENSE** 文件中读取。

/opt/kafka/LICENSE 文件中读取。

8 将源数据发布到的 Kafka 主题。



注意

您可以 从外部来源（如 OpenShift Secret 或 ConfigMap）为连接器加载 机密配置值。

Kafka Connect API

使用 Kafka Connect REST API 作为使用 **KafkaConnector** 资源管理连接器的替代选择。Kafka Connect REST API 作为一个运行在 ***<connect_cluster_name>*-connect-api:8083** 的服务其中 *<connect_cluster_name>* 是 Kafka Connect 集群的名称。

您可以将连接器配置添加为 JSON 对象。

添加连接器配置的 curl 请求示例

```
curl -X POST \
  http://my-connect-cluster-connect-api:8083/connectors \
  -H 'Content-Type: application/json' \
  -d '{ "name": "my-source-connector",
    "config":
    {
      "connector.class":"org.apache.kafka.connect.file.FileStreamSourceConnector",
      "file": "/opt/kafka/LICENSE",
      "topic":"my-topic",
      "tasksMax": "4",
      "type": "source"
    }
}'
```

如果启用了 KafkaConnectors，Cluster Operator 将恢复使用 Kafka Connect REST API 进行的手动更改。

REST API 支持的操作在 Apache Kafka Connect API 文档中 进行了描述。

注意

您可以在 OpenShift 外部公开 Kafka Connect API 服务。为此，您可以创建一个使用连接机制来提供访问的服务，如入口或路由。我们建议使用，因为连接是不安全的。

其他资源

- Kafka Connect 配置选项
- 多个实例的 Kafka 连接配置
- 使用插件扩展 Kafka 连接
- 使用 Apache Kafka 的 Streams 自动创建新容器镜像
- 从 Kafka Connect 基础镜像创建 Docker 镜像
- 构建架构参考

- 源和接收器连接器配置选项
- 从外部来源加载配置值

7.8. KAFKA BRIDGE 配置

Kafka Bridge 配置需要它连接到的 Kafka 集群的 bootstrap 服务器规格，以及所需的加密和身份验证选项。

Kafka Bridge 消费者和生成者配置是标准的，如 Apache Kafka configuration documentation for consumers 和 Apache Kafka configuration documentation for producers 所述。

与 HTTP 相关的配置选项设置服务器侦听的端口连接。

CORS

Kafka Bridge 支持使用 Cross-Origin Resource Sharing (CORS)。CORS 是一种 HTTP 机制，它允许浏览器从多个来源访问选定资源，例如，不同域中的资源。如果您选择使用 CORS，可以定义一个允许的资源来源列表，并通过 HTTP 方法通过 Kafka Bridge 与 Kafka 集群交互。列表在 Kafka Bridge 配置的 **http** 规格中定义。

CORS 允许在不同域中的源之间的 *简单* 和 *preflighted* 请求。

- 简单的请求是一个 HTTP 请求，必须在其标头中定义允许的源。
- preflighted 请求在实际请求之前发送一个初始 OPTIONS HTTP 请求，以检查允许原始和方法。

显示 Kafka Bridge 配置的 YAML 示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  bootstrapServers: my-cluster-kafka:9092
  http:
    port: 8080
    cors:
      allowedOrigins: "https://strimzi.io"
      allowedMethods: "GET,POST,PUT,DELETE,OPTIONS,PATCH"
  consumer:
    config:
      auto.offset.reset: earliest
  producer:
    config:
      delivery.timeout.ms: 300000
  # ...
```

其他资源

- 获取 CORS 规格

第 8 章 保护 KAFKA

一个安全部署 Apache Kafka 的 Streams 可能会包括以下一个或多个安全措施：

- 数据交换加密
- 证明身份的身份验证
- 允许或拒绝用户执行操作的授权
- 在启用了 FIPS 的 OpenShift 集群上运行 Apache Kafka 的流，以确保数据安全性和系统互操作性

8.1. ENCRYPTION

Apache Kafka 的流支持传输层安全(TLS)，它是一个加密通信的协议。

在通信是始终进行加密：

- Kafka 代理
- Zookeeper 节点
- Kafka 代理和 ZooKeeper 节点
- Operator 和 Kafka 代理
- operator 和 ZooKeeper 节点
- Kafka Exporter

您还可以在 Kafka 代理和客户端之间配置 TLS 加密。为 Kafka 代理配置外部监听程序时，为外部客户端指定 TLS。

Apache Kafka 组件和 Kafka 客户端的流使用数字证书进行加密。Cluster Operator 设置证书以便在 Kafka 集群中启用加密。您可以提供自己的服务器证书，称为 *Kafka 侦听器证书*，用于 Kafka 客户端和 Kafka 代理之间的通信，以及集群间通信。

Apache Kafka 的流使用 *Secret* 将 mTLS 所需的证书和私钥存储 PEM 和 PKCS detailed 格式。

TLS CA （证书颁发机构）签发证书来验证组件的身份。Apache Kafka 的流会根据 CA 证书验证组件的证书。

- 针对 *集群 CA*验证 Apache Kafka 组件的流
- Kafka 客户端会根据 *客户端 CA*验证

8.2. 身份验证

Kafka 侦听程序使用身份验证来确保到 Kafka 集群的安全客户端连接。

支持的验证机制：

- mTLS 身份验证（在启用了 TLS 的加密的监听程序中）
- SASL SCRAM-SHA-512

- 基于 OAuth 2.0 令牌的身份验证
- 自定义身份验证

User Operator 管理 mTLS 和 SCRAM 身份验证的用户凭证，但不管理 OAuth 2.0。例如，通过 User Operator，您可以创建一个代表需要访问 Kafka 集群的客户端的用户，并将 **tls** 指定为身份验证类型。

使用基于 OAuth 2.0 令牌的身份验证时，应用程序客户端可以在不公开帐户凭证的情况下访问 Kafka 代理。授权服务器处理访问权限的授予和询问有关访问权限的查询。

自定义身份验证允许任何类型的 Kafka 支持的身份验证。它可以提供更大的灵活性，但也增加了复杂性。

8.3. 授权

Kafka 集群使用授权来控制特定客户端或用户在 Kafka 代理上允许的操作。如果应用到 Kafka 集群，会为用于客户端连接的所有监听程序启用授权。

如果一个用户被添加到 Kafka 代理配置的 *超级用户*列表中，无论通过授权机制实施的任何授权限制，用户都可以无限地访问集群。

支持的授权机制：

- 简单授权
- OAuth 2.0 授权（如果您使用基于 OAuth 2.0 令牌的身份验证）
- 开放策略代理 (OPA) 授权
- 自定义授权

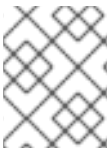
简单授权使用 **AclAuthorizer** 和 **StandardAuthorizer** Kafka 插件，该插件负责管理访问控制列表 (ACL)，用于指定用户对各种资源的访问权限。对于自定义授权，您可以将自己的 **Authorizer** 插件配置为强制实施 ACL 规则。

OAuth 2.0 和 OPA 从授权服务器提供基于策略的控制。在授权服务器中定义用于授予对 Kafka 代理资源访问权限的安全策略和权限。

URL 用于连接到授权服务器，并验证客户端或用户请求的操作是允许或拒绝的。用户和客户端与授权服务器中创建的策略匹配，允许访问 Kafka 代理上的特定操作。

8.4. 联邦信息处理标准(FIPS)

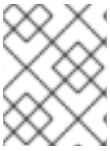
Federal Information Processing Standards (FIPS)是美国政府制定的一组安全标准，以确保信息系统处理或传输敏感数据的保密性、完整性和可用性。在启用了 FIPS 的 OpenShift 集群中运行时，在 Apache Kafka 容器镜像中使用的 OpenJDK 会自动启用 FIPS 模式。



注意

如果您不想在 Java OpenJDK 中启用 FIPS 模式，您可以使用 **FIPS_MODE** 环境变量在 Cluster Operator 的部署配置中禁用它。

有关 NIST 验证程序并验证模块的更多信息，请参阅 NIST 网站上 的加密模块验证计划。



注意

对于 Apache Kafka Proxy 和 Streams for Apache Kafka 控制台，还没有测试与 FIPS 支持的流的兼容性。虽然它们正常工作，但目前我们不能保证完全支持。

第 9 章 （预览） APACHE KAFKA 代理的流

Apache Kafka 代理的流是一个 Apache Kafka 协议感知代理，旨在增强基于 Kafka 的系统。通过它的过滤器机制，它允许在基于 Kafka 的系统中引入额外的行为，而无需更改您的应用程序或 Kafka 集群本身。

Apache Kafka 代理的流当前作为技术预览提供，它为 Apache Kafka 代理的记录加密过滤器引入了流。Record Encryption 过滤器提供加密，使用行业标准的加密技术将加密应用到 Kafka 消息，确保 Kafka 集群中存储的数据的机密性。

有关连接到 Apache Kafka 代理的流以及使用 Apache Kafka 代理的更多信息，请参阅 Apache Kafka 文档中的 代理指南。

第 10 章 （预览） APACHE KAFKA 控制台（用户界面）的流

部署 Kafka 集群后，您可以将其连接到 Apache Kafka 控制台的 Streams。Apache Kafka 控制台的流支持监控和管理 Kafka 集群。

连接由 Streams for Apache Kafka 管理的 Kafka 集群，以获取实时 insights 并从其用户界面优化集群性能。控制台的主页显示连接的 Kafka 集群，允许您访问有关代理、主题、分区和消费者组等组件的详细信息。

在 Apache Kafka 控制台的 Streams 中，您可以在进入到查看集群代理和主题的信息或连接到 Kafka 集群的消费者组前查看 Kafka 集群的状态。

您可以通过在控制台中检查以下内容来监控部署：

- 连接的集群
- 集群代理的状态
- 主题消息流
- 特定主题的分区状态
- 与特定主题关联的消费者组

有关连接到 Apache Kafka 控制台的流以及使用 Apache Kafka 控制台的更多信息，请参阅 Apache Kafka 文档中的 控制台指南。

注意

Apache Kafka 控制台的流当前作为技术预览提供。

第 11 章 监控

监控数据允许您监控 Apache Kafka 的 Streams 性能和健康状况。您可以配置部署来捕获指标数据进行分析和通知。

在调查连接和数据发送问题时，指标数据很有用。例如，指标数据可以识别出复制的分区或信息被消耗的速率。警报规则可以通过指定的通信频道提供此类指标上的时间通知。监控视觉化呈现实时指标数据，以帮助确定如何更新部署配置。Apache Kafka Streams 提供了指标配置文件示例。

分布式追踪通过 Apache Kafka 的 Streams 提供端到端跟踪信息，从而补充指标数据的收集。

Cruise Control 支持根据工作负载数据重新平衡 Kafka 集群。

指标和监控工具

Apache Kafka 的流可使用以下工具进行指标和监控：

Prometheus

Prometheus 从 Kafka、ZooKeeper 和 Kafka Connect 集群拉取指标。Prometheus **Alertmanager** 插件处理警报并将其路由到通知服务。

Kafka Exporter

Kafka Exporter 添加额外的 Prometheus 指标。

Grafana

Grafana Labs 提供 Prometheus 指标的仪表板视觉化。

OpenTelemetry

OpenTelemetry 文档 提供了分布式追踪支持，用于跟踪应用程序间的事务。

Sything Control

Cruise Control 监控数据分布，并在 Kafka 集群间执行数据重新平衡。

11.1. PROMETHEUS

Prometheus 可以从 Kafka 组件和 Apache Kafka Operator 的 Streams 中提取指标数据。

要使用 Prometheus 获取指标数据并提供警报，必须部署 Prometheus 和 Prometheus Alertmanager 插件。还必须使用指标配置部署或重新部署 Kafka 资源，以公开指标数据。

Prometheus 提取公开的指标数据用于监控。当条件根据预定义的警报规则来指示潜在的问题时，Alertmanager 会发出警报。

Apache Kafka 的 Streams 提供了指标和警报规则配置文件示例。由 Apache Kafka 的 Streams 提供的示例警报机制被配置为将通知发送到 Slack 频道。

11.2. GRAFANA

Grafana 使用 Prometheus 公开的指标数据呈现用于监控的仪表板视觉化。

需要部署 Grafana，Prometheus 会添加为数据源。通过 Grafana 界面导入 Apache Kafka 的 Streams 作为 JSON 文件的仪表板示例，以显示监控数据。

11.3. KAFKA EXPORTER

Kafka Exporter 是一个开源项目，用于增强对 Apache Kafka 代理和客户端的监控。Kafka Exporter 通过 Kafka 集群部署，从与偏移、消费者组、消费者群和主题相关的 Kafka 代理提取额外的 Prometheus 指标数据。您可以使用提供的 Grafana 仪表板来视觉化 Prometheus 从 Kafka Exporter 收集的数据。

Apache Kafka 的 Streams 提供了示例配置文件、警报规则和 Kafka Exporter 的 Grafana 仪表板。

11.4. 分布式追踪

分布式追踪跟踪分布式系统中应用程序间的事务进度。在微服务架构中，追踪跟踪服务间事务的进度。跟踪数据对于监控应用程序性能和目标系统和最终用户应用程序的问题非常有用。

在 Apache Kafka 的流中，追踪有助于对消息的端到端跟踪：从源系统到 Kafka，然后从 Kafka 到目标系统和应用程序。分布式追踪补充了 Grafana 仪表板中的指标监控，以及组件日志记录器。

以下 Kafka 组件内置了对追踪的支持：

- MirrorMaker 将来自源集群的信息追踪到目标集群
- Kafka 连接到由 Kafka Connect 使用和生成的 trace 信息
- Kafka Bridge 用于跟踪 Kafka 和 HTTP 客户端应用程序之间的信息

Kafka 代理不支持追踪。

您可以通过其自定义资源为这些组件启用和配置追踪。您可以使用 **spec.template** 属性添加追踪配置。

您可以使用 **spec.tracing.type** 属性指定追踪类型来启用追踪：

OpenTelemetry

指定 **type: opentelemetry** 以使用 OpenTelemetry。默认情况下，OpenTelemetry 使用 OTLP (OpenTelemetry 协议) exporter 和端点来获取 trace 数据。您可以指定 OpenTelemetry 支持的其他追踪系统，包括 Jaeger tracing。要做到这一点，您可以在追踪配置中更改 OpenTelemetry 导出器和端点。

小心

Apache Kafka 的流不再支持 OpenTracing。如果您之前将 OpenTracing 与 **type: jaeger** 选项搭配使用，我们建议您改为使用 OpenTelemetry 过渡到。

Kafka 客户端的追踪
还可以设置 Kafka producer 和使用者等客户端应用程序，以便监控事务。客户端配置了追踪配置集，并且初始化 tracer 供客户端应用使用。

11.5. SYTHING CONTROL

Cruise Control 是一个开源系统，它支持以下 Kafka 操作：

- 监控集群工作负载
- 根据预定义的限制重新平衡集群

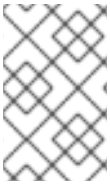
运行更均衡的 Kafka 集群来帮助更有效地使用代理 pod。

典型的集群随着时间的推移可能会不均匀地加载。处理大量消息流量的分区可能无法在可用代理中均匀分布。要重新平衡集群，管理员必须监控代理上的负载，并将忙碌的分区手动分配给具有备用容量的代理。

Cruise Control 自动执行集群重新平衡过程。它为集群构建了一个基于 CPU、磁盘和网络负载的资源利用率 *的工作负载模型*，并为更多均衡的分区分配生成优化建议（您可以批准或拒绝）。一组可配置的优化目标用于计算这些提议。

您可以在特定模式中生成优化方案。默认 **full** 模式在所有代理间重新平衡分区。您还可以使用 **add-brokers** 和 **remove-brokers** 模式来适应扩展集群或缩减时的更改。

当您批准一个优化建议时，Cruise Control 会将它应用到您的 Kafka 集群。您可以使用 **KafkaRebalance** 资源配置和生成优化建议。您可以使用注解配置资源，以便自动或手动批准优化提议。



注意

Prometheus 可以提取 Cruise Control 指标数据，包括与优化提议和重新平衡操作相关的数据。使用 Apache Kafka 的 Streams 提供了示例配置文件和 Cruise Control 的 Grafana 仪表板。

附录 A. 使用您的订阅

Apache Kafka 的流通过软件订阅提供。要管理您的订阅，请访问红帽客户门户中的帐户。

访问您的帐户

1. 转至 access.redhat.com。
2. 如果您还没有帐户，请创建一个帐户。
3. 登录到您的帐户。

激活订阅

1. 转至 access.redhat.com。
2. 导航到 **My Subscriptions**。
3. 导航到 **激活订阅** 并输入您的 16 位激活号。

下载 Zip 和 Tar 文件

要访问 zip 或 tar 文件，请使用客户门户网站查找下载的相关文件。如果您使用 RPM 软件包，则不需要这一步。

1. 打开浏览器并登录红帽客户门户网站 **产品下载页面**，网址为 access.redhat.com/downloads。
2. 在 **INTEGRATION AND AUTOMATION** 目录中找到 **Apache Kafka for Apache Kafka**的流。
3. 选择 Apache Kafka 产品所需的流。此时会打开 **Software Downloads** 页面。
4. 单击组件的 **Download** 链接。

使用 DNF 安装软件包

要安装软件包以及所有软件包的依赖软件包，请使用：

```
dnf install <package_name>
```

要从本地目录中安装之前下载的软件包，请使用：

```
dnf install <path_to_download_package>
```

更新于 2024-04-30