Red Hat AMQ 2021.Q3

在 OpenShift 中使用 AMQ Streams

用于 OpenShift Container Platform 上的 AMQ Streams 1.8

Last Updated: 2023-02-23

Red Hat AMQ 2021.Q3 在 OpenShift 中使用 AMQ Streams

用于 OpenShift Container Platform 上的 AMQ Streams 1.8

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

本指南描述了如何安装、配置和管理 Red Hat AMQ Streams 来构建大规模消息传递网络。

目录

使开源包含更多

红帽承诺替换我们的代码、文档和网页属性中存在问题的语言。我们从这四个术语开始： master、slave、blacklist 和 whitelist。这些更改将在即将发行的几个发行本中逐渐实施。如需了解更多详细信息，请参阅 CTO Chris Wright 信息。

第 1 章 AMQ 流概述

AMQ Streams 简化了在 OpenShift 集群中运行 Apache Kafka 的过程。

本指南提供有关配置 Kafka 组件和使用 AMQ Streams Operator 的说明。步骤与您可能想要修改部署的方式相关，并引进其他功能，如 Cruise Control 或分布式追踪。

您可以使用 AMQ Streams 自定义资源 配置部署。自定义资源 API 引用 描述了您可以在配置中使用的属性。

注意

您是否想要开始使用 AMQ Streams？有关逐步部署说明，*请参阅 OpenShift 中的部署和升级 AMQ Streams 指南*。

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的吞吐量和低延迟共享数据
- 消息排序保证
- 从数据存储中重获/重播消息以重建应用程序状态
- 使用键值日志时删除旧记录的消息紧凑
- 集群配置中的水平可扩展性
- 数据复制来控制容错
- 保留大量数据以便立即访问

1.2. KAFKA 用例

Kafka 的功能使其适合：

- 事件驱动的架构
- 事件源，以捕获对应用状态的更改作为事件日志
- 消息代理
- 网站活动跟踪
- 通过指标进行操作监控
- 日志收集和聚合
- 为分布式系统提交日志
- 流处理，以便应用程序能够实时响应数据

1.3. AMQ 流如何支持 KAFKA

AMQ Streams 提供容器镜像和 Operator，以便在 OpenShift 上运行 Kafka。AMQ Streams Operator 是运行 AMQ Streams 的基础。AMQ Streams 提供的 Operator 是专门构建的，具有可有效管理 Kafka 的专业操作知识。

Operator 简化了以下流程：

- 部署和运行 Kafka 集群
- 部署和运行 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

1.4. AMQ STREAMS OPERATOR

AMQ Streams *支持使用 Operator 的 Kafka 来部署和管理 Kafka 到 OpenShift 的组件和依赖项。*

Operator 是一种打包、部署和管理 OpenShift 应用的方法。AMQ Streams Operator 扩展 OpenShift 功能，自动执行与 Kafka 部署相关的常见复杂任务。通过在代码中了解 Kafka 操作，Kafka 管理任务可以简化，无需人工干预。

Operator
AMQ Streams 提供 Operator 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 和 Entity Operator

实体 Operator

由主题 Operator 和 User Operator 组成

主题 Operator

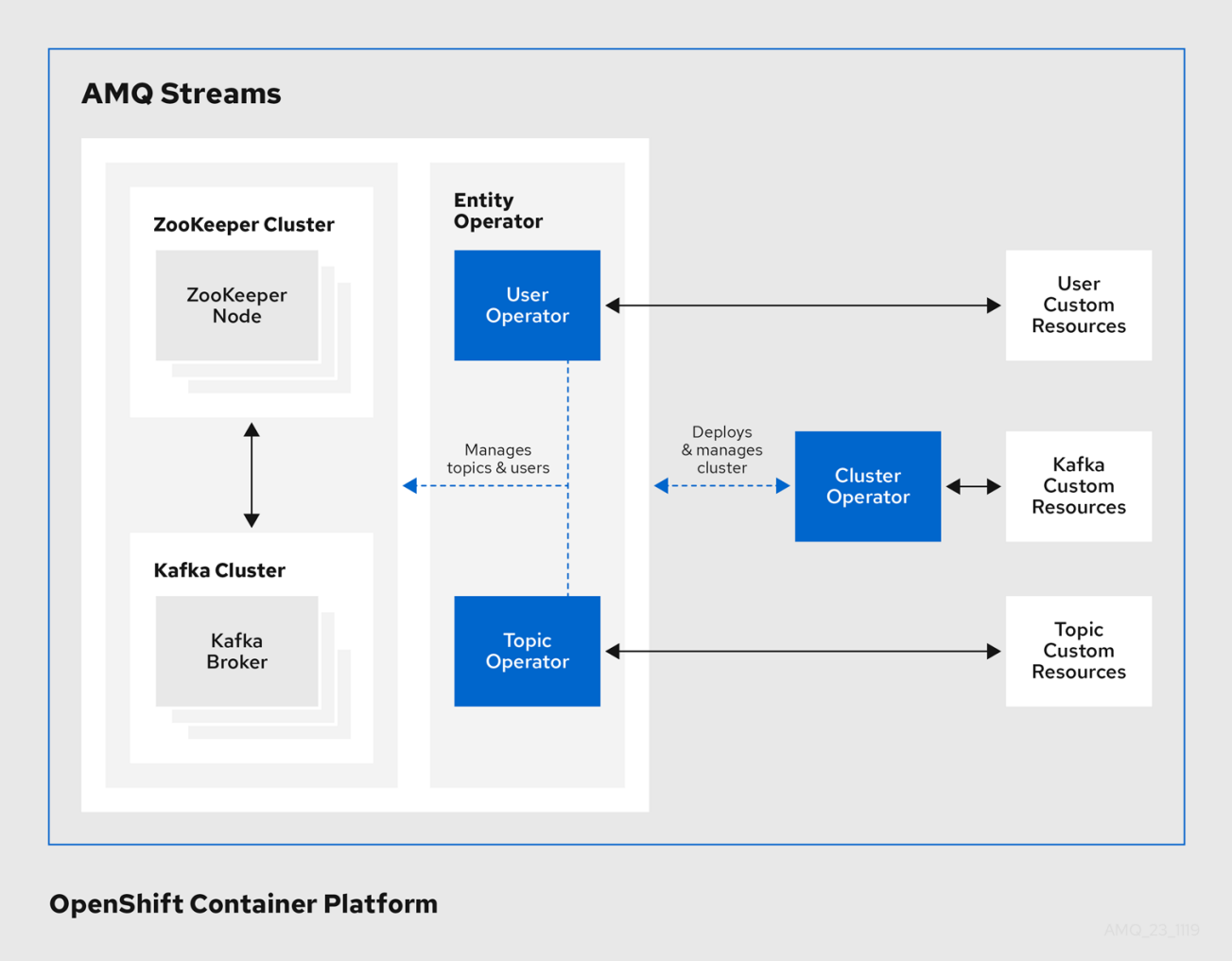
管理 Kafka 主题

User Operator

管理 Kafka 用户

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

AMQ Streams 架构中的 Operator



1.4.1. Cluster Operator

AMQ Streams 使用 Cluster Operator 来部署和管理集群：

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

自定义资源用于部署集群。

例如，部署 Kafka 集群：

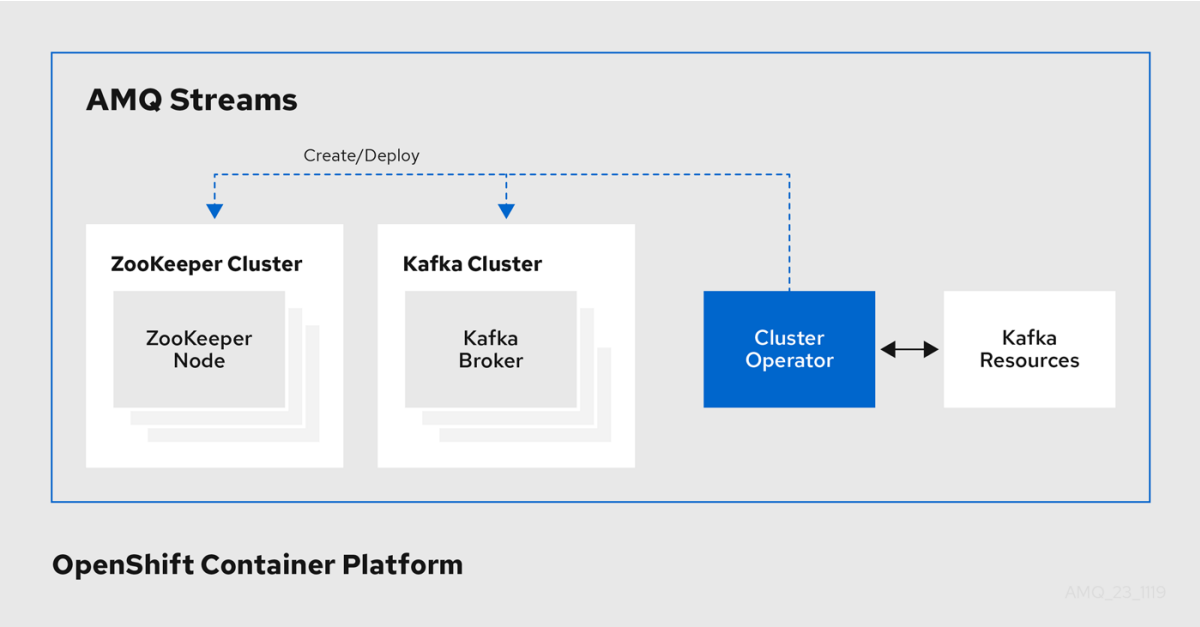
- 在 **OpenShift 集群中创建了带有集群配置的 Kafka** 资源。
- Cluster Operator 根据 Kafka 资源中声明的内容部署对应的 **Kafka** 集群。

Cluster Operator 也可以部署（通过配置 **Kafka** 资源）：

- 通过 **KafkaTopic** 自定义资源提供 operator 风格主题管理的主题 Operator
- 一个 User Operator，通过 **KafkaUser** 自定义资源提供 operator 风格的用户管理

部署的 Entity Operator 中的 Topic Operator 和 User Operator 功能。

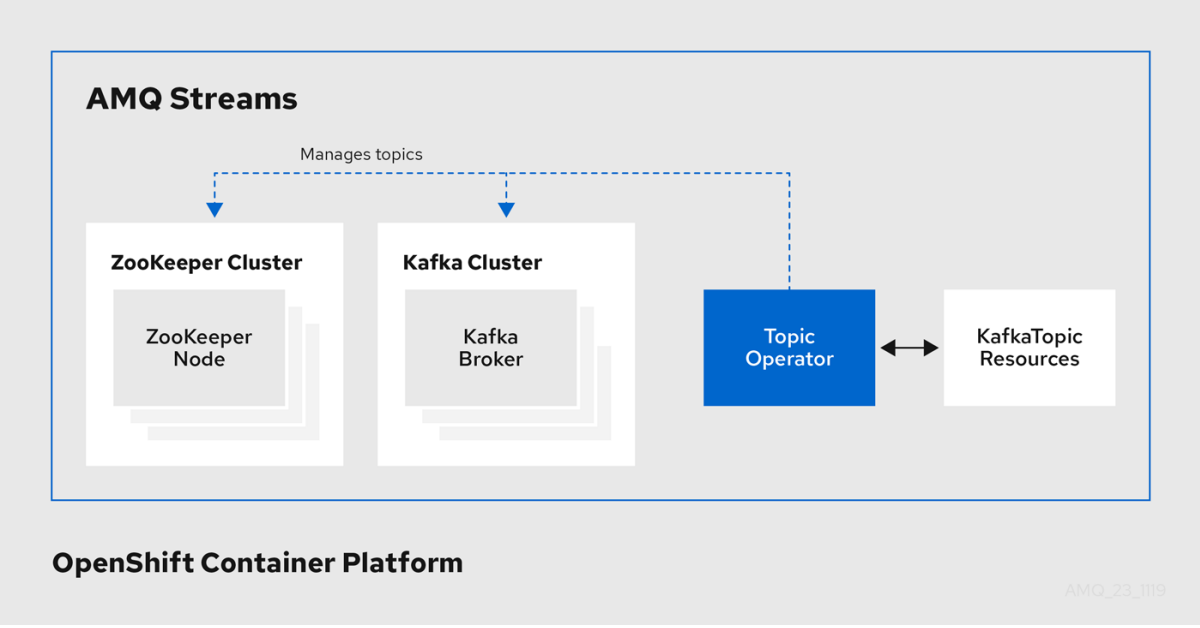
Cluster Operator 的架构示例



1.4.2. 主题 Operator

Topic Operator 提供了通过 OpenShift 资源管理 Kafka 集群中主题的方法。

Topic Operator 的架构示例



*Topic Operator 的角色是保留一组 **KafkaTopic** OpenShift 资源，描述与对应的 Kafka 主题同步中的 Kafka 主题。*

*特别是，如果 **KafkaTopic** 是：*

- *创建，主题 Operator 会创建该主题*
- *删除的，主题 Operator 会删除该主题*
- *更改，主题 Operator 更新该主题*

在另一个方向上工作，如果一个主题是：

- 在 Kafka 集群中创建，Operator 会创建一个 **KafkaTopic**
- 从 Kafka 集群中删除，Operator 会删除 **KafkaTopic**
- 在 Kafka 集群中更改，Operator 会更新 **KafkaTopic**

这可让您声明 **KafkaTopic** 作为应用程序部署的一部分，Topic Operator 将为您创建该主题。您的应用程序只需要处理从所需主题中产生或使用的内容。

Topic Operator 在主题 存储中维护每个主题 的信息， 它与来自 Kafka 主题或 OpenShift **KafkaTopic** 自定义资源的更新持续同步。应用到本地内存中主题存储的操作的更新会保留在磁盘上的备份主题存储中。如果某个主题被重新配置或重新分配给其他代理，**KafkaTopic** 将始终保持最新状态。

1.4.3. User Operator

User Operator 通过监视 Kafka 用户描述 **Kafka 用户并确保在** Kafka 集群中正确配置了这些用户来管理 Kafka 集群的 Kafka 用户。

例如， 如果 **KafkaUser** 是：

- 创建，User Operator 创建它描述的用户
- 删除时，User Operator 会删除它所描述的用户
- 更改后，User Operator 会更新它所描述的用户

与主题 Operator 不同，User Operator 不会将 Kafka 集群的任何更改与 OpenShift 资源同步。Kafka 主题可以直接由 Kafka 中的应用程序创建，但用户不必与 User Operator 并行直接在 Kafka 集群中管理。

User Operator 允许您将 **KafkaUser** 资源声明为应用程序部署的一部分。您可以为用户指定身份验证和授权机制。您还可以配置 用户配额 来控制对 Kafka 资源的使用，例如，确保用户不会专利对代理的访问。

创建用户时，会在 **Secret** 中创建用户凭据。您的应用需要使用用户 及其凭据进行身份验证，并生成或使用消息。

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

1.4.4. AMQ Streams Operator 中的功能门

您可以使用功能 门来启用和禁用 Operator 的一些功能。

功能门在操作器配置中设置，具有三个成熟度阶段：alpha、beta 或 General Availability(GA)。

如需更多信息，请参阅 功能门。

1.5. AMQ STREAMS 自定义资源

使用 AMQ Streams 将 Kafka 组件部署到 OpenShift 集群可通过应用自定义资源进行高度配置。自定义资源作为自定义资源定义(CRD)添加的 API 实例创建，以扩展 OpenShift 资源。

CRD 用作描述 OpenShift 集群中自定义资源的配置说明，并为部署中使用的每个 Kafka 组件以及用户和主题提供 AMQ Streams。CRD 和自定义资源定义为 YAML 文件。AMQ Streams 发行版提供了示例 YAML 文件。

CRD 还允许 AMQ Streams 资源受益于原生 OpenShift 功能，如 CLI 访问和配置验证。

其它资源

- 使用 CustomResourceDefinitions 扩展 Kubernetes API

1.5.1. AMQ Streams 自定义资源示例

CRD 需要在集群中进行一次性安装，以定义用于实例化和管理 AMQ Streams 特定资源的 schema。

通过安装 CRD 在集群中添加新的自定义资源类型后，您可以根据具体规格创建资源实例。

根据集群设置，安装通常需要集群管理员特权。

注意

管理自定义资源的访问权限仅限于 AMQ Streams 管理员。如需更多信息，请参阅 OpenShift 上部署和升级 AMQ 流指南中的指定 AMQ 流 管理员。

CRD 在 OpenShift 集群中定义 **一种** 新型资源，如 **kind:Kafka**。

Kubernetes API 服务器允许基于 **kind** 创建自定义资源，并且从 CRD 中了解如何在添加到 OpenShift 集群时验证和存储自定义资源。

警告

当 CRD 被删除时，该类型的自定义资源也会被删除。另外，自定义资源创建的资源也会被删除，如 pod 和 statefulsets。

每个 AMQ Streams 特定自定义资源都符合 CRD 为资源类型定义的 schema 。AMQ Streams 组件的自定义资源具有常见的配置属性，这些属性在 **spec** 下定义。

要了解 CRD 和自定义资源之间的关系，让我们看一下 Kafka 主题的 CRD 示例。

Kafka 主题 CRD

```
apiVersion: kafka.strimzi.io/v1beta2
kind: CustomResourceDefinition
metadata: 1
  name: kafkatopics.kafka.strimzi.io
  labels:
    app: strimzi
spec: 2
  group: kafka.strimzi.io
  versions:
    v1beta2
  scope: Namespaced
  names:
    # ...
    singular: kafkatopic
    plural: kafkatopics
    shortNames:
```

```
  - kt (3)
 additionalPrinterColumns: (4)
   # ...
 subresources:
  status: {} (5)
 validation: (6)
  openAPIV3Schema:
   properties:
    spec:
     type: object
     properties:
      partitions:
       type: integer
       minimum: 1
      replicas:
       type: integer
       minimum: 1
       maximum: 32767
   # ...
```

(1) *主题 CRD 的元数据、名称和标识 CRD 的标签。*

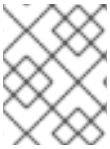
(2) *此 CRD 的规格，包括组（域）名称、复数名称和支持的 schema 版本，用于 URL 以访问该主题的 API。其他名称用于标识 CLI 中的实例资源。例如，oc **get kafkatopic my-topic** 或 **oc get kafkatopics**。*

(3) *可以在 CLI 命令中使用短名称。例如，**oc get kt** 可用作缩写而不是 **oc get kafkatopic**。*

(4) *在自定义资源上使用 **get** 命令时显示的信息。*

(5) *CRD 的当前状态，如资源的 schema 引用 中所述。*

(6) *OpenAPIV3Schema 验证提供了用于创建主题自定义资源的验证。例如，一个主题至少需要一个分区和一个副本。*



注意

您可以识别 AMQ Streams 安装文件提供的 CRD YAML 文件，因为文件名包含索引号后跟 'Crd'。

*以下是 **KafkaTopic** 自定义资源的对应示例。*

Kafka 主题自定义资源

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic (1)
metadata:
  name: my-topic
  labels:
    strimzi.io/cluster: my-cluster (2)
spec: (3)
  partitions: 1
  replicas: 1
```

```
  config:
    retention.ms: 7200000
    segment.bytes: 1073741824
status:
  conditions: 4
    lastTransitionTime: "2019-08-20T11:37:00.706Z"
    status: "True"
    type: Ready
  observedGeneration: 1
  / ...
```

1 **kind** 和 **apiVersion** 标识自定义资源是实例的 CRD。

2 个标签，仅适用于 **KafkaTopic** 和 **KafkaUser** 资源，定义 Kafka 集群的名称（与 **Kafka** 资源的名称相同），该标签定义某个主题或用户所属的名称。

3 spec 显示主题的分区和副本数量，以及主题本身的配置参数。在本例中，消息保留周期保留在主题中，并指定日志的片段文件大小。

4 **KafkaTopic** 资源的状态条件。在 **lastTransitionTime** 中，类型 条件更改为 **Ready**。

自定义资源可以通过平台 CLI 应用到集群。创建自定义资源时，它将使用与 Kubernetes API 内置资源相同的验证。

创建 **KafkaTopic** 自定义资源后，Tpic Operator 将获得通知，并在 AMQ Streams 中创建对应的 Kafka 主题。

1.6. 监听程序配置

侦听器用于连接到 Kafka 代理。

AMQ Streams 提供了一个 generic **GenericKafkaListener** 模式，其中包含用于通过 **Kafka** 资源配置监听器的属性。

The **GenericKafkaListener** 提供了用于监听器配置的灵活方法。您可以指定属性来配置 内部 侦听器，以便在 OpenShift 集群内连接，或者指定 外部 侦听器以连接 OpenShift 集群外部。

每个监听程序都 定义为 **Kafka** 资源中的一个数组。您可以根据需要配置任意数量的侦听器，只要它们的名称和端口是唯一的。

您可能希望配置多个外部监听器，例如，以处理来自需要不同身份验证机制的网络的访问。或者，您可能需要将 OpenShift 网络加入外部网络。在这种情况下，您可以配置内部侦听器（**使用 ServiceDnsDomain** 属性），以便不使用 OpenShift 服务 DNS 域（通常为 **.cluster.local**）。

有关监听器可用的配置选项的更多信息，请参阅 **GenericKafkaListener** 模式参考。

配置监听程序来保护对 Kafka 代理的访问

您可以使用身份验证为安全连接配置监听程序。有关保护对 Kafka 代理的访问的更多信息，请参阅管理对 Kafka 的访问。

为 OpenShift 外部的客户端访问配置外部侦听器

您可以使用指定的连接机制（如 loadbalancer），为 OpenShift 环境外的客户端访问配置外部侦听器。有关连接外部客户端的配置选项的更多信息，请参阅配置外部监听程序。

监听程序证书

您可以为启用了 TLS 加密的 TLS 监听程序或者外部监听程序提供自己的服务器证书，称为 Kafka 侦听程序证书。如需更多信息，请参阅 Kafka 侦听程序证书。

1.7. 文档规范

Replaceables

在本文档中，可替换的文本采用 单空间 格式，使用斜体、大写和连字符。

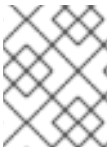
例如，在以下代码中，您要将 **MY-NAMESPACE** 替换为命名空间的名称：

```
sed -i 's/namespace: .*/namespace: MY-NAMESPACE/' install/cluster-operator/*RoleBinding*.yaml
```

第 2 章 部署配置

本章论述了如何使用自定义资源配置受支持的部署的不同方面：

- Kafka 集群
- Kafka Connect 集群
- 支持 Source2Image 的 Kafka Connect 集群
- Kafka MirrorMaker
- Kafka Bridge
- Sything Control



注意

应用到自定义资源的标签也会应用到由 Kafka MirrorMaker 组成的 OpenShift 资源。这提供了一种便捷的机制，可使资源根据需要进行标记。

2.1. KAFKA 集群配置

本节论述了如何在 AMQ Streams 集群中配置 Kafka 部署。Kafka 集群使用 ZooKeeper 集群部署。部署也可以包括管理 Kafka 主题和用户的 Topic Operator 和 User Operator。

您可以使用 Kafka 资源配置 **Kafka**。配置选项也可用于 **Kafka** 资源中的 ZooKeeper 和 Entity Operator。Entity Operator 由 Topic Operator 和 User Operator 组成。

Kafka 资源的完整 schema 信息包括在 第 13.2.1 节 "**Kafka** 模式参考" 中。

监听程序配置

您可以配置监听程序以将客户端连接到 Kafka 代理。有关为连接代理配置监听程序的更多信息，请参阅 Listener 配置。

授权对 Kafka 的访问

您可以将 Kafka 集群配置为允许或拒绝用户执行的操作。有关保护对 Kafka 代理的访问的更多信息，请参阅管理对 Kafka 的访问。

管理 TLS 证书

在部署 Kafka 时，Cluster Operator 会自动设置并更新 TLS 证书，以在集群中启用加密和身份验证。如果需要，您可以在续订周期结束前手动续订集群和客户端 CA 证书。您还可以替换集群和客户端 CA 证书使用的密钥。如需更多信息，请参阅 手动重新更新 CA 证书和 替换私钥。

其它资源

- 有关 Apache Kafka 的详情，请查看 Apache Kafka 网站。

2.1.1. 配置 Kafka

使用 **Kafka** 资源的属性来配置 Kafka 部署。

除了配置 Kafka 外，您还可以为 ZooKeeper 和 AMQ Streams Operator 添加配置。常见的配置属性（如日志记录和健康检查）是为每个组件独立配置的。

此流程仅显示一些可能的配置选项，但尤为重要选项包括：

- 资源请求(CPU / Memory)
- 用于最大和最小内存分配的 JVM 选项
- 侦听器（和客户端身份验证）
- Authentication
- 存储
- 机架感知
- 指标
- 用于集群重新平衡的精简控制

Kafka 版本

Kafka **配置** 的 **log.message.format .version 和 inter.broker.protocol.version** 属性必须是指定 Kafka 版本(**spec.kafka.version**)支持的版本。属性表示附加到消息的日志格式版本，以及 Kafka 集群中使用的 Kafka 协议版本。在升级 Kafka 版本时，需要更新这些属性。如需更多信息，请参阅 OpenShift 上部署 和升级 AMQ Streams 中的升级 Kafka。

先决条件

- OpenShift 集群
- 一个正在运行的 Cluster Operator

有关 部署方法的说明，请参阅 OpenShift 指南中的部署和升级 AMQ Streams ：

- Cluster Operator
- Kafka 集群

流程

1. 编辑 **Kafka** 资源的 **spec** 属性。
   您可以配置的属性显示在此示例配置中：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3 (1)
    version: 2.8.0 (2)
    logging: (3)
      type: inline
      loggers:
```

```
    kafka.root.logger.level: "INFO"
  resources: 4
    requests:
      memory: 64Gi
      cpu: "8"
    limits:
      memory: 64Gi
      cpu: "12"
  readinessProbe: 5
    initialDelaySeconds: 15
    timeoutSeconds: 5
  livenessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 5
  jvmOptions: 6
    -Xms: 8192m
    -Xmx: 8192m
  image: my-org/my-image:latest 7
  listeners: 8
    - name: plain 9
      port: 9092 10
      type: internal 11
      tls: false 12
      configuration:
        useServiceDnsDomain: true 13
    - name: tls
      port: 9093
      type: internal
      tls: true
      authentication: 14
        type: tls
    - name: external 15
      port: 9094
      type: route
      tls: true
      configuration:
        brokerCertChainAndKey: 16
          secretName: my-secret
          certificate: my-certificate.crt
          key: my-key.key
  authorization: 17
    type: simple
  config: 18
    auto.create.topics.enable: "false"
    offsets.topic.replication.factor: 3
    transaction.state.log.replication.factor: 3
    transaction.state.log.min.isr: 2
    log.message.format.version: 2.8
    inter.broker.protocol.version: 2.8
    ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" 19
    ssl.enabled.protocols: "TLSv1.2"
    ssl.protocol: "TLSv1.2"
  storage: 20
    type: persistent-claim 21
```

```
    size: 10000Gi 22
  rack: 23
    topologyKey: topology.kubernetes.io/zone
  metricsConfig: 24
    type: jmxPrometheusExporter
    valueFrom:
      configMapKeyRef: 25
        name: my-config-map
        key: my-key
  # ...
zookeeper: 26
  replicas: 3 27
  logging: 28
    type: inline
    loggers:
      zookeeper.root.logger: "INFO"
  resources:
    requests:
      memory: 8Gi
      cpu: "2"
    limits:
      memory: 8Gi
      cpu: "2"
  jvmOptions:
    -Xms: 4096m
    -Xmx: 4096m
  storage:
    type: persistent-claim
    size: 1000Gi
  metricsConfig:
    # ...
entityOperator: 29
  tlsSidecar: 30
    resources:
      requests:
        cpu: 200m
        memory: 64Mi
      limits:
        cpu: 500m
        memory: 128Mi
  topicOperator:
    watchedNamespace: my-topic-namespace
    reconciliationIntervalSeconds: 60
    logging: 31
      type: inline
      loggers:
        rootLogger.level: "INFO"
    resources:
      requests:
        memory: 512Mi
        cpu: "1"
      limits:
        memory: 512Mi
        cpu: "1"
  userOperator:
```

```
    watchedNamespace: my-topic-namespace
    reconciliationIntervalSeconds: 60
    logging: 32
      type: inline
      loggers:
        rootLogger.level: INFO
    resources:
      requests:
        memory: 512Mi
        cpu: "1"
      limits:
        memory: 512Mi
        cpu: "1"
  kafkaExporter: 33
    # ...
  cruiseControl: 34
    # ...
    tlsSidecar: 35
    # ...
```

1 副本节点的数量。如果您的集群已经定义了主题，您可以扩展集群。

2 Kafka 版本，可按照升级过程将其改为受支持的版本。

3 指定 Kafka 日志记录器和日志级别直接（**内联**）或通过 ConfigMap 间接（**外部**）添加。自定义 ConfigMap 必须放在 **log4j.properties** 键下。对于 Kafka **kafka.root.logger.level** logger，您可以将日志级别设置为 INFO、ERROR、WARN、TRACE、DEBUG、FATAL 或 OFF。

4 请求保留支持的资源、当前 **cpu 和 memory，以及限制，以指定可消耗的最大资源。**

5 **健康检查以了解何时重新启动容器（存活度）以及容器何时可以接受流量（就绪度）。**

6 **用于优化运行 Kafka 的虚拟机(VM)性能的 JVM 配置选项。**

7 **ADVANCED OPTION：容器镜像配置，只在特殊情况下推荐这样做。**

8 侦听器配置客户端如何通过 **bootstrap** 地址连接到 **Kafka** 集群。侦听器配置为**内部或外部**侦听器，以便从 **OpenShift** 集群内部或外部进行连接。

9

10

Kafka 内侦听器使用的端口号。在给定 Kafka 集群中，端口号必须是唯一的。允许的端口号是 9092 及以上，但端口 9404 和 9999 除外，它们已用于 Prometheus 和 JMX。根据监听程序类型，端口号可能与连接 Kafka 客户端的端口号不同。

11

侦听器类型指定为 内部，或用于外部监听器，作为 路由、负载均衡器、 节点端口 或入口。

12

为每个侦听器启用 TLS 加密。默认为 false。路由 监听程序不需要 TLS 加密。

13

定义是否分配了包括集群服务后缀（通常. cluster.local）在内的完全限定 DNS 名称。

14

侦听器身份验证机制 指定为 mutual TLS、SCRAM-SHA-512 或基于令牌的 OAuth 2.0。

15

外部监听程序配置指定 Kafka 集群如何公开 OpenShift 外部，如通过 路由、loadbalancer 或 nodeport。

16

由外部 证书颁发机构管理的 Kafka 侦听器 证书的可选配置。brokerCertChainAndKey 指定包含服务器证书和私钥的 Secret。您可以在任何启用 TLS 加密的监听程序上配置 Kafka 侦听程序证书。

17

授权在 Kafka 代理上启用简单、OAUTH 2.0 或 OPA 授权。简单授权使用 AclAuthorizer Kafka 插件。

18

config 指定代理配置。标准 Apache Kafka 配置可能会提供，仅限于不是由 AMQ Streams 直接管理的属性。

19

启用 TLS 加密的监听器的 SSL 属性，以启用特定的 密码套件 或 TLS 版本。

20

存储 被配置为 临时、持久声明或 jbod。

21

持久性卷的存储大小可能会增加，并 可向 JBOD 存储添加额外的卷。

22

持久性存储具有 额外的配置选项，如用于动态卷置备的存储 id 和 class。

23

将机架感知 配置为在不同机架之间分布副本。topologykey 必须与集群节点标签匹配。

24

启用了 Prometheus 指标。在本例中，为 Prometheus JMX Exporter（默认指标导出器）配置了指标数据。

25

通过 Prometheus JMX Exporter 将指标导出到 Grafana 仪表板的 Prometheus 规则，这可以通过引用包含 Prometheus JMX 导出器配置的 ConfigMap 来启用。您可以使用对包含 metricsConfig.valueFrom.configMapKeyRef.key 下的空文件的 ConfigMap 的引用来启用指标，而无需进一步配置。

26

zookeeper 特定的配置，其中包含与 Kafka 配置类似的属性。

27

ZooKeeper 节点的数量。zookeeper 集群或ensemble 通常使用奇数的节点（通常为三个、五个或 7 个）运行。大多数节点必须可用，才能保持有效的仲裁。如果 ZooKeeper 集群丢失其仲裁，它将停止响应客户端，Kafka 代理将停止工作。拥有稳定且高度可用的 ZooKeeper 集群对于 AMQ 流至关重要。

28

指定 ZooKeeper 日志记录器和日志级别。

29

30

实体操作器 TLS sidecar 配置.实体 Operator 使用 TLS sidecar 与 ZooKeeper 进行安全通信。

31

指定 Topic Operator 日志记录器和日志级别。这个示例使用 内联 日志记录。

32

指定 的用户 Operator 日志记录器和日志级别。

33

Kafka Exporter 配置.Kafka Exporter 是一个可选组件，用于从 Kafka 代理提取指标数据，特别是消费者滞后数据。

34

Cruise Control 的可选配置，用于 重新平衡 Kafka 集群。

35

插入控制 TLS sidecar 配置.cruise Control 使用 TLS sidecar 与 ZooKeeper 进行安全通信。

2. 创建或更新资源：

```
oc apply -f KAFKA-CONFIG-FILE
```

2.1.2. 配置实体 Operator

Entity Operator 负责管理正在运行的 Kafka 集群中与 Kafka 相关的实体。

Entity Operator 包括：

- 用于管理 Kafka 主题的主题 Operator

- 管理 Kafka 用户的用户 Operator

通过 Kafka 资源配置，Cluster Operator 可在部署 Kafka 集群时部署 Entity Operator，包括一个或多个操作器。

注意

部署后，实体 Operator 根据部署配置包含操作器。

Operator 会自动配置为管理 Kafka 集群的主题和用户。

2.1.2.1. 实体 Operator 配置属性

使用 Kafka.spec 中的 principal Operator 属性来配置 Entity Operator。

entity Operator 属性支持以下几个子属性：

- tlsSidecar
- topicOperator
- userOperator
- 模板

tlsSidecar 属性包含 TLS sidecar 容器的配置，用于与 ZooKeeper 通信。

template 属性包含 Entity Operator pod 的配置，如标签、注解、关联性和容限。有关配置模板的详情请参考 第 2.6 节 “自定义 OpenShift 资源”。

topicOperator 属性包含 Topic Operator 的配置。当缺少这个选项时，Entity Operator 会在没有

Topic Operator 的情况下部署。

userOperator 属性包含 User Operator 的配置。当缺少这个选项时，Entity Operator 会在没有 User Operator 的情况下部署。

有关配置 Entity Operator 的属性的更多信息，请参阅 EntityUserOperatorSpec schema 参考。

启用两个运算符的基本配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    topicOperator: {}
    userOperator: {}
```

如果将空对象({})用于 topicOperator 和 userOperator，则所有属性都使用其默认值。

当缺少 topicOperator 和 userOperator 属性时，Entity Operator 不会被部署。

2.1.2.2. 主题 Operator 配置属性

可以使用 topicOperator 对象中的附加选项来配置主题 Operator 部署。支持以下属性：

watchedNamespace

主题操作器监视 KafkaTopics 的 OpenShift 命名空间。默认为部署 Kafka 集群的命名空间。

reconciliationIntervalSeconds

定期协调之间的间隔（以秒为单位）。默认值 120.

zookeeperSessionTimeoutSeconds

ZooKeeper 会话超时，以秒为单位。默认值为 18。

topicMetadataMaxAttempts

从 Kafka 获取主题元数据的尝试数量。每次尝试之间的时间都定义为指数回退。考虑在因为分区或副本数量而创建主题时增加这个值。默认 6.

image

image 属性可用于配置要使用的容器镜像。有关配置自定义容器镜像的详情，请参考 第 13.1.6 节 “image”。

资源

resources 属性配置分配给 Topic Operator 的资源数量。有关资源请求和限制配置的详情，请参阅 第 13.1.5 节 “资源”。

logging

logging 属性配置 Topic Operator 的日志。如需了解更多详细信息，请参阅 第 13.2.45.1 节 “logging”。

Topic Operator 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    topicOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
    # ...
```

2.1.2.3. 用户 Operator 配置属性

用户 Operator 部署可以使用 userOperator 对象中的附加选项进行配置。支持以下属性：

watchedNamespace

用户操作员在其中监视 KafkaUsers 的 OpenShift 命名空间。默认为部署 Kafka 集群的命名空间。

reconciliationIntervalSeconds

定期协调之间的间隔（以秒为单位）。默认值 120.

zookeeperSessionTimeoutSeconds

ZooKeeper 会话超时，以秒为单位。默认值为 18。

image

image 属性可用于配置要使用的容器镜像。有关配置自定义容器镜像的详情，请参考 第 13.1.6 节 “image”。

资源

resources 属性配置分配给 User Operator 的资源数量。有关资源请求和限制配置的详情，请参阅 第 13.1.5 节 “资源”。

logging

logging 属性配置 User Operator 的日志记录。如需了解更多详细信息，请参阅 第 13.2.45.1 节 “logging”。

secretPrefix

secretPrefix 属性在从 KafkaUser 资源创建的所有 Secret 的名称中添加前缀。例如，STRIMZI_SECRET_PREFIX=kafka- 将为所有 Secret 名称添加前缀 kafka-。因此，名为 my-user 的 KafkaUser 将创建名为 kafka-my-user 的 Secret。

User Operator 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
```

```
  # ...
zookeeper:
  # ...
entityOperator:
  # ...
  userOperator:
    watchedNamespace: my-user-namespace
    reconciliationIntervalSeconds: 60
  # ...
```

2.1.3. Kafka 和 ZooKeeper 存储类型

作为有状态应用程序，Kafka 和 ZooKeeper 需要将数据存储在磁盘上。AMQ Streams 支持用于此数据的三种存储类型：

- ephemeral
- persistent
- JBOD 存储

注意

JBOD 存储只支持 Kafka，不适用于 ZooKeeper。

在配置 Kafka 资源时，您可以指定 Kafka 代理使用的存储类型及其对应的 ZooKeeper 节点。您可以使用以下资源中的 storage 属性配置存储类型：

- Kafka.spec.kafka
- Kafka.spec.zookeeper

存储类型在 type 字段中配置。

警告

部署 Kafka 集群后，无法更改存储类型。

其它资源

- 如需有关临时存储的更多信息，请参阅 临时存储模式参考。
- 有关持久性存储的更多信息，请参阅 持久性存储架构参考。
- 有关 JBOD 存储的更多信息，请参阅 JBOD 模式参考。
- 有关 Kafka 模式的更多信息，请参阅 Kafka 模式参考。

2.1.3.1. 数据存储注意事项

高效的数据存储基础架构对于 AMQ 流的最佳性能至关重要。

块存储是必需的。文件存储（如 NFS）无法使用 Kafka。

从以下选项中为块存储选择：

- 基于云的块存储解决方案，如 Amazon Elastic Block Store(EBS)
- 本地持久性卷
- 通过 光纤通道 或 iSCSI等协议访问的存储区域网络(SAN)卷

注意

AMQ Streams 不需要 OpenShift 原始块卷。

2.1.3.1.1. 文件系统

建议您将存储系统配置为使用 XFS 文件系统。AMQ Streams 还与 ext4 文件系统兼容，但为了获得最佳结果，可能需要额外的配置。

2.1.3.1.2. Apache Kafka 和 ZooKeeper 存储

Apache Kafka 和 ZooKeeper 使用单独的磁盘。

支持三种类型的数据存储：

- 临时（仅适用于开发的建议）
- persistent
- JBOD（只是一个磁盘绑定，仅适用于 Kafka）

如需更多信息，请参阅 Kafka 和 ZooKeeper 存储。

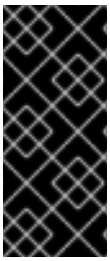
在异步发送和从多个主题接收数据的大型集群中，固态驱动器(SSD)（虽然不必要）可以提高 Kafka 的性能。SSD 在 ZooKeeper 中特别有效，它需要快速、低延迟的数据访问。

注意

您不需要置备复制存储，因为 Kafka 和 ZooKeeper 都内置了数据复制。

2.1.3.2. 临时存储

临时存储使用 emptyDir 卷来存储数据。要使用临时存储，请将 type 字段设置为 ephemeral。



重要

emptyDir 卷不是持久性的，存储在其中的数据会在 pod 重启时丢失。新 pod 启动后，它必须从集群的其他节点恢复所有数据。临时存储不适用于单节点 ZooKeeper 集群，或者适用于复制因数为 1 的 Kafka 主题。此配置将导致数据丢失。

临时存储示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    storage:
      type: ephemeral
    # ...
  zookeeper:
    # ...
    storage:
      type: ephemeral
    # ...
```

2.1.3.2.1. 日志目录

Kafka 代理使用临时卷作为挂载到以下路径的日志目录：

```
/var/lib/kafka/data/kafka-logIDX
```

其中 IDX 是 Kafka 代理 pod 索引。例如 /var/lib/kafka/data/kafka-log0。

2.1.3.3. 持久性存储

永久存储使用持久卷声明 来调配用于存储数据的持久卷。持久卷声明可用于调配许多不同类型的卷，具体取决于要调配卷的 存储类。可以与持久卷声明一起使用的数据类型包括许多类型的 SAN 存储 和本地持久卷。

若要使用持久存储，必须将 type 设置为 persistent-claim。持久性存储支持额外的配置选项：

id （可选）

存储标识号.对于 JBOD 存储声明中定义的存储卷，此选项是必需的。默认值为 0。

大小 （必需）

定义持久卷声明的大小，如 "1000Gi"。

class （可选）

用于动态卷置备的 OpenShift 存储类。

selector （可选）

允许选择要使用的特定持久性卷。它包含键：值对，代表用于选择此类卷的标签。

deleteClaim (optional)

指定在取消部署集群时是否需要删除持久卷声明的布尔值。默认为 false。

警告

仅支持重新定义持久性卷大小的 OpenShift 版本支持在现有 AMQ Streams 集群中增加持久性卷的大小。要调整持久性卷的大小必须使用支持卷扩展的存储类。对于不支持卷扩展的其他 OpenShift 版本和存储类，您必须在部署集群前决定必要的存储大小。无法减少现有持久性卷的大小。

使用 1000Gi 大小的持久性存储配置片段示例

```
# ...
storage:
  type: persistent-claim
  size: 1000Gi
# ...
```

以下示例演示了存储类的使用。

使用特定存储类的持久性存储配置片段示例

```
# ...
storage:
  type: persistent-claim
  size: 1Gi
  class: my-storage-class
# ...
```

最后，可以利用 选择器 来选择特定的标记持久卷，以提供 SSD 等必要功能。

使用选择器的持久性存储配置片段示例

```
# ...
storage:
  type: persistent-claim
  size: 1Gi
  selector:
    hdd-type: ssd
  deleteClaim: true
# ...
```

2.1.3.3.1. 存储类覆盖

您可以为一个或多个 Kafka 代理或 ZooKeeper 节点指定不同的存储类，而不是使用默认存储类。例如，当存储类仅限于不同的可用区或数据中心时，这很有用。您可以使用 overrides 字段来实现这一目的。

在本例中，默认存储类名为 my-storage-class ：

使用存储类覆盖的 AMQ Streams 集群示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  labels:
    app: my-cluster
  name: my-cluster
  namespace: myproject
spec:
  # ...
  kafka:
    replicas: 3
    storage:
      deleteClaim: true
      size: 100Gi
      type: persistent-claim
      class: my-storage-class
      overrides:
        - broker: 0
          class: my-storage-class-zone-1a
        - broker: 1
          class: my-storage-class-zone-1b
        - broker: 2
          class: my-storage-class-zone-1c
  # ...
  zookeeper:
    replicas: 3
    storage:
      deleteClaim: true
      size: 100Gi
      type: persistent-claim
      class: my-storage-class
      overrides:
        - broker: 0
          class: my-storage-class-zone-1a
        - broker: 1
          class: my-storage-class-zone-1b
        - broker: 2
          class: my-storage-class-zone-1c
  # ...
```

配置的 overrides 属性后，卷使用以下存储类：

- *ZooKeeper 节点 0 的持久卷将使用 my-storage-class-zone-1a。*

- *ZooKeeper 节点 1 的持久卷将使用 my-storage-class-zone-1b。*
- *ZooKeeepr 节点 2 的持久卷将使用 my-storage-class-zone-1c。*
- *Kafka 代理 0 的持久性卷将使用 my-storage-class-zone-1a。*
- *Kafka 代理 1 的持久性卷将使用 my-storage-class-zone-1b。*
- *Kafka 代理 2 的持久性卷将使用 my-storage-class-zone-1c。*

overrides 属性目前只用于覆盖存储类配置。目前不支持覆盖其他存储配置字段。目前不支持存储配置中的其他字段。

2.1.3.3.2. 持久性卷声明命名

使用持久性存储时，它会使用以下名称创建持久性卷声明：

data-cluster-name-kafka-idx

用于存储 Kafka 代理 pod IDx 数据的卷的持久性卷声明。

data-cluster-name-zookeeper-idx

用于为 ZooKeeper 节点 pod idx 存储数据的卷的持久性卷声明。

2.1.3.3.3. 日志目录

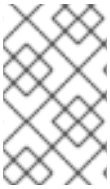
Kafka 代理使用持久性卷作为挂载到以下路径的日志目录：

```
/var/lib/kafka/data/kafka-logIDX
```

其中 IDX 是 Kafka 代理 pod 索引。例如 /var/lib/kafka/data/kafka-log0。

2.1.3.4. 重新定义持久性卷大小

您可以通过增大现有 AMQ Streams 集群使用的持久性卷的大小来置备存储容量。在使用一个持久性卷或在 JBOD 存储配置中使用多个持久性卷的集群中，支持重新定义持久性卷大小。



注意

您可以增大但不能缩小持久性卷的大小。OpenShift 目前不支持缩小持久卷的大小。

先决条件

- 个 OpenShift 集群，其支持卷大小调整。
- Cluster Operator 正在运行。
- 使用使用支持卷扩展的存储类创建的持久性卷的 Kafka 集群。

流程

1. 在 Kafka 资源中，增大分配给 Kafka 集群和 ZooKeeper 集群的持久性卷的大小。
   - 要增大分配给 Kafka 集群的卷大小，编辑 spec.kafka.storage 属性。
   - 要增大分配给 ZooKeeper 集群的卷大小，请编辑 spec.zookeeper.storage 属性。

     例如，将卷大小从 1000Gi 增加到 2000Gi ：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    storage:
      type: persistent-claim
      size: 2000Gi
      class: my-storage-class
```

```
    # ...
  zookeeper:
    # ...
```

2. 创建或更新资源：

```
oc apply -f KAFKA-CONFIG-FILE
```

OpenShift 会响应来自 Cluster Operator 的请求，增加了所选持久性卷的容量。完成大小调整后，Cluster Operator 会重启所有使用调整大小持久性卷的 pod。这会自动发生。

其它资源

有关在 OpenShift 中调整持久性卷大小的更多信息，请参阅使用 Kubernetes 重新定义持久性卷 大小。

2.1.3.5. JBOD 存储概述

您可以将 AMQ Streams 配置为使用 JBOD，这是多个磁盘或卷的数据存储配置。JBOD 是为 Kafka 代理提供更多数据存储的一种方法。它还能提高性能。

一个或多个卷描述了 JBOD 配置，每个卷可以 临时 或永久 。JBOD 卷声明的规则和限制与临时和永久存储的规则和限制相同。例如，您不能在置备后缩小持久性存储卷的大小，或者 type=ephemeral 时您无法更改 sizeLimit 的值。

2.1.3.5.1. JBOD 配置

若要将 JBOD 与 AMQ Streams 搭配使用，存储 类型 必须设置为 jbod。借助 volumes 属性，您可以描述组成 JBOD 存储阵列或配置的磁盘。以下片段显示了 JBOD 配置示例：

```
# ...
storage:
  type: jbod
  volumes:
  - id: 0
    type: persistent-claim
    size: 100Gi
    deleteClaim: false
  - id: 1
    type: persistent-claim
    size: 100Gi
    deleteClaim: false
# ...
```

创建 JBOD 卷后，无法更改 id。

用户可以从 JBOD 配置中添加或删除卷。

2.1.3.5.2. JBOD 和持久性卷声明

当使用持久性存储声明 JBOD 卷时，生成的持久性卷声明的命名方案如下：

data-id-cluster-name-kafka-idx

其中 id 是用于存储 Kafka 代理 pod idx 数据的卷 ID。

2.1.3.5.3. 日志目录

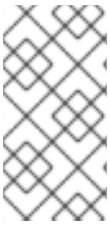
Kafka 代理将使用 JBOD 卷作为挂载到以下路径的日志目录：

*/var/lib/kafka/data-id/kafka-log_idx_*

其中 id 是用于存储 Kafka 代理 pod idx 数据的卷 ID。例如 /var/lib/kafka/data-0/kafka-log0。

2.1.3.6. 将卷添加到 JBOD 存储

这个步骤描述了如何在配置为使用 JBOD 存储的 Kafka 集群中添加卷。它不能应用到配置为使用任何其他存储类型的 Kafka 集群。



注意

在已经使用并被删除的 id 下添加新卷时，您必须确保之前使用的 PersistentVolumeClaims 已被删除。

先决条件

- *OpenShift 集群*
- *一个正在运行的 Cluster Operator*

- *带有 JBOD 存储的 Kafka 集群*

流程

1. *编辑 Kafka 资源中的 spec.kafka.storage.volumes 属性。将新卷添加到 volumes 数组。例如，添加 ID 为 2 的新卷：*

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
      - id: 1
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
      - id: 2
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
    # ...
  zookeeper:
    # ...
```

2. *创建或更新资源：*

```
oc apply -f KAFKA-CONFIG-FILE
```

3. *创建新主题或将现有分区重新分配至新磁盘.*

其它资源

有关重新分配主题的详情请参考 第 2.1.4.2 节 “分区重新分配”。

2.1.3.7. 从 JBOD 存储中删除卷

这个步骤描述了如何从配置为使用 JBOD 存储的 Kafka 集群中删除卷。它不能应用到配置为使用任何其他存储类型的 Kafka 集群。JBOD 存储始终必须至少包含一个卷。

重要

为避免数据丢失，您必须先移动所有分区，然后再删除卷。

先决条件

- *OpenShift 集群*
- *一个正在运行的 Cluster Operator*
- *带有两个或多个卷的 JBOD 存储的 Kafka 集群*

流程

1. *从要删除的磁盘中重新分配所有分区。分区中仍然分配到将要删除的磁盘中的任何数据都可能会丢失。*

2. *编辑 Kafka 资源中的 spec.kafka.storage.volumes 属性。从 volumes 阵列中删除一个或多个 卷。例如，删除 ID 为 1 和 2 的卷：*

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
    # ...
  zookeeper:
    # ...
```

3. 创建或更新资源：

```
oc apply -f KAFKA-CONFIG-FILE
```

其它资源

有关重新分配主题的详情请参考 第 2.1.4.2 节 “分区重新分配”。

2.1.4. 扩展集群

2.1.4.1. 扩展 Kafka 集群

2.1.4.1.1. 在集群中添加代理

提高某个主题吞吐量的主要方法是增加该主题的分区数量。这是因为额外的分区允许在集群中的不同代理之间共享该主题的负载。但是，如果每个代理都受到使用更多分区的特定资源（通常是 I/O）限制，则不会增加吞吐量。相反，您需要在集群中添加代理。

当您在集群中添加额外的代理时，Kafka 不会自动为其分配任何分区。您必须决定从现有代理迁移到新代理的分区。

在所有代理之间重新分布分区后，应该减少每个代理的资源利用率。

2.1.4.1.2. 从集群中删除代理

因为 AMQ Streams 使用 StatefulSet 来管理代理 Pod，所以 您无法从集群中删除 任何 pod。您只能从集群中移除一个或多个数字最高的 pod。例如，在一个有 12 个代理的集群中，pod 被命名为 cluster-name-kafka-0 to cluster-name-kafka-11。如果您决定缩减一个代理，则会删除 cluster-name-kafka-11。

在从集群中删除代理前，请确保它没有被分配给任何分区。您还应决定剩余的代理中哪些将负责要停用代理中的每个分区。代理没有分配分区后，您可以安全地缩减集群。

2.1.4.2. 分区重新分配

Topic Operator 目前不支持将副本分配给不同的代理，因此需要直接连接到代理 Pod 以将副本分配到代理。

在代理 pod 中，kafka-reassign-partitions.sh 实用程序允许您将分区重新分配给不同的代理。

它有三种不同的模式：

--generate

取一组主题和代理，并生成 重新分配 JSON 文件，该文件将导致这些主题的分区分配给这些代理。由于这适用于整个主题，因此当您只想重新分配某些主题的某些分区时，无法使用它。

--execute

取 重新分配 JSON 文件 并将其应用到集群中的分区和代理。因此获得分区的代理会成为分区领导的追随者。对于给定分区，新代理一旦捕获并加入 ISR（同步副本）后，旧的代理将停止成为追随者，并删除其副本。

--verify

使用与 --execute 步骤相同的 重新分配 JSON 文件，-- verify 检查文件中的所有分区是否已移到其预期代理中。如果重新分配完成后，--verify 也会移除任何有效的 节流。除非被删除，否则节流将继续影响群集，即使重新分配完成后也是如此。

只能在任何给定时间在集群中运行一个重新分配，且无法取消正在运行的重新分配。如果您需要取消重新分配，请等待它完成，然后执行另一个重新分配以恢复第一次重新分配的影响。kafka-reassign-partitions.sh 将显示此重新版本的重新分配 JSON 作为其输出的一部分。在需要停止进行中的重新分配时，应将非常大的重新分配分成几个较小的重新分配。

2.1.4.2.1. 重新分配 JSON 文件

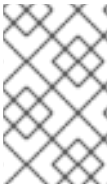
重新分配 JSON 文件 有一个特定的结构：

```
{
  "version": 1,
  "partitions": [
    <PartitionObjects>
  ]
}
```

其中 <PartitionObjects> 是一个用逗号分开的对象列表，例如：

```
{
  "topic": <TopicName>,
  "partition": <Partition>,
```

```
  "replicas": [ <AssignedBrokerIds> ]
}
```



注意

虽然 Kafka 也支持 "log_dirs" 属性，但这不应用于 AMQ Streams。

以下是重新分配 JSON 文件的示例，该文件将 主题 4 的分区 4 分配给代理 2 、 4 和 7，并将 主题 2 的分区 2 分配给 代理 1、5 和 7 ：

```
{
  "version": 1,
  "partitions": [
    {
      "topic": "topic-a",
      "partition": 4,
      "replicas": [2,4,7]
    },
    {
      "topic": "topic-b",
      "partition": 2,
      "replicas": [1,5,7]
    }
  ]
}
```

JSON 中没有包括的分区不会更改。

2.1.4.2.2. 在 JBOD 卷间重新分配分区

在 Kafka 集群中使用 JBOD 存储时，您可以选择在特定卷及其日志目录之间重新分配分区（每个卷都有一个日志目录）。要将分区重新分配给特定卷，请将 log_dirs 选项添加到重新分配 JSON 文件中的 <PartitionObjects> 中。

```
{
  "topic": <TopicName>,
  "partition": <Partition>,
  "replicas": [ <AssignedBrokerIds> ],
  "log_dirs": [ <AssignedLogDirs> ]
}
```

log_dirs 对象应包含与 replicas 对象中指定的副本数相同的日志目录 数量。该值应当是日志目录的绝对路径或 any 关键字。

例如：

```
{
    "topic": "topic-a",
    "partition": 4,
    "replicas": [2,4,7].
    "log_dirs": [ "/var/lib/kafka/data-0/kafka-log2", "/var/lib/kafka/data-0/kafka-log4",
"/var/lib/kafka/data-0/kafka-log7" ]
}
```

2.1.4.3. 生成重新分配 JSON 文件

此流程描述了如何生成重新分配 JSON 文件，该文件使用 kafka-reassign-partitions.sh 工具为给定主题集重新分配所有分区。

先决条件

- 一个正在运行的 Cluster Operator
- Kafka 资源
- 重新分配分区的一组主题

流程

1. 准备名为 topic .json 的 JSON 文件，其中列出要移动的主题。它必须具有以下结构：

```
{
  "version": 1,
  "topics": [
    <TopicObjects>
  ]
}
```

其中 <TopicObjects> 是一个以逗号分隔的对象列表，例如：

```
{
  "topic": <TopicName>
}
```

例如，如果要重新分配 topic-a 和 topic- b 的所有 分区，则需要准备类似以下的 topic .json 文件：

```
{
  "version": 1,
  "topics": [
    { "topic": "topic-a"},
    { "topic": "topic-b"}
  ]
}
```

2. 将 topic.json 文件复制到其中一个代理 pod 中：

```
cat topics.json | oc exec -c kafka <BrokerPod> -i -- \
  /bin/bash -c \
  'cat > /tmp/topics.json'
```

3. 使用 kafka-reassign-partitions.sh 命令生成重新分配 JSON。

```
oc exec <BrokerPod> -c kafka -it -- \
  bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 \
  --topics-to-move-json-file /tmp/topics.json \
  --broker-list <BrokerList> \
  --generate
```

例如，将 topic-a 和 topic- b 的所有分区移动到 broker 4 和 7

```
oc exec <BrokerPod> -c kafka -it -- \
  bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 \
  --topics-to-move-json-file /tmp/topics.json \
  --broker-list 4,7 \
  --generate
```

2.1.4.4. 手动创建重新分配 JSON 文件

如果要移动特定的分区，可以手动创建重新分配 JSON 文件。

2.1.4.5. 重新分配节流

分区重新分配可能是一个缓慢的过程，因为它涉及到在代理之间传输大量数据。为避免对客户端造成重大影响，您可以限制重新分配过程。这可能会导致重新分配需要更长的时间。

- 如果限流太低，新分配的代理将无法与正在发布的记录保持同步，并且重新分配永远不会完成。

- 如果限流太高，客户端就会受到影响。

例如，对于制作者而言，这可能会比等待确认的正常延迟更高。对于消费者而言，这可能会因为轮询之间延迟较高而导致吞吐量下降。

2.1.4.6. 扩展 Kafka 集群

这个步骤描述了如何在 Kafka 集群中增加代理数量。

先决条件

- 现有的 Kafka 集群。

- 名为 reassignment .json 的重新分配 JSON 文件 描述了如何在放大的集群中将分区重新分配给代理。

流程

1. 通过增加 Kafka.spec.kafka.replicas 配置选项，根据需要添加多个新代理。

2. 验证新代理 Pod 是否已启动。

3. 将 reassignment.json 文件复制到您稍后要执行命令的代理 pod 中：

```
cat reassignment.json | \
  oc exec broker-pod -c kafka -i -- /bin/bash -c \
  'cat > /tmp/reassignment.json'
```

例如：

```
cat reassignment.json | \
  oc exec my-cluster-kafka-0 -c kafka -i -- /bin/bash -c \
  'cat > /tmp/reassignment.json'
```

4. 使用同一代理 pod 的 kafka-reassign-partitions.sh 命令行工具执行分区重新分配。

```
oc exec broker-pod -c kafka -it -- \
  bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 \
  --reassignment-json-file /tmp/reassignment.json \
  --execute
```

如果您要节流复制，您也可以使用 --throttle 选项，每秒使用节流率（以字节为单位）。例如：

```
oc exec my-cluster-kafka-0 -c kafka -it -- \
  bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 \
  --reassignment-json-file /tmp/reassignment.json \
  --throttle 5000000 \
  --execute
```

此命令将显示两个重新分配 JSON 对象：第一个记录了正在移动的分区的当前分配。如果稍后需要恢复重新分配，您应该将其保存到本地文件（而不是 pod 中的文件）。第二个 JSON 对象是您在重新分配 JSON 文件中传递的目标重新分配。

5. 如果您需要在重新分配期间更改节流，您可以使用具有不同节流率的同一命令行。例如：

```
oc exec my-cluster-kafka-0 -c kafka -it -- \
  bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 \
  --reassignment-json-file /tmp/reassignment.json \
  --throttle 10000000 \
  --execute
```

6. 定期验证是否使用任何代理 Pod 的 kafka-reassign-partitions.sh 命令行工具完成重新分配。这与上一步中的命令相同，但使用 --verify 选项而不是 --execute 选项。

```
oc exec broker-pod -c kafka -it -- \
  bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 \
  --reassignment-json-file /tmp/reassignment.json \
  --verify
```

例如：

```
oc exec my-cluster-kafka-0 -c kafka -it -- \
  bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 \
  --reassignment-json-file /tmp/reassignment.json \
  --verify
```

7. 当 --verify 命令报告每个分区成功完成时，重新分配已完成。最后的 --verify 还会导致移除任何重新分配节流。现在，如果您保存了 JSON，将分配还原到其原始代理中，您可以删除还原文件。

2.1.4.7. 缩减 Kafka 集群

这个步骤描述了如何减少 Kafka 集群中的代理数量。

先决条件

- 现有的 Kafka 集群。
- 名为 reassignment .json 的重新分配 JSON 文件 描述在 删除了最高编号 Pod 中的代理后，应如何将 分区重新分配给集群中的代理。

流程

1. 将 reassignment.json 文件复制到您稍后要执行命令的代理 pod 中：

```
cat reassignment.json | \
  oc exec broker-pod -c kafka -i -- /bin/bash -c \
  'cat > /tmp/reassignment.json'
```

例如：

```
cat reassignment.json | \
  oc exec my-cluster-kafka-0 -c kafka -i -- /bin/bash -c \
  'cat > /tmp/reassignment.json'
```

2. 使用同一代理 pod 的 kafka-reassign-partitions.sh 命令行工具执行分区重新分配。

```
oc exec broker-pod -c kafka -it -- \
  bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 \
  --reassignment-json-file /tmp/reassignment.json \
  --execute
```

如果您要节流复制，您也可以使用 --throttle 选项，每秒使用节流率（以字节为单位）。例如：

```
oc exec my-cluster-kafka-0 -c kafka -it -- \
  bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 \
  --reassignment-json-file /tmp/reassignment.json \
  --throttle 5000000 \
  --execute
```

此命令将显示两个重新分配 JSON 对象：第一个记录了正在移动的分区的当前分配。如果稍后需要恢复重新分配，您应该将其保存到本地文件（而不是 pod 中的文件）。第二个 JSON 对象是您在重新分配 JSON 文件中传递的目标重新分配。

3. 如果您需要在重新分配期间更改节流，您可以使用具有不同节流率的同一命令行。例如：

```
oc exec my-cluster-kafka-0 -c kafka -it -- \
  bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 \
  --reassignment-json-file /tmp/reassignment.json \
  --throttle 10000000 \
  --execute
```

4. 定期验证是否使用任何代理 Pod 的 kafka-reassign-partitions.sh 命令行工具完成重新分配。这与上一步中的命令相同，但使用 --verify 选项而不是 --execute 选项。

```
oc exec broker-pod -c kafka -it -- \
  bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 \
  --reassignment-json-file /tmp/reassignment.json \
  --verify
```

例如：

```
oc exec my-cluster-kafka-0 -c kafka -it -- \
  bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 \
  --reassignment-json-file /tmp/reassignment.json \
  --verify
```

5. 当 --verify 命令报告每个分区成功完成时，重新分配已完成。最后的 --verify 还会导致移除任何重新分配节流。现在，如果您保存了 JSON，将分配还原到其原始代理中，您可以删除还原文件。

6. 所有分区重新分配完成后，删除的代理不应对集群中的任何分区负责。您可以通过检查代理的数据日志目录是否包含任何实时分区日志来验证这一点。如果代理上的日志目录包含与扩展正

则表达式 [a-zA-Z0-9.-]+\ 不匹配的目录。[a-z0-9]+-delete$ 仍然具有实时分区，不应停止。

您可以执行以下命令检查：

```
oc exec my-cluster-kafka-0 -c kafka -it -- \
 /bin/bash -c \
 "ls -l /var/lib/kafka/kafka-log_<N>_ | grep -E '^d' | grep -vE '[a-zA-Z0-9.-]+\.[a-z0-9]+-delete$'"
```

其中 N 是要删除的 Pod 数。

如果上述命令显示任何输出，则代理仍然有实时分区。在这种情况下，重新分配没有完成，或者重新分配 JSON 文件不正确。

7. 确认代理没有实时分区后，您可以编辑 Kafka.spec.kafka.replicas 资源的 Kafka.spec.kafka.replicas，这会缩减 StatefulSet，从而删除最高数字的代理 Pod。

2.1.5. 用于滚动更新的维护时间窗

通过维护时间窗，您可以计划对 Kafka 和 ZooKeeper 集群进行某些滚动更新，以便在方便的时间启动。

2.1.5.1. 维护时间窗概述

在大多数情况下，Cluster Operator 只更新 Kafka 或 ZooKeeper 集群，以响应对对应 Kafka 资源的更改。这可让您计划何时对 Kafka 资源应用更改，以最大程度降低对 Kafka 客户端应用程序的影响。

但是，在不对 Kafka 资源 进行任何相应的更改的情况下，可能会对 Kafka 和 ZooKeeper 集群进行一些更新。例如，如果 Cluster Operator 管理的 CA（证书授权机构）证书接近到期时间，则 Cluster Operator 将需要执行滚动重启。

虽然 pod 的滚动重启不应影响服务 的可用性 （假设正确的代理和主题配置），这可能会影响 Kafka 客户端应用程序 的性能。通过维护时间窗，您可以调度 Kafka 和 ZooKeeper 集群的自发滚动更新，以便在方便的时间开始。如果没有为群集配置维护时间窗，那么这种自发滚动更新可能会在不方便的时间（如在可预测的高负载期间）发生。

2.1.5.2. 维护时间窗定义

您可以通过在 Kafka.spec.maintenanceTimeWindows 属性中输入字符串数组来配置维护时间窗。每个字符串是一个 cron 表达式，解释为使用 UTC（协调世界时间，其实际用途与 Greenwich Mean Time 相同）。

以下示例配置了一个维护时间窗口，该时间窗口从午夜开始，到 01:59am(UTC)、周日、周一、星期二和周四结束：

```
# ...
maintenanceTimeWindows:
  - "* * 0-1 ? * SUN,MON,TUE,WED,THU *"
# ...
```

在实践中，维护窗口应当与 Kafka.spec.clusterCa.renewalDays 和 Kafka.spec.clientsCa.renewalDays 属性一同设置，以确保在 配置的维护时间窗口中完成必要的 CA 证书续订。

注意

AMQ Streams 不完全根据给定的窗口调度维护操作。相反，对于每个协调，它会检查维护窗口当前是否"打开"。这意味着，给定时间窗内的维护操作启动可能会被 Cluster Operator 协调间隔延迟。因此维护时间窗必须至少是这个长。

其它资源

- 有关 Cluster Operator 配置的更多信息，请参阅 第 5.1.1 节 “Cluster Operator 配置”。

2.1.5.3. 配置维护时间窗

您可以配置维护时间窗，用于由支持的进程触发的滚动更新。

先决条件

- OpenShift 集群。
- Cluster Operator 正在运行。

流程

1.

在 Kafka 资源中添加或编辑 maintenanceTimeWindows 属性。例如，允许在 0800 到 1059 之间以及 1400 到 1559 之间的维护，您可以设置 maintenanceTimeWindows，如下所示：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  maintenanceTimeWindows:
    - "* * 8-10 * * ?"
    - "* * 14-15 * * ?"
```

2. 创建或更新资源：

```
oc apply -f KAFKA-CONFIG-FILE
```

其它资源

执行滚动更新：

- 第 12.3.2 节 “使用 StatefulSet 注解执行滚动更新”
- 第 12.3.3 节 “使用 Pod 注解执行滚动更新”

2.1.6. 从终端连接到 ZooKeeper

大多数 Kafka CLI 工具可以直接连接到 Kafka，因此在正常情况下，您不需要连接到 ZooKeeper。zookeeper 服务通过加密和身份验证进行保护，它们不应该由不属于 AMQ Streams 的外部应用程序使用。

但是，如果要使用需要连接到 ZooKeeper 的 Kafka CLI 工具，您可以使用 ZooKeeper 容器中的终端，并连接 localhost:12181 作为 ZooKeeper 地址。

先决条件

- OpenShift 集群可用。
- Kafka 集群正在运行。
- Cluster Operator 正在运行。

流程

1. 使用 OpenShift 控制台打开终端，或者从 CLI 运行 exec 命令。

   例如：

   ```
   oc exec -ti my-cluster-zookeeper-0 -- bin/kafka-topics.sh --list --zookeeper
   localhost:12181
   ```

   务必使用 localhost:12181。

   现在，您可以向 ZooKeeper 运行 Kafka 命令。

2.1.7. 手动删除 Kafka 节点

此流程描述了如何使用 OpenShift 注解删除现有 Kafka 节点。删除 Kafka 节点包括删除运行 Kafka 代理的 Pod 和相关的 PersistentVolumeClaim （如果集群部署了持久性存储）。删除后，Pod 及其相关的 PersistentVolumeClaim 会被自动重新创建。

警告

删除 PersistentVolumeClaim 可能会导致持久性数据丢失。只有在您遇到存储问题时才应执行以下步骤。

先决条件

有关运行 以下的说明，请参阅 OpenShift 指南中的部署和升级 AMQ Streams ：

- Cluster Operator
- Kafka 集群

流程

1. 查找您要删除的 Pod 的名称。

   例如，如果集群命名为 cluster-name，pod 被命名为 cluster-name-kafka-index，其中 索引 以零开始，结尾是副本总数。

2. 给 OpenShift 中的 容器集资源标注：

   使用 oc annotate:

   ```
   oc annotate pod cluster-name-kafka-index strimzi.io/delete-pod-and-pvc=true
   ```

3. 等待下一次协调，当带有底层持久性卷声明的注解的 pod 被删除后再重新创建。

2.1.8. 手动删除 ZooKeeper 节点

此流程描述了如何使用 OpenShift 注释删除现有的 ZooKeeper 节点。删除 ZooKeeper 节点包括删除运行 ZooKeeper 的 Pod 和相关的 PersistentVolumeClaim （如果集群部署了持久性存储）。删除后，Pod 及其相关的 PersistentVolumeClaim 会被自动重新创建。

警告

删除 PersistentVolumeClaim 可能会导致持久性数据丢失。只有在您遇到存储问题时才应执行以下步骤。

先决条件

有关运行 以下的说明，请参阅 OpenShift 指南中的部署和升级 AMQ Streams ：

- *Cluster Operator*
- *Kafka 集群*

流程

1. *查找您要删除的 Pod 的名称。*

   例如，如果集群名为 cluster-name，pod 被命名为 cluster-name-zookeeper-index，其中索引 从零开始，结尾是副本总数。

2. *给 OpenShift 中的 容器集资源标注：*

   使用 oc annotate:

   ```
   oc annotate pod cluster-name-zookeeper-index strimzi.io/delete-pod-and-pvc=true
   ```

3. *等待下一次协调，当带有底层持久性卷声明的注解的 pod 被删除后再重新创建。*

2.1.9. Kafka 集群资源列表

以下资源由 OpenShift 集群中的 Cluster Operator 创建：

共享资源

cluster-name-cluster-ca

使用用于加密集群通信的集群 CA 私钥的机密。

cluster-name-cluster-ca-cert

使用 Cluster CA 公钥的 secret。这个密钥可以用来验证 Kafka 代理的身份。

cluster-name-clients-ca

带有用于签署用户证书的 Clients CA 私钥的 secret

cluster-name-clients-ca-cert

使用 Clients CA 公钥的机密。此密钥可用于验证 Kafka 用户的身份。

cluster-name-cluster-operator-certs

使用集群操作器密钥与 Kafka 和 ZooKeeper 通信的机密。

zookeeper 节点

cluster-name-zookeeper

StatefulSet，它负责管理 ZooKeeper 节点 pod。

cluster-name-zookeeper-idx

由 Zookeeper StatefulSet 创建的 Pod。

cluster-name-zookeeper-nodes

无头服务需要直接解析 ZooKeeper pod IP 地址。

cluster-name-zookeeper-client

Kafka 代理用于连接 ZooKeeper 节点的服务作为客户端。

cluster-name-zookeeper-config

包含 ZooKeeper 辅助配置的 ConfigMap，由 ZooKeeper 节点 pod 挂载为卷。

cluster-name-zookeeper-nodes

使用 ZooKeeper 节点密钥的机密。

cluster-name-zookeeper

Zookeeper 节点使用的服务帐户。

cluster-name-zookeeper

为 ZooKeeper 节点配置的 Pod Disruption Budget。

cluster-name-network-policy-zookeeper

管理对 ZooKeeper 服务访问的网络策略.

data-cluster-name-zookeeper-idx

用于为 ZooKeeper 节点 pod idx 存储数据的卷的持久性卷声明。只有在选择了持久存储来调配持久卷以存储数据时，才会创建此资源。

Kafka 代理

cluster-name-kafka

StatefulSet，它负责管理 Kafka 代理 pod。

cluster-name-kafka-idx

由 Kafka StatefulSet 创建的 Pod。

cluster-name-kafka-brokers

需要 DNS 直接解析 Kafka 代理 Pod IP 地址的服务。

cluster-name-kafka-bootstrap

服务可用作从 OpenShift 集群内连接的 Kafka 客户端的 bootstrap 服务器。

cluster-name-kafka-external-bootstrap

为从 OpenShift 集群外部连接的客户端引导服务。只有在启用了外部侦听器时才创建此资源。当监听器名称为 外部 并且端口为 9094 时，旧服务名称将用于向后兼容。

cluster-name-kafka-pod-id

用于将流量从 OpenShift 集群外部路由到单个容器集的服务。只有在启用了外部侦听器时才创建此资源。当监听器名称为 外部 并且端口为 9094 时，旧服务名称将用于向后兼容。

cluster-name-kafka-external-bootstrap

从 OpenShift 集群外部连接的客户端的引导路由。只有在启用了外部侦听器并设置为 type route 时才会创建此资源。当监听器名称为 外部 并且端口为 9094 时，旧路由名称将用于向后兼容。

cluster-name-kafka-pod-id

从 OpenShift 集群外部的流量路由到单个容器集。只有在启用了外部侦听器并设置为 type route 时才会创建此资源。当监听器名称为 外部 并且端口为 9094 时，旧路由名称将用于向后兼容。

cluster-name-kafka-listener-name-bootstrap

为从 OpenShift 集群外部连接的客户端引导服务。只有在启用了外部侦听器时才创建此资源。新服务名称将用于所有其他外部侦听器。

cluster-name-kafka-listener-name-pod-id

用于将流量从 OpenShift 集群外部路由到单个容器集的服务。只有在启用了外部侦听器时才创建此资源。新服务名称将用于所有其他外部侦听器。

cluster-name-kafka-listener-name-bootstrap

从 OpenShift 集群外部连接的客户端的引导路由。只有在启用了外部侦听器并设置为 type route 时才会创建此资源。新路由名称将用于所有其他外部侦听器。

cluster-name-kafka-listener-name-pod-id

从 OpenShift 集群外部的流量路由到单个容器集。只有在启用了外部侦听器并设置为 type route 时才会创建此资源。新路由名称将用于所有其他外部侦听器。

cluster-name-kafka-config

包含 Kafka 辅助配置且由 Kafka 代理 Pod 挂载为卷的 ConfigMap。

cluster-name-kafka-brokers

使用 Kafka 代理密钥的 secret。

cluster-name-kafka

Kafka 代理使用的服务帐户。

cluster-name-kafka

为 Kafka 代理配置的 Pod Disruption Budget。

cluster-name-network-policy-kafka

管理对 Kafka 服务访问的网络策略。

strimzi-namespace-name-cluster-name-kafka-init

Kafka 代理使用的集群角色绑定。

cluster-name-jmx

用来保护 Kafka 代理端口的 JMX 用户名和密码的 secret。只有在 Kafka 中启用了 JMX 时才会创建此资源。

data-cluster-name-kafka-idx

用于存储 Kafka 代理 pod IDx 数据的卷的持久性卷声明。只有在选择了持久存储来调配持久卷以存储数据时才创建此资源。

data-id-cluster-name-kafka-idx

用于存储 Kafka 代理 pod ID 的卷 ID 的 持久性卷声明。只有在调配持久卷以存储数据时，才会为 JBOD 卷选择持久存储时创建此资源。

实体 Operator

只有在使用 Cluster Operator 部署 Entity Operator 时才会创建这些资源。

cluster-name-entity-operator

使用主题和用户操作器进行部署.

cluster-name-entity-operator-random-string

由 Entity Operator 部署创建的 Pod。

cluster-name-entity-topic-operator-config

带有主题 Operator 辅助配置的 ConfigMap。

cluster-name-entity-user-operator-config

带有用户 Operator 辅助配置的 ConfigMap.

cluster-name-entity-operator-certs

使用 Entity Operator 密钥与 Kafka 和 ZooKeeper 通信的 secret。

cluster-name-entity-operator

Entity Operator 使用的服务帐户。

strimzi-cluster-name-entity-topic-operator

Entity Topic Operator 使用的角色绑定。

strimzi-cluster-name-entity-user-operator

Entity User Operator 使用的角色绑定。

Kafka Exporter

只有在使用 Cluster Operator 部署 Kafka Exporter 时才会创建这些资源。

cluster-name-kafka-exporter

使用 Kafka 导出器部署.

cluster-name-kafka-exporter-random-string

由 Kafka Exporter 部署创建的 Pod。

cluster-name-kafka-exporter

用于收集消费者滞后指标的服务.

cluster-name-kafka-exporter

Kafka Exporter 使用的服务帐户。

Sything Control

只有在使用 Cluster Operator 部署 Cruise Control 时才会创建这些资源。

cluster-name-cruise-control

通过 Cruise 控制进行部署.

cluster-name-cruise-control-random-string

由 Cruise Control 部署创建的 Pod。

cluster-name-cruise-control-config

包含 Cruise Control 辅助配置的 ConfigMap，并被 Cruise Control pod 作为一个卷挂载。

cluster-name-cruise-control-certs

使用 Cruise Control 密钥与 Kafka 和 ZooKeeper 通信的机密。

cluster-name-cruise-control

用于与 Cruise Control 通信的服务.

cluster-name-cruise-control

Cruise Control 使用的服务帐户.

cluster-name-network-policy-cruise-control

管理对 Cruise 控制服务的访问的网络策略.

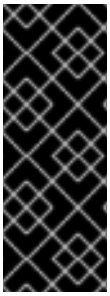
2.2. KAFKA CONNECT/S2I 集群配置

本节论述了如何在 AMQ Streams 集群中配置 Kafka Connect 或 Kafka Connect with Source-to-Image(S2I)部署。

Kafka Connect 是一个集成工具包，用于使用连接器插件在 Kafka 代理和其他系统间流传输数据。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，以使用连接器导入或导出数据。连接器是提供所需连接配置的插件。

如果使用 Kafka Connect，您可以配置 KafkaConnect 或 KafkaConnect S2I 资源。如果您使用 Source-to-Image(S2I)框架来部署 Kafka Connect，请使用 KafkaConnectS2I 资源。

- KafkaConnect 资源的完整 schema 信息包括在 第 13.2.59 节 “KafkaConnect 模式参考” 中。
- KafkaConnectS2I 资源的完整 schema 包括在 第 13.2.84 节 “KafkaConnectS2I 模式参考” 中。



重要

随着 KafkaConnect 资源引入 构建配置，AMQ Streams 现在可以使用数据连接所需的连接器插件自动构建容器镜像。因此，使用 Source-to-Image(S2I)进行 Kafka Connect 的支持已弃用，并将在 AMQ Streams 1.8 后被删除。要准备此更改，您可以将 Kafka Connect S2I 实例迁移到 Kafka Connect 实例。

其它资源

- 创建和管理连接器
- 将 KafkaConnector 资源部署到 Kafka Connect

- 通过注解 KafkaConnector 资源重启 Kafka 连接器
- 通过注解 KafkaConnector 资源重启 Kafka 连接器任务

2.2.1. 配置 Kafka 连接

使用 Kafka Connect 设置到 Kafka 集群的外部数据连接。

使用 KafkaConnect 或 KafkaConnect S2I 资源的属性来配置 Kafka Connect 部署。此流程中演示的示例适用于 KafkaConnect 资源，但 KafkaConnectS2I 资源的属性相同。

Kafka 连接器配置

KafkaConnector 资源允许您以 OpenShift 原生方式为 Kafka Connect 创建和管理连接器实例。

在 Kafka Connect 配置中，您可以通过添加 strimzi.io/use-connector-resources 注解为 Kafka Connect 集群启用 KafkaConnectors。您还可以添加 构建配置，以便 AMQ Streams 会自动构建带有数据连接所需的连接器插件的容器镜像。Kafka Connect 连接器的外部配置通过 externalConfiguration 属性指定。

要管理连接器，您可以使用 Kafka Connect REST API，或使用 KafkaConnector 自定义资源。KafkaConnector 资源必须部署到与其链接的 Kafka Connect 集群相同的命名空间中。有关使用这些方法创建、重新配置或删除连接器的更多信息，请参阅 OpenShift 上部署和升级 AMQ Streams 中的创建和管理连接器。

连接器配置作为 HTTP 请求的一部分传递给 Kafka Connect，并存储在 Kafka 本身中。ConfigMap 和机密是用于存储配置和机密数据的标准 OpenShift 资源。您可以使用 ConfigMap 和 Secret 配置连接器的某些元素。然后，您可以引用 HTTP REST 命令中的配置值，以便在需要时保持配置独立和安全。这个方法尤其适用于机密数据，如用户名、密码或证书。

先决条件

- OpenShift 集群
- 一个正在运行的 Cluster Operator

有关运行 以下的说明，请参阅 OpenShift 指南中的部署和升级 AMQ Streams ：

- *Cluster Operator*
- *Kafka 集群*

流程

1. *编辑 KafkaConnect 或 KafkaConnect S2I 资源的 spec 属性。*

   您可以配置的属性显示在此示例配置中：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect (1)
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" (2)
spec:
  replicas: 3 (3)
  authentication: (4)
    type: tls
    certificateAndKey:
      certificate: source.crt
      key: source.key
      secretName: my-user-source
  bootstrapServers: my-cluster-kafka-bootstrap:9092 (5)
  tls: (6)
    trustedCertificates:
      - secretName: my-cluster-cluster-cert
        certificate: ca.crt
      - secretName: my-cluster-cluster-cert
        certificate: ca2.crt
  config: (7)
    group.id: my-connect-cluster
    offset.storage.topic: my-connect-cluster-offsets
    config.storage.topic: my-connect-cluster-configs
    status.storage.topic: my-connect-cluster-status
    key.converter: org.apache.kafka.connect.json.JsonConverter
    value.converter: org.apache.kafka.connect.json.JsonConverter
    key.converter.schemas.enable: true
    value.converter.schemas.enable: true
    config.storage.replication.factor: 3
    offset.storage.replication.factor: 3
    status.storage.replication.factor: 3
  build: (8)
```

```
    output: 9
      type: docker
      image: my-registry.io/my-org/my-connect-cluster:latest
      pushSecret: my-registry-credentials
    plugins: 10
      - name: debezium-postgres-connector
        artifacts:
          - type: tgz
            url: https://repo1.maven.org/maven2/io/debezium/debezium-connector-
postgres/1.3.1.Final/debezium-connector-postgres-1.3.1.Final-plugin.tar.gz
            sha512sum:
962a12151bdf9a5a30627eebac739955a4fd95a08d373b86bdcea2b4d0c27dd6e1edd5cb5
48045e115e33a9e69b1b2a352bee24df035a0447cb820077af00c03
      - name: camel-telegram
        artifacts:
          - type: tgz
            url:
https://repo.maven.apache.org/maven2/org/apache/camel/kafkaconnector/camel-
telegram-kafka-connector/0.7.0/camel-telegram-kafka-connector-0.7.0-package.tar.gz
            sha512sum:
a9b1ac63e3284bea7836d7d24d84208c49cdf5600070e6bd1535de654f6920b74ad950d517
33e8020bf4187870699819f54ef5859c7846ee4081507f48873479
  externalConfiguration: 11
    env:
      - name: AWS_ACCESS_KEY_ID
        valueFrom:
          secretKeyRef:
            name: aws-creds
            key: awsAccessKey
      - name: AWS_SECRET_ACCESS_KEY
        valueFrom:
          secretKeyRef:
            name: aws-creds
            key: awsSecretAccessKey
  resources: 12
    requests:
      cpu: "1"
      memory: 2Gi
    limits:
      cpu: "2"
      memory: 2Gi
  logging: 13
    type: inline
    loggers:
      log4j.rootLogger: "INFO"
  readinessProbe: 14
    initialDelaySeconds: 15
    timeoutSeconds: 5
  livenessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 5
  metricsConfig: 15
    type: jmxPrometheusExporter
    valueFrom:
      configMapKeyRef:
```

```
      name: my-config-map
      key: my-key
jvmOptions: 16
  "-Xmx": "1g"
  "-Xms": "1g"
image: my-org/my-image:latest 17
rack:
  topologyKey: topology.kubernetes.io/zone 18
template: 19
  pod:
    affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
                - key: application
                  operator: In
                  values:
                    - postgresql
                    - mongodb
            topologyKey: "kubernetes.io/hostname"
  connectContainer: 20
    env:
      - name: JAEGER_SERVICE_NAME
        value: my-jaeger-service
      - name: JAEGER_AGENT_HOST
        value: jaeger-agent-name
      - name: JAEGER_AGENT_PORT
        value: "6831"
```

1

根据需要，使用 KafkaConnect 或 KafkaConnectS2I。

2

为 Kafka Connect 集群启用 KafkaConnectors。

3

副本节点的数量。

4

Kafka Connect 集群的身份验证，使用 TLS 机制 （如下所示）、使用 OAuth bearer 令牌 或基于 SASL 的 SCRAM-SHA-512 或 PLAIN 机制。默认情况下，Kafka Connect 使用纯文本连接连接到 Kafka 代理。

5

用于连接到 Kafka Connect 集群的 bootstrap 服务器。

6

使用密钥名称进行 TLS 加密，在其下，TLS 证书以 X.509 格式存储到集群的 X.509 格式。如果证书存储在同一个 secret 中，则可以多次列出证书。

7

Kafka Connect worker （而不是连接器）配置。标准 Apache Kafka 配置可能会提供，仅限于不是由 AMQ Streams 直接管理的属性。

8

用于自动使用连接器插件构建容器镜像的 构建配置属性。

9

（必需）配置推送新镜像的容器注册表。

10

（必需）要添加到新容器镜像的连接器插件及其工件列表。每个插件必须至少配置一个 工件。

11

使用环境变量（如下所示）或卷 为 Kafka 连接器的外部配置。您还可以使用 Kubernetes 配置提供程序 从外部来源加载配置值。

12

13

指定 Kafka 连接日志记录器和日志级别 直接（内联）或通过ConfigMap 间接（外部）添加。自定义 ConfigMap 必须放在 log4j.properties 或 log4j2.properties 键下。对于 Kafka Connect log4j.rootLogger 日志记录器，您可以将日志级别设置为 INFO、ERROR、WARN、TRACE、DEBUG、FATAL 或 OFF。

14

健康检查以了解 何时重新启动容器（存活度）以及容器何时可以接受流量（就绪度）。

15

Prometheus metrics，通过引用包含本例中 Prometheus JMX 导出器配置的 ConfigMap 来启用它。您可以使用对包含 metricsConfig.valueFrom.configMapKeyRef.key 下的空文件的 ConfigMap 的引用来启用指标，而无需进一步配置。

16 运行 Kafka Connect 的虚拟机(VM)性能优化 JVM 配置选项。

17 ADVANCED OPTION： 容器镜像配置，只在特殊情况下推荐这样做。

18 将机架感知 配置为在不同机架之间分布副本。topologykey 必须与集群节点标签匹配。

19 模板自定义.在这里，pod 被调度为反关联性，因此 pod 不会调度到具有相同主机名的节点。

20 还 使用 Jaeger 为分布式追踪设置 环境变量。

2. 创建或更新资源：

```
oc apply -f KAFKA-CONNECT-CONFIG-FILE
```

3. 如果为 Kafka Connect 启用了授权， 请配置 Kafka Connect 用户以启用对 Kafka Connect consumer 组和主题的访问。

2.2.2. 多个实例的 Kafka Connect 配置

如果您正在运行多个 Kafka Connect 实例，您必须更改以下配置 属性 的默认配置：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    group.id: connect-cluster 1
```

```
    offset.storage.topic: connect-cluster-offsets 2
    config.storage.topic: connect-cluster-configs 3
    status.storage.topic: connect-cluster-status 4
    # ...
# ...
```

1

属于该实例的 Kafka Connect 集群组。

2

存储连接器偏移的 Kafka 主题。

3

存储连接器和任务状态配置的 Kafka 主题。

4

存储连接器和任务状态更新的 Kafka 主题。

注意

对于具有同一 group.id 的所有 Kafka Connect 实例，三个主题的值必须相同。

除非更改默认设置，否则每个连接到同一 Kafka 集群的 Kafka Connect 实例都使用相同的值部署。实际上，所有实例都合并在一起，在集群中运行并使用相同的主题。

如果多个 Kafka Connect 集群尝试使用相同的主题，Kafka Connect 将无法按预期工作并生成错误。

如果要运行多个 Kafka Connect 实例，请为每个实例更改这些属性的值。

2.2.3. 配置 Kafka Connect 用户授权

这个步骤描述了如何授权用户对 Kafka Connect 的访问。

当 Kafka 中使用任何类型的授权时，Kafka Connect 用户需要对消费者组的读/写访问权限以及 Kafka Connect 的内部主题。

用户组和内部主题的属性由 AMQ Streams 自动配置，或者可以在 KafkaConnect 或 KafkaConnect S2I 资源的 spec 中明确指定。

KafkaConnect 资源中的配置属性示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
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
    # ...
  # ...
```

1 属于该实例的 Kafka Connect 集群组。

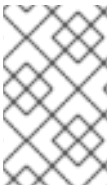
2 存储连接器偏移的 Kafka 主题。

3 存储连接器和任务状态配置的 Kafka 主题。

4 存储连接器和任务状态更新的 Kafka 主题。

此流程演示了在使用 简单 授权时如何提供访问权限。

简单的授权使用由 Kafka AclAuthorizer 插件处理的 ACL 规则来提供正确的访问级别。有关将 KafkaUser 资源配置为使用简单授权的更多信息，请参阅 AclRule schema 引用。



注意

运行多个实例时，使用者组和主题的默认值将有所不同。

先决条件

- OpenShift 集群
- 一个正在运行的 Cluster Operator

流程

1. 编辑 KafkaUser 资源中 的授权 属性，以便为用户提供访问权限。

   在以下示例中，使用 字面 名称值为 Kafka Connect 主题和消费者组配置访问权限：

属性	名称
offset.storage.topic	connect-cluster-offsets
status.storage.topic	connect-cluster-status
config.storage.topic	connect-cluster-configs
group	connect-cluster

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  # ...
  authorization:
    type: simple
    acls:
      # access to offset.storage.topic
      - resource:
```

```
    type: topic
    name: connect-cluster-offsets
    patternType: literal
  operation: Write
  host: "*"
- resource:
    type: topic
    name: connect-cluster-offsets
    patternType: literal
  operation: Create
  host: "*"
- resource:
    type: topic
    name: connect-cluster-offsets
    patternType: literal
  operation: Describe
  host: "*"
- resource:
    type: topic
    name: connect-cluster-offsets
    patternType: literal
  operation: Read
  host: "*"
# access to status.storage.topic
- resource:
    type: topic
    name: connect-cluster-status
    patternType: literal
  operation: Write
  host: "*"
- resource:
    type: topic
    name: connect-cluster-status
    patternType: literal
  operation: Create
  host: "*"
- resource:
    type: topic
    name: connect-cluster-status
    patternType: literal
  operation: Describe
  host: "*"
- resource:
    type: topic
    name: connect-cluster-status
    patternType: literal
  operation: Read
  host: "*"
# access to config.storage.topic
- resource:
    type: topic
    name: connect-cluster-configs
    patternType: literal
  operation: Write
  host: "*"
- resource:
```

```
      type: topic
      name: connect-cluster-configs
      patternType: literal
    operation: Create
    host: "*"
  - resource:
      type: topic
      name: connect-cluster-configs
      patternType: literal
    operation: Describe
    host: "*"
  - resource:
      type: topic
      name: connect-cluster-configs
      patternType: literal
    operation: Read
    host: "*"
  # consumer group
  - resource:
      type: group
      name: connect-cluster
      patternType: literal
    operation: Read
    host: "*"
```

2. 创建或更新资源。

```
oc apply -f KAFKA-USER-CONFIG-FILE
```

2.2.4. 执行 Kafka 连接器重启

此流程描述了如何使用 OpenShift 注解手动触发 Kafka 连接器重启。

先决条件

- Cluster Operator 正在运行。

流程

1. 查找控制您要重启的 Kafka 连接器的 KafkaConnector 自定义资源的名称：

```
oc get KafkaConnector
```

2. 要重启连接器，请注解 OpenShift 中的 KafkaConnector 资源。例如，使用 oc annotate：

```
oc annotate KafkaConnector KAFKACONNECTOR-NAME strimzi.io/restart=true
```

3. 等待下一次协调发生（默认为每隔两分钟）。

Kafka 连接器重启，只要协调过程检测到注解。当 Kafka Connect 接受重启请求时，该注解会从 KafkaConnector 自定义资源中删除。

其它资源

- 在部署和管理连接器 指南中 创建和管理连接器.

2.2.5. 执行 Kafka 连接器任务重启

此流程描述了如何使用 OpenShift 注解手动触发 Kafka 连接器任务重启。

先决条件

- Cluster Operator 正在运行。

流程

1. 查找控制您要重启的 Kafka 连接器任务的 KafkaConnector 自定义资源的名称：

```
oc get KafkaConnector
```

2. 从 KafkaConnector 自定义资源中找到要重启的任务 ID。任务 ID 是非负整数，从 0 开始。

```
oc describe KafkaConnector KAFKACONNECTOR-NAME
```

3. 要重启连接器任务，请注解 OpenShift 中的 KafkaConnector 资源。例如，使用 oc annotate 重启任务 0:

```
oc annotate KafkaConnector KAFKACONNECTOR-NAME strimzi.io/restart-task=0
```

4. 等待下一次协调发生（默认为每隔两分钟）。

Kafka 连接器任务重启，只要协调过程检测到注解。当 Kafka Connect 接受重启请求时，该注解会从 KafkaConnector 自定义资源中删除。

其它资源

- 在部署和管理连接器 指南中 创建和管理连接器.

2.2.6. 从使用 S2I 的 Kafka Connect 迁移到 Kafka Connect

对 Kafka Connect with S2I 和 KafkaConnectS2I 资源的支持已弃用。这会在 KafkaConnect 资源引入 构建配置 属性，用于使用连接器插件来自动构建容器镜像。

这个步骤描述了如何使用 S2I 实例将 Kafka Connect 迁移到标准 Kafka Connect 实例。为此，您需要配置一个新的 KafkaConnect 自定义资源来替换 KafkaConnectS2I 资源，然后删除该资源。

警告

迁移过程涉及从 KafkaConnectS2I 实例删除到新 KafkaConnect 实例部署成功时的停机时间。在此期间，连接器将不会运行和处理数据。但是，更改后，它们应从停止的时间点继续。

先决条件

- 使用 KafkaConnectS2I 配置部署带有 S 2I 的 Kafka Connect
- 与 S2I 的 Kafka Connect 使用带有使用 S2I 构建添加的连接器的镜像
- sink 和源连接器实例是使用 KafkaConnector 资源或 Kafka Connect REST API 创建的

流程

1. 使用与用于 Kafka connectS2I 资源的名称相同的名称创建新的 Kafka Connect 自定义资源。

2. 将 KafkaConnectS2I 资源属性复制到 KafkaConnect 资源。

3. 如果指定，请确保使用相同的 spec.config 属性：

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

   如果未指定这些属性，则使用默认值。在这种情况下，也将其从 KafkaConnect 资源配置中退出。

   现在，将特定于 KafkaConnect 资源的配置添加到新资源中。

4. 添加 构建配置，以配置您要添加到 Kafka Connect 部署中的所有连接器和其他库。

   注意

   或者，您也可以使用连接器手动构建新镜像，并使用 .spec.image 属性指定它。

5. 删除旧的 KafkaConnectS2I 资源：

   ```
   oc delete -f MY-KAFKA-CONNECT-S2I-CONFIG-FILE
   ```

   将 MY-KAFKA-CONNECT-S2I-CONFIG-FILE 替换为包含 KafkaConnectS2I 资源配置的文件名。

另外，您可以指定资源的名称：

```
oc delete kafkaconnects2i MY-KAFKA-CONNECT-S2I
```

将 MY-KAFKA-CONNECT-S2I 替换为 KafkaConnectS2I 资源的名称。

等待 Kafka Connect with S2I 部署被删除，并且 pod 已被删除。

警告

不得删除任何其他资源。

6. 部署新的 KafkaConnect 资源：

```
oc apply -f MY-KAFKA-CONNECT-CONFIG-FILE
```

将 MY-KAFKA-CONNECT-CONFIG-FILE 替换为包含新 KafkaConnect 资源配置的文件名称。

等待新镜像构建好，部署创建好，并且容器集已启动。

7. 如果您使用 KafkaConnector 资源来管理 Kafka Connect 连接器，请检查所有预期的连接器是否存在并正在运行：

```
oc get kctr --selector strimzi.io/cluster=MY-KAFKA-CONNECT-CLUSTER -o name
```

将 MY-KAFKA-CONNECT-CLUSTER 替换为 Kafka Connect 集群的名称。

通过 Kafka Connect 存储自动恢复连接器。即使您使用 Kafka Connect REST API 管理它们，您不应该手动重新创建它们。

其它资源

- *配置 Kafka 连接*
- *使用 AMQ Streams 自动创建新容器镜像*
- *从 Kafka Connect 基础镜像创建 Docker 镜像*
- *创建和管理连接器*

2.2.7. Kafka Connect 集群资源列表

以下资源由 OpenShift 集群中的 Cluster Operator 创建：

connect-cluster-name-connect

用于创建 Kafka Connect worker 节点 pod 的 Deployment。

connect-cluster-name-connect-api

公开用于管理 Kafka Connect 集群的 REST 接口的服务。

connect-cluster-name-config

包含 Kafka Connect 辅助配置且由 Kafka 代理 Pod 挂载为卷的 ConfigMap。

connect-cluster-name-connect

为 Kafka Connect worker 节点配置的 Pod Disruption Budget。

2.2.8. Kafka Connect(S2I)集群资源列表

以下资源由 OpenShift 集群中的 Cluster Operator 创建：

connect-cluster-name-connect-source

镜像流，用作新构建的 Docker 镜像的基础镜像。

connect-cluster-name-connect

BuildConfig，它负责构建新的 Kafka Connect Docker 镜像。

connect-cluster-name-connect

将推送新构建 Docker 镜像的镜像流。

connect-cluster-name-connect

DeploymentConfig，它负责创建 Kafka Connect worker 节点 pod。

connect-cluster-name-connect-api

公开用于管理 Kafka Connect 集群的 REST 接口的服务。

connect-cluster-name-config

包含 Kafka Connect 辅助配置且由 Kafka 代理 Pod 挂载为卷的 ConfigMap。

connect-cluster-name-connect

为 Kafka Connect worker 节点配置的 Pod Disruption Budget。

2.2.9. 与 Debezium 集成以捕获更改数据

红帽 Debezium 是一个分布式更改数据捕获平台。它捕获数据库中的行级更改，创建更改事件记录，并将记录流传输到 Kafka 主题。Debezium 基于 Apache Kafka 构建。您可以将 Debezium 与 AMQ Streams 部署和集成。部署 AMQ Streams 后，您可以通过 Kafka Connect 将 Debezium 部署为连接器配置。Debezium 将更改事件记录传递到 OpenShift 上的 AMQ Streams。应用程序可以读取 这些更改事件流，并按发生更改事件的顺序访问更改事件。

Debezium 具有多个用途，包括：

- *数据复制*
- *更新缓存和搜索索引*
- *简化单体式应用程序*

- 数据集成
- 启用流查询

要捕获数据库更改，请使用 Debezium 数据库连接器部署 Kafka Connect。您可以配置 KafkaConnector 资源来定义连接器实例。

有关使用 AMQ Streams 部署 Debezium 的更多信息，请参阅产品文档。Debezium 文档包括 Debezium 入门指南，指导您完成设置服务和查看数据库更新事件记录所需的连接器所需的流程。

2.3. KAFKA MIRRORMAKER 集群配置

本章论述了如何在 AMQ Streams 集群中配置 Kafka 镜像Maker 部署以在 Kafka 集群间复制数据。

您可以使用带有 MirrorMaker 或 MirrorMaker 2.0 的 AMQ Streams。MirrorMaker 2.0 是最新版本，为在 Kafka 集群间镜像数据提供了一种更有效的方法。

如果使用 MirrorMaker，您需要配置 KafkaMirrorMaker 资源。

以下流程演示了如何配置资源：

- 配置 Kafka MirrorMaker

KafkaMirrorMaker 资源的完整 schema 信息包括在 KafkaMirrorMaker schema 引用 中。

2.3.1. Configuring Kafka MirrorMaker

使用 KafkaMirrorMaker 资源的属性来配置 Kafka MirrorMaker 部署。

您可以使用 TLS 或 SASL 身份验证为生产者和使用者配置访问控制。此流程演示了在使用者和制作者侧使用 TLS 加密和身份验证的配置。

先决条件

- *有关运行 以下的说明，请参阅 OpenShift 指南中的部署和升级 AMQ Streams ：*
  - *Cluster Operator*
  - *Kafka 集群*
- *源和目标 Kafka 集群必须可用*

流程

1. *编辑 KafkaMirrorMaker 资源的 spec 属性。*

   您可以配置的属性显示在此示例配置中：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker
metadata:
  name: my-mirror-maker
spec:
  replicas: 3 (1)
  consumer:
    bootstrapServers: my-source-cluster-kafka-bootstrap:9092 (2)
    groupId: "my-group" (3)
    numStreams: 2 (4)
    offsetCommitInterval: 120000 (5)
    tls: (6)
      trustedCertificates:
      - secretName: my-source-cluster-ca-cert
        certificate: ca.crt
    authentication: (7)
      type: tls
      certificateAndKey:
        secretName: my-source-secret
        certificate: public.crt
        key: private.key
    config: (8)
      max.poll.records: 100
      receive.buffer.bytes: 32768
      ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" (9)
      ssl.enabled.protocols: "TLSv1.2"
      ssl.protocol: "TLSv1.2"
      ssl.endpoint.identification.algorithm: HTTPS (10)
```

```
producer:
  bootstrapServers: my-target-cluster-kafka-bootstrap:9092
  abortOnSendFailure: false (11)
  tls:
    trustedCertificates:
    - secretName: my-target-cluster-ca-cert
      certificate: ca.crt
  authentication:
    type: tls
    certificateAndKey:
      secretName: my-target-secret
      certificate: public.crt
      key: private.key
  config:
    compression.type: gzip
    batch.size: 8192
    ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" (12)
    ssl.enabled.protocols: "TLSv1.2"
    ssl.protocol: "TLSv1.2"
    ssl.endpoint.identification.algorithm: HTTPS (13)
include: "my-topic|other-topic" (14)
resources: (15)
  requests:
    cpu: "1"
    memory: 2Gi
  limits:
    cpu: "2"
    memory: 2Gi
logging: (16)
  type: inline
  loggers:
    mirrormaker.root.logger: "INFO"
readinessProbe: (17)
  initialDelaySeconds: 15
  timeoutSeconds: 5
livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
metricsConfig: (18)
 type: jmxPrometheusExporter
 valueFrom:
   configMapKeyRef:
     name: my-config-map
     key: my-key
jvmOptions: (19)
  "-Xmx": "1g"
  "-Xms": "1g"
image: my-org/my-image:latest (20)
template: (21)
  pod:
    affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
```

```
            - key: application
              operator: In
              values:
                - postgresql
                - mongodb
          topologyKey: "kubernetes.io/hostname"
  connectContainer: 22
    env:
      - name: JAEGER_SERVICE_NAME
        value: my-jaeger-service
      - name: JAEGER_AGENT_HOST
        value: jaeger-agent-name
      - name: JAEGER_AGENT_PORT
        value: "6831"
  tracing: 23
    type: jaeger
```

1

副本节点的数量。

2

为 使用者和生产者引导服务器.

3

消费者的组 ID。

4

消费者流的数量。

5

以毫秒为单位的偏移自动提交间隔。

6

使用密钥名称进行 TLS 加密，其中 TLS 证书存储在 X.509 格式供使用者或生产者使用。如果证书存储在同一个 secret 中，则可以多次列出证书。

7

对使用者或生产者进行身份验证，使用此处所示的 TLS 机制，使用 OAuth bearer 令牌 或基于 SASL 的 SCRAM-SHA-512 或 PLAIN 机制。

8

9

使用特定 密码套件 为 TLS 版本运行外部侦听器的 SSL 属性。

10

通过将 设置为 HTTPS 来 启用主机名验证。空字符串将禁用验证。

11

如果 abort OnSendFailure 属性 设为 true，则 Kafka MirrorMaker 将退出，容器将在发送失败消息后重启。

12

使用特定 密码套件 为 TLS 版本运行外部侦听器的 SSL 属性。

13

通过将 设置为 HTTPS 来 启用主机名验证。空字符串将禁用验证。

14

包括的主题 （从源镜像到目标 Kafka 集群）。

15

请求保留 支持的资源、当前 cpu 和 memory，以及限制，以指定可消耗的最大资源。

16

指定日志记录器和日志级别 通过 ConfigMap 直接（内联）或间接（外部）添加。自定义 ConfigMap 必须放在 log4j.properties 或 log4j2.properties 键下。MirrorMaker 有一个名为 mirrormaker.root.logger 的日志记录器。您可以将日志级别设置为 INFO、ERROR、WARN、TRACE、DEBUG、FATAL 或 OFF。

17

健康检查以了解 何时重新启动容器（存活度）以及容器何时可以接受流量（就绪度）。

18

Prometheus metrics，通过引用包含本例中 Prometheus JMX 导出器配置的 ConfigMap 来启用它。您可以使用对包含

metricsConfig.valueFrom.configMapKeyRef.key 下的空文件的 ConfigMap 的引用来启用指标，而无需进一步配置。

19 用于优化运行 Kafka MirrorMaker 的虚拟机(VM)性能的 JVM 配置选项。

20 ADVANCED OPTION： 容器镜像配置，只在特殊情况下推荐这样做。

21 模板自定义.在这里，pod 被调度为反关联性，因此 pod 不会调度到具有相同主机名的节点。

22 还 使用 Jaeger 为分布式追踪设置 环境变量。

23 为 Jaeger 启用分布式追踪。

> **警告**
>
> 将 abort OnSendFailure 属性设置为 false 时，制作者会尝试发送主题中的下一个消息。原始消息可能会丢失，因为没有尝试重新发送失败的消息。

2. 创建或更新资源：

```
oc apply -f <your-file>
```

2.3.2. Kafka MirrorMaker 集群资源列表

以下资源由 OpenShift 集群中的 Cluster Operator 创建：

<mirror-maker-name>-mirror-maker

负责创建 Kafka MirrorMaker pod 的部署。

<mirror-maker-name>-config

包含 Kafka MirrorMaker 的辅助配置的 ConfigMap，并由 Kafka 代理 Pod 挂载为卷。

<mirror-maker-name>-mirror-maker

为 Kafka MirrorMaker worker 节点配置的 Pod Disruption Budget。

2.4. KAFKA MIRRORMAKER 2.0 集群配置

本节论述了如何在 AMQ Streams 集群中配置 Kafka MirrorMaker 2.0 部署。

MirrorMaker 2.0 用于在数据中心内或之间的两个或多个活跃 Kafka 集群之间复制数据。

集群间的数据复制支持以下条件：

- 发生系统故障时恢复数据
- 聚合数据以进行分析
- 对特定集群的数据访问的限制
- 在特定位置置备数据以缩短延迟

如果使用 MirrorMaker 2.0，您需要配置 KafkaMirrorMaker2 资源。

MirrorMaker 2.0 引入了一种全新的在集群间复制数据的方法。

因此，资源配置与以前的 MirrorMaker 版本不同。如果您选择使用 MirrorMaker 2.0，当前没有传统支持，因此任何资源都必须手动转换为新格式。

MirrorMaker 2.0 复制数据的方式如下：

- MirrorMaker 2.0 数据复制

以下流程演示了如何为 MirrorMaker 2.0 配置资源：

- 在 Kafka 集群间同步数据

KafkaMirrorMaker2 资源的完整 schema 包括在 KafkaMirrorMaker2 架构引用 中。

2.4.1. MirrorMaker 2.0 数据复制

MirrorMaker 2.0 使用源 Kafka 集群的信息，并将其写入目标 Kafka 集群。

MirrorMaker 2.0 使用：

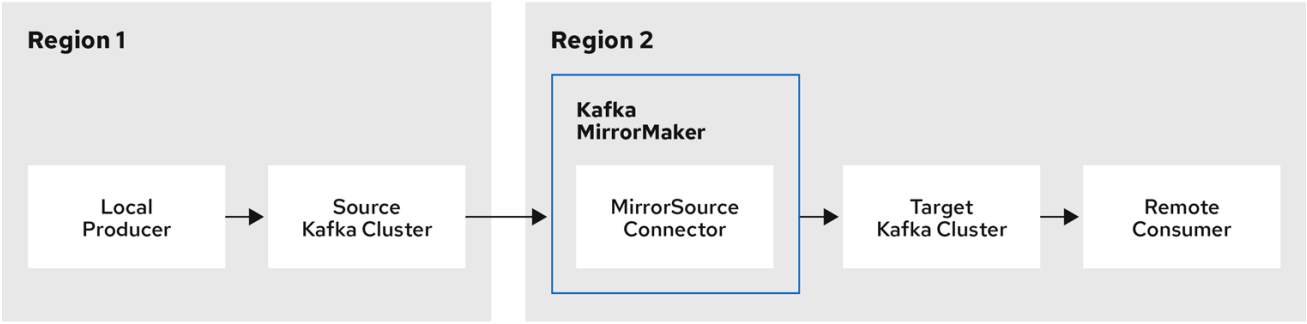
- 源集群配置以使用源集群的数据
- 将数据输出到目标集群的目标集群配置

MirrorMaker 2.0 基于 Kafka Connect 框架，即管理集群间数据传输的 连接器。MirrorMaker 2.0 MirrorSourceConnector 将主题从源集群复制到目标集群。

将数据从一个集群 镜像 到另一个集群的过程是异步的。建议的模式是让信息与源 Kafka 集群一起生成，然后远程使用与目标 Kafka 集群类似的信息。

MirrorMaker 2.0 可以用于多个源集群。

图 2.1. 在两个集群间复制



默认情况下，每 10 分钟检查源集群中的新主题。您可以通过在源连接器配置中添加 refresh.topics.interval.seconds 来更改频率。但是，增加操作的频率可能会影响整体性能。

2.4.2. 集群配置

您可以在 主动/ 被动或主动/主动 集群配置中使用 MirrorMaker 2.0。

- 在 主动/主动 配置中，两个集群都处于活动状态并同时提供相同的数据，如果您想在不同的地理位置在本地提供相同的数据，这很有用。

- 在 主动/被动 配置中，来自主动/被动群集的数据复制到被动群集中，该群集仍处于备用状态，例如，在发生系统故障时进行数据恢复。

预计生产者和消费者只能连接到活跃的集群。

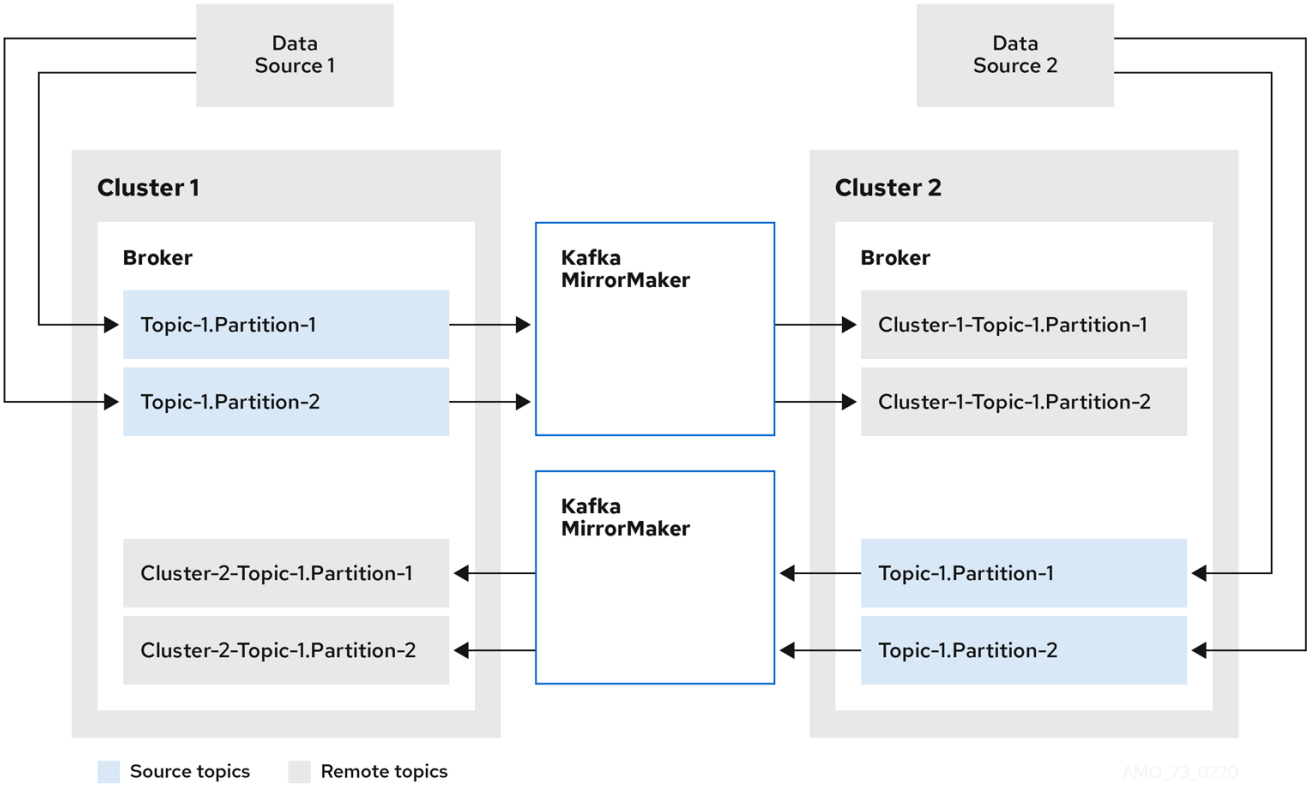
每个目标目的地都需要一个 MirrorMaker 2.0 集群。

2.4.2.1. 双向复制（主动/主动）

MirrorMaker 2.0 架构支持 主动/主动 集群配置中的双向复制。

每个集群使用 源 和远程 主题的概念复制其他集群的数据。由于每个集群中存储了相同的主题，因此 MirrorMaker 2.0 会自动重命名远程主题来代表源集群。原始集群的名称前面是主题名称的前面。

图 2.2. 主题重命名



通过标记原始集群，主题不会复制到该集群中。

在配置需要数据聚合的架构时，通过 远程 主题复制的概念非常有用。消费者可以订阅同一群集内的源和远程主题，无需单独的聚合集群。

2.4.2.2. 单向复制（主动/被动）

MirrorMaker 2.0 架构支持 主动/被动 集群配置中的单向复制。

您可以使用 主动/被动 集群配置进行备份，或将数据迁移到另一个集群。在这种情况下，您可能不希望自动重命名远程主题。

您可以通过在源连接器配置中添加 IdentityReplicationPolicy 来覆盖自动重命名。应用此配置后，主题会保留其原始名称。

2.4.2.3. 主题配置同步

主题配置会在源集群和目标集群之间自动同步。通过同步配置属性，可以减少重新平衡的需求。

2.4.2.4. 数据完整性

MirrorMaker 2.0 监控源主题，并将配置更改传播到远程主题，检查并创建缺少的分区。只有 MirrorMaker 2.0 可以写入远程主题。

2.4.2.5. 偏移跟踪

MirrorMaker 2.0 使用 内部主题 跟踪消费者组的偏移量。

- 偏移同步 主题映射从记录元数据中复制主题分区的来源和目标偏移
- checkpoint 主题映射源和目标集群中为各个消费者组中复制主题分区的最后提交偏移

检查点 主题的偏移会通过配置预先确定的时间间隔进行跟踪。这两个主题都允许从故障转移上的正确偏移位置完全恢复复制。

MirrorMaker 2.0 使用其 MirrorCheckpointConnector 来发送 检查点 以进行偏移。

2.4.2.6. 同步消费者组偏移

__consumer_offsets 主题存储关于每个消费者组已提交的偏移的信息。偏移同步定期将源集群使用者组的使用者偏移量传输到目标集群的使用者偏移主题。

偏移同步在 主动/被动 配置中特别有用。如果活跃集群停机，使用者应用可以切换到被动(standby)集群，并从最近传输的偏移位置获取。

使用主题偏移同步：

- 通过将 sync. group.offsets.enabled 添加到 checkpoint 连接器配置来启用同步，并将 属性设置为 true。默认情况下禁用同步。
- 将 IdentityReplicationPolicy 添加到源和检查点连接器配置中，以便目标集群中的主题保留其原始名称。

要进行主题偏移同步，目标集群中的消费者组无法使用与源集群中的组相同的 id。

如果启用，则定期与源集群进行偏移同步。您可以通过将 sync. group.offsets.interval.seconds 和 emit.checkpoints.interval.seconds 添加到 checkpoint 连接器配置来更改频率。属性指定消费者组偏移同步的频率（以秒为单位），以及发出用于偏移跟踪的检查点频率。两个属性的默认值都是 60 秒。您还可以使用 refresh. groups.interval.seconds 属性更改新使用者 组的检查频率，默认为每 10 分钟执行一次。

由于同步是基于时间的，因此使用者到被动群集的任何切换都可能会导致消息出现某种重复。

2.4.2.7. 连接检查

一个 心跳 的内部主题检查集群之间的连通性。

heartbeat 主题从源集群中复制。

目标集群使用该主题检查：

- 集群之间的连接器正在运行
- 源集群可用

MirrorMaker 2.0 使用其 MirrorHeartbeatConnector 发送执行这些检查 的心跳。

2.4.3. ACL 规则同步

如果没有 使用 User Operator，则可以获得对远程主题的 ACL 访问权限。

如果使用 AclAuthorizer（ 没有 User Operator），则管理对代理访问权限的 ACL 规则也适用于远程主题。能够读取源主题的用户可以读取其远程等效内容。

注意

OAuth 2.0 授权不支持以这种方式访问远程主题。

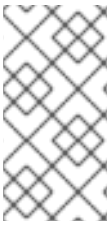
2.4.4. 使用 MirrorMaker 2.0 在 Kafka 集群间同步数据

使用 MirrorMaker 2.0 通过配置同步 Kafka 集群之间的数据。

配置必须指定：

- 每个 Kafka 集群
- 每个集群的连接信息，包括 TLS 身份验证
- 复制流和方向
  - 集群到集群
  - 主题

使用 KafkaMirrorMaker2 资源的属性来配置 Kafka MirrorMaker 2.0 部署。



注意

以前版本的 MirrorMaker 继续受到支持。如果要使用为之前版本配置的资源，则必须将其更新为 MirrorMaker 2.0 所支持的格式。

MirrorMaker 2.0 为复制因素等属性提供默认配置值。最小配置（默认值保持不变）会类似如下：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
```

```
spec:
  version: 2.8.0
  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: {}
```

您可以使用 TLS 或 SASL 身份验证为源和目标集群配置访问控制。此流程显示为源和目标集群使用 TLS 加密和身份验证的配置。

先决条件

- 有关运行 以下的说明，请参阅 OpenShift 指南中的部署和升级 AMQ Streams ：
  - Cluster Operator
  - Kafka 集群
- 源和目标 Kafka 集群必须可用

流程

1. 编辑 KafkaMirrorMaker2 资源的 spec 属性。

   您可以配置的属性显示在此示例配置中：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  version: 2.8.0 1
  replicas: 3 2
  connectCluster: "my-cluster-target" 3
  clusters: 4
  - alias: "my-cluster-source" 5
```

```
    authentication: 6
      certificateAndKey:
        certificate: source.crt
        key: source.key
        secretName: my-user-source
      type: tls
    bootstrapServers: my-cluster-source-kafka-bootstrap:9092 7
    tls: 8
      trustedCertificates:
      - certificate: ca.crt
        secretName: my-cluster-source-cluster-ca-cert
  - alias: "my-cluster-target" 9
    authentication: 10
      certificateAndKey:
        certificate: target.crt
        key: target.key
        secretName: my-user-target
      type: tls
    bootstrapServers: my-cluster-target-kafka-bootstrap:9092 11
    config: 12
      config.storage.replication.factor: 1
      offset.storage.replication.factor: 1
      status.storage.replication.factor: 1
      ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" 13
      ssl.enabled.protocols: "TLSv1.2"
      ssl.protocol: "TLSv1.2"
      ssl.endpoint.identification.algorithm: HTTPS 14
    tls: 15
      trustedCertificates:
      - certificate: ca.crt
        secretName: my-cluster-target-cluster-ca-cert
  mirrors: 16
  - sourceCluster: "my-cluster-source" 17
    targetCluster: "my-cluster-target" 18
    sourceConnector: 19
      tasksMax: 10 20
      config:
        replication.factor: 1 21
        offset-syncs.topic.replication.factor: 1 22
        sync.topic.acls.enabled: "false" 23
        refresh.topics.interval.seconds: 60 24
        replication.policy.separator: "" 25
        replication.policy.class:
"io.strimzi.kafka.connect.mirror.IdentityReplicationPolicy" 26
    heartbeatConnector: 27
      config:
        heartbeats.topic.replication.factor: 1 28
    checkpointConnector: 29
      config:
        checkpoints.topic.replication.factor: 1 30
        refresh.groups.interval.seconds: 600 31
        sync.group.offsets.enabled: true 32
```

```
      sync.group.offsets.interval.seconds: 60 33
      emit.checkpoints.interval.seconds: 60 34
      replication.policy.class:
"io.strimzi.kafka.connect.mirror.IdentityReplicationPolicy"
  topicsPattern: ".*" 35
  groupsPattern: "group1|group2|group3" 36
 resources: 37
  requests:
   cpu: "1"
   memory: 2Gi
  limits:
   cpu: "2"
   memory: 2Gi
 logging: 38
  type: inline
  loggers:
   connect.root.logger.level: "INFO"
 readinessProbe: 39
  initialDelaySeconds: 15
  timeoutSeconds: 5
 livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
 jvmOptions: 40
  "-Xmx": "1g"
  "-Xms": "1g"
 image: my-org/my-image:latest 41
 template: 42
  pod:
   affinity:
    podAntiAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
           - key: application
             operator: In
             values:
              - postgresql
              - mongodb
        topologyKey: "kubernetes.io/hostname"
  connectContainer: 43
   env:
    - name: JAEGER_SERVICE_NAME
      value: my-jaeger-service
    - name: JAEGER_AGENT_HOST
      value: jaeger-agent-name
    - name: JAEGER_AGENT_PORT
      value: "6831"
 tracing:
  type: jaeger 44
 externalConfiguration: 45
  env:
   - name: AWS_ACCESS_KEY_ID
     valueFrom:
```

```
      secretKeyRef:
        name: aws-creds
        key: awsAccessKey
    - name: AWS_SECRET_ACCESS_KEY
      valueFrom:
        secretKeyRef:
          name: aws-creds
          key: awsSecretAccessKey
```

1

Kafka Connect 和 Mirror Maker 2.0 版本，版本 将始终相同。

2

副本节点的数量。

3

Kafka Connect 的 Kafka 集群别名，它必须指定 目标 Kafka 集群。Kafka 集群由 Kafka Connect 用于其内部主题。

4

正在同步 的 Kafka 集群的规格。

5

6

使用此处所示的 TLS 机制 （使用 OAuth bearer 令牌 或基于 SASL 的 SCRAM-SHA-512 或 PLAIN 机制）对源集群进行身份验证。

7

用于 连接到源 Kafka 集群的 bootstrap 服务器。

8

使用源 Kafka 集群的 TLS 证书以 X.509 格式存储的密钥名称进行 TLS 加密。如果证书存储在同一个 secret 中，则可以多次列出证书。

9

目标 Kafka 集群的集群 别名。

10

11

用于连接到目标 Kafka 集群的 bootstrap 服务器。

12

Kafka Connect 配置.标准 Apache Kafka 配置可能会提供，仅限于不是由 AMQ Streams 直接管理的属性。

13

使用特定 密码套件 为 TLS 版本运行外部侦听器的 SSL 属性。

14

通过将 设置为 HTTPS 来 启用主机名验证。空字符串将禁用验证。

15

目标 Kafka 集群的 TLS 加密配置方式与源 Kafka 集群相同。

16

MirrorMaker 2.0 连接器.

17

MirrorMaker 2.0 连接器使用的源集群的集群 别名。

18

MirrorMaker 2.0 连接器使用的目标集群的集群 别名。

19

创建远程主题的 MirrorSourceConnector配置。配置 会覆盖默认配置选项。

20

连接器可以创建的任务数量上限。任务处理数据复制，并并行运行。如果基础架构支持处理开销，增加这个值可以提高吞吐量。Kafka Connect 在集群成员之间分发任务。如果任务数量超过工作程序，则为工作程序分配多项任务。对于接收器连接器，旨在为每个主题分区使用一个任务。对于源连接器，可以并行运行的任务数量也可能取决于外部系统。如果无法实现并行性，连接器会创建少于任务的最大数量。

21

在目标集群中创建的已镜像主题的复制因素。

22

MirrorSourceConnector offset-syncs 内部主题的复制因素，用于映射源和目标集群的偏移。

23

启用 ACL 规则同步 时，将应用 ACL 来同步主题。默认值为 true。

24

可选设置，可更改新主题的检查频率。默认为每 10 分钟检查一次。

25

定义用于重命名远程主题的分隔符。

26

添加可覆盖远程主题自动重命名的策略。该主题不会用源集群的名称来附加名称，而是保留其原始名称。此可选设置对主动/被动备份和数据迁移很有用。要配置主题偏移同步，还必须为 checkpointConnector.config 设置此属性。

27

执行连接检查 的 MirrorHeartbeatConnector配置。配置 会覆盖默认配置选项。

28

在目标集群中创建的心跳主题的复制因素。

29

跟踪偏移的 MirrorCheckpointConnector配置。配置 会覆盖默认配置选项。

30

在目标集群中创建的检查点主题的复制因素。

31

可选设置，以更改新使用者组的检查频率。默认为每 10 分钟检查一次。

32

可选设置来同步消费者组偏移，这对于主动/被动配置中的恢复非常有用。默认情况下不启用同步。

33

如果启用了消费者组偏移的同步，您可以调整同步的频率。

34

调整检查偏移跟踪的频率。如果更改了偏移同步的频率，您可能还需要调整这些检查的频率。

35

从 定义为正则表达式模式的 源集群进行主题复制。这里我们请求所有主题。

36

来自 定义为正则表达式模式的 源集群的使用者组复制。在这里，我们按名称请求三个消费者组。您可以使用逗号分隔的列表。

37

38

指定 Kafka 连接日志记录器和日志级别 直接（内联）或通过ConfigMap 间接（外部）添加。自定义 ConfigMap 必须放在 log4j.properties 或 log4j2.properties 键下。对于 Kafka Connect log4j.rootLogger 日志记录器，您可以将日志级别设置为 INFO、ERROR、WARN、TRACE、DEBUG、FATAL 或 OFF。

39

健康检查以了解 何时重新启动容器（存活度）以及容器何时可以接受流量（就绪度）。

40

用于优化运行 Kafka MirrorMaker 的虚拟机(VM)性能的 JVM 配置选项。

41

ADVANCED OPTION： 容器镜像配置，只在特殊情况下推荐这样做。

42

43 还使用 Jaeger 为分布式追踪设置环境变量。

44 为 Jaeger 启用分布式追踪。

45 作为环境变量挂载到 Kafka MirrorMaker 的 OpenShift Secret 的外部配置。您还可以使用 Kubernetes 配置提供程序从外部来源加载配置值。

2. 创建或更新资源：

```
oc apply -f MIRRORMAKER-CONFIGURATION-FILE
```

2.4.5. 执行 Kafka MirrorMaker 2.0 连接器重启

此流程描述了如何使用 OpenShift 注解手动触发 Kafka MirrorMaker 2.0 连接器重启。

先决条件

- Cluster Operator 正在运行。

流程

1. 查找控制您要重启的 Kafka MirrorMaker 2.0 连接器的 KafkaMirrorMaker2 自定义资源的名称：

```
oc get KafkaMirrorMaker2
```

2. 从 Kafka MirrorMaker2 自定义资源中找到要重启的 Kafka MirrorMaker 2.0 连接器的名称。

```
oc describe KafkaMirrorMaker2 KAFKAMIRRORMAKER-2-NAME
```

3.

要重启连接器，请注解 OpenShift 中的 KafkaMirrorMaker2 资源。在本例中，oc annotate 会重启名为 my-source->my-target.MirrorSourceConnector 的连接器：

```
oc annotate KafkaMirrorMaker2 KAFKAMIRRORMAKER-2-NAME "strimzi.io/restart-connector=my-source->my-target.MirrorSourceConnector"
```

4. 等待下一次协调发生（默认为每隔两分钟）。

Kafka MirrorMaker 2.0 连接器重启，只要协调过程检测到注解。当重启请求被接受时，该注解会从 KafkaMirrorMaker2 自定义资源中删除。

其它资源

- Kafka MirrorMaker 2.0 集群配置。

2.4.6. 执行 Kafka MirrorMaker 2.0 连接器任务重启

此流程描述了如何使用 OpenShift 注解手动触发 Kafka MirrorMaker 2.0 连接器任务的重启。

先决条件

- Cluster Operator 正在运行。

流程

1. 查找控制您要重启的 Kafka MirrorMaker 2.0 连接器的 KafkaMirrorMaker2 自定义资源的名称：

```
oc get KafkaMirrorMaker2
```

2. 查找 Kafka MirrorMaker 2.0 连接器的名称，以及从 KafkaMirrorMaker2 自定义资源重启的任务 ID。任务 ID 是非负整数，从 0 开始。

```
oc describe KafkaMirrorMaker2 KAFKAMIRRORMAKER-2-NAME
```

3. 要重启连接器任务，请注解 OpenShift 中的 KafkaMirrorMaker2 资源。在本例中，oc annotate restarts task 0 of a connector of my-source->my-target.MirrorSourceConnector:

```
oc annotate KafkaMirrorMaker2 KAFKAMIRRORMAKER-2-NAME "strimzi.io/restart-connector-task=my-source->my-target.MirrorSourceConnector:0"
```

4. 等待下一次协调发生（默认为每隔两分钟）。

Kafka MirrorMaker 2.0 连接器任务重启，只要协调过程检测到注解。当重启任务请求被接受时，注解会从 KafkaMirrorMaker2 自定义资源中删除。

其它资源

- Kafka MirrorMaker 2.0 集群配置。

2.5. KAFKA BRIDGE 集群配置

本节论述了如何在 AMQ Streams 集群中配置 Kafka Bridge 部署。

Kafka Bridge 为将基于 HTTP 的客户端与 Kafka 集群集成提供了一个 API。

如果使用 Kafka Bridge，您可以配置 KafkaBridge 资源。

KafkaBridge 资源的完整 schema 信息包括在 第 13.2.110 节 “KafkaBridge 模式参考” 中。

2.5.1. 配置 Kafka 网桥

使用 Kafka Bridge 向 Kafka 集群发出基于 HTTP 的请求。

使用 KafkaBridge 资源的属性来配置 Kafka Bridge 部署。

为了防止不同 Kafka 网桥实例处理客户端消费者请求时出现问题，必须使用基于地址的路由来确保请求路由到正确的 Kafka Bridge 实例。另外，每个独立的 Kafka Bridge 实例都必须有一个副本。Kafka Bridge 实例具有自己的状态，不与其他实例共享。

先决条件

- *OpenShift 集群*
- *一个正在运行的 Cluster Operator*

有关运行 以下的说明，请参阅 OpenShift 指南中的部署和升级 AMQ Streams ：

- *Cluster Operator*
- *Kafka 集群*

流程

1. *编辑 KafkaBridge 资源的 spec 属性。*

   您可以配置的属性显示在此示例配置中：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  replicas: 3 (1)
  bootstrapServers: my-cluster-kafka-bootstrap:9092 (2)
  tls: (3)
    trustedCertificates:
      - secretName: my-cluster-cluster-cert
        certificate: ca.crt
      - secretName: my-cluster-cluster-cert
        certificate: ca2.crt
  authentication: (4)
    type: tls
    certificateAndKey:
      secretName: my-secret
      certificate: public.crt
      key: private.key
  http: (5)
    port: 8080
    cors: (6)
      allowedOrigins: "https://strimzi.io"
      allowedMethods: "GET,POST,PUT,DELETE,OPTIONS,PATCH"
  consumer: (7)
    config:
```

```
    auto.offset.reset: earliest
producer: 8
  config:
    delivery.timeout.ms: 300000
resources: 9
  requests:
    cpu: "1"
    memory: 2Gi
  limits:
    cpu: "2"
    memory: 2Gi
logging: 10
  type: inline
  loggers:
    logger.bridge.level: "INFO"
    # enabling DEBUG just for send operation
    logger.send.name: "http.openapi.operation.send"
    logger.send.level: "DEBUG"
jvmOptions: 11
  "-Xmx": "1g"
  "-Xms": "1g"
readinessProbe: 12
  initialDelaySeconds: 15
  timeoutSeconds: 5
livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
image: my-org/my-image:latest 13
template: 14
  pod:
    affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
                - key: application
                  operator: In
                  values:
                    - postgresql
                    - mongodb
            topologyKey: "kubernetes.io/hostname"
  bridgeContainer: 15
    env:
      - name: JAEGER_SERVICE_NAME
        value: my-jaeger-service
      - name: JAEGER_AGENT_HOST
        value: jaeger-agent-name
      - name: JAEGER_AGENT_PORT
        value: "6831"
```

1

副本节点的数量。

2

用于连接到目标 Kafka 集群的 bootstrap 服务器。

3

使用源 Kafka 集群的 TLS 证书以 X.509 格式存储的密钥名称进行 TLS 加密。如果证书存储在同一个 secret 中，则可以多次列出证书。

4

Kafka Bridge 集群的身份验证，使用 TLS 机制 （此处所示）、使用 OAuth bearer 令牌 或基于 SASL 的 SCRAM-SHA-512 或 PLAIN 机制。默认情况下，Kafka Bridge 在不进行身份验证的情况下连接到 Kafka 代理。

5

对 Kafka 代理的 HTTP 访问.

6

CORS 访问 指定选定的资源和访问方法.请求中的附加 HTTP 标头描述了允许访问 Kafka 集群的原始数据。

7

使用者配置选项.

8

制作者配置选项.

9

请求保留 支持的资源、当前 cpu 和 memory，以及限制，以指定可消耗的最大资源。

10

指定 Kafka Bridge 日志记录器和日志级别 直接（内联）或通过ConfigMap 间接（外部）添加。自定义 ConfigMap 必须放在 log4j.properties 或 log4j2.properties 键下。对于 Kafka Bridge loggers，您可以将日志级别设置为 INFO、ERROR、WARN、TRACE、DEBUG、FATAL 或 OFF。

11

运行 Kafka 网桥的虚拟机(VM)的 JVM 配置选项 优化性能。

12 健康检查以了解何时重新启动容器（存活度）以及容器何时可以接受流量（就绪度）。

13 ADVANCED OPTION： 容器镜像配置，只在特殊情况下推荐这样做。

14 模板自定义.在这里，pod 被调度为反关联性，因此 pod 不会调度到具有相同主机名的节点。

15 还 使用 Jaeger 为分布式追踪设置 环境变量。

2. 创建或更新资源：

```
oc apply -f KAFKA-BRIDGE-CONFIG-FILE
```

2.5.2. Kafka Bridge 集群资源列表

以下资源由 OpenShift 集群中的 Cluster Operator 创建：

bridge-cluster-name-bridge

部署，用于创建 Kafka Bridge worker 节点 pod。

bridge-cluster-name-bridge-service

公开 Kafka Bridge 集群的 REST 接口的服务。

bridge-cluster-name-bridge-config

包含 Kafka Bridge 辅助配置且由 Kafka 代理 Pod 挂载为卷的 ConfigMap。

bridge-cluster-name-bridge

为 Kafka Bridge worker 节点配置的 Pod Disruption Budget。

2.6. 自定义 OPENSHIFT 资源

AMQ Streams 创建几个 OpenShift 资源，如 Deployment、StatefulSet、 Pod 和 Services，它们由 AMQ Streams 操作器管理。只有负责管理特定 OpenShift 资源的操作器才能更改该资源。如果您尝试手动更改由 operator 管理的 OpenShift 资源， Operator 会将您的更改还原。

但是，更改由 Operator 管理的 OpenShift 资源对于执行某些任务很有用，例如：

- 添加控制 Istio 或其他服务如何处理 Pod 的自定义标签或注解
- 管理集群如何创建 Loadbalancer-type Services

您可以使用 AMQ Streams 自定义资源中的 template 属性进行此类更改。template 属性在以下资源中受到支持：API 引用提供了有关可自定义字段的更多详情。

Kafka.spec.kafka

请查看 第 13.2.32 节 “KafkaClusterTemplate 模式参考”

Kafka.spec.zookeeper

请查看 第 13.2.43 节 “ZookeeperClusterTemplate schema reference”

Kafka.spec.entityOperator

请查看 第 13.2.48 节 “EntityOperatorTemplate 模式参考”

Kafka.spec.kafkaExporter

请查看 第 13.2.54 节 “KafkaExporterTemplate 模式参考”

Kafka.spec.cruiseControl

请查看 第 13.2.51 节 “CruiseControlTemplate 模式参考”

KafkaConnect.spec

请查看 第 13.2.68 节 “KafkaConnectTemplate 模式参考”

KafkaConnectS2I.spec

请查看 第 13.2.68 节 “KafkaConnectTemplate 模式参考”

KafkaMirrorMaker.spec

请查看 第 13.2.108 节 “KafkaMirrorMakerTemplate 模式参考”

KafkaMirrorMaker2.spec

请查看 第 13.2.68 节 “KafkaConnectTemplate 模式参考”

KafkaBridge.spec

请查看 第 13.2.118 节 “KafkaBridgeTemplate 模式引用”

KafkaUser.spec

请查看 第 13.2.101 节 “KafkaUserTemplate 模式参考”

在以下示例中，使用 template 属性来修改 Kafka 代理的 StatefulSet 中的标签：

模板自定义示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  labels:
    app: my-cluster
spec:
  kafka:
    # ...
    template:
      statefulset:
        metadata:
          labels:
            mylabel: myvalue
    # ...
```

2.6.1. 自定义镜像拉取策略

AMQ Streams 允许您自定义 Cluster Operator 部署的所有 pod 中容器的镜像拉取策略。镜像拉取策

略使用 Cluster Operator 部署中的环境变量 STRIMZI_IMAGE_PULL_POLICY 进行配置。STRIMZI_IMAGE_PULL_POLICY 环境变量可设置为三个不同的值：

Always

每次 pod 启动或重启时，都会从注册表调取容器镜像。

IfNotPresent

容器镜像只有在之前没有拉取(pull)时才从注册表中提取。

Never

容器镜像永远不会从注册表调取。

镜像拉取策略目前只能对所有 Kafka、Kafka Connect 和 Kafka MirrorMaker 集群进行自定义。更改策略会导致所有 Kafka、Kafka Connect 和 Kafka MirrorMaker 集群滚动更新。

其它资源

- 有关 Cluster Operator 配置的更多信息，请参阅 第 5.1 节 “使用 Cluster Operator”。
- 有关镜像拉取策略的更多信息，请参阅 Disruptions。

2.7. 配置 POD 调度

当两个应用调度到同一个 OpenShift 节点时，这两个应用可能使用相同的资源，如磁盘 I/O 并影响性能。这可能会导致性能下降。以避免与其他关键工作负载共享节点的方式，仅为 Kafka 使用正确的节点或专用一组节点是如何避免这些问题的最佳方法，调度 Kafka pod 是避免这些问题的最佳方法。

2.7.1. 指定关联性、容限和拓扑分布约束

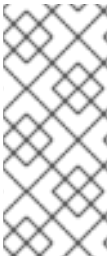
使用关联性、容限和拓扑分布约束将 kafka 资源的 pod 调度到节点上。关联性、容限和拓扑分布约束使用以下资源中 的关联性、容限 和 topologySpreadConstraint 属性进行配置：

- Kafka.spec.kafka.template.pod

- *Kafka.spec.zookeeper.template.pod*
- *Kafka.spec.entityOperator.template.pod*
- *KafkaConnect.spec.template.pod*
- *KafkaConnectS2I.spec.template.pod*
- *KafkaBridge.spec.template.pod*
- *KafkaMirrorMaker.spec.template.pod*
- *KafkaMirrorMaker2.spec.template.pod*

关联性、容限 和 topologySpreadConstraint 属性的格式遵循 OpenShift 规范。关联性配置可以包含不同类型的关联性：

- *pod 关联性和反关联性*
- *节点关联性*



注意

在 OpenShift 1.16 和 1.17 中，对 topologySpreadConstraint 的支持会被默认禁用。要使用 topologySpreadConstraint，您必须在 Kubernetes API 服务器和调度程序中启用 EvenPodsSpread 功能门。

其它资源

- *Kubernetes 节点和 pod 关联性文档*

- *Kubernetes 污点和容限*
- *使用 pod 拓扑分布限制控制 pod 放置*

2.7.1.1. 使用 pod 反关联性以避免关键应用程序共享节点

使用 pod 反关联性来确保关键应用永远不会调度到同一磁盘上。运行 Kafka 集群时，建议使用 pod 反关联性来确保 Kafka 代理不与其他工作负载（如数据库）共享节点。

2.7.1.2. 使用节点关联性将工作负载调度到特定的节点上

OpenShift 集群通常包含许多不同类型的工作程序节点。些已针对 CPU 繁重工作负载优化，有些则用于内存，另一些则可能针对存储（快速本地 SSD）或网络进行了优化。使用不同的节点有助于优化成本和性能。要获得最佳性能，务必要允许调度 AMQ Streams 组件以使用正确的节点。

OpenShift 使用节点关联性将工作负载调度到特定的节点上。通过节点关联性，您可以为要在其上调度 pod 的节点创建调度约束。约束指定为标签选择器。您可以使用内置节点标签（如 beta.kubernetes.io/instance-type） 或自定义标签来指定标签，以选择正确的节点。

2.7.1.3. 对专用节点使用节点关联性和容限

使用污点来创建专用节点，然后通过配置节点关联性和容限将 Kafka pod 调度到专用节点上。

集群管理员可以将选定的 OpenShift 节点标记为污点。具有污点的节点不在常规调度中，一般的 pod 不会调度到它们上运行。只有可以容许节点上设置的污点的服务才可以调度到其中。此类节点上运行的唯一其他服务是系统服务，如日志收集器或软件定义型网络。

在专用节点上运行 Kafka 及其组件可能会有许多优点。同一节点上运行的其他应用程序不会造成干扰或消耗 Kafka 所需的资源。这可提高性能和稳定性。

2.7.2. 配置 pod 反关联性，以将每个 Kafka 代理调度到不同的 worker 节点上

许多 Kafka 代理或 ZooKeeper 节点可以在同一 OpenShift worker 节点上运行。如果 worker 节点失败，它们都会同时不可用。要提高可靠性，您可以使用 podAntiAffinity 配置在不同的 OpenShift worker 节点上调度每个 Kafka 代理或 ZooKeeper 节点。

先决条件

- *OpenShift 集群*
- *一个正在运行的 Cluster Operator*

流程

1. *编辑指定集群部署的资源中的 affinity 属性。要确保 Kafka 代理或 ZooKeeper 节点没有共享 worker 节点，请使用 strimzi.io/name 标签。将 topologyKey 设置为 kubernetes.io/hostname，以指定所选 pod 没有调度到具有相同主机名的节点。这仍然允许同一 worker 节点由单个 Kafka 代理和单个 ZooKeeper 节点共享。例如：*

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    template:
      pod:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: strimzi.io/name
                      operator: In
                      values:
                        - CLUSTER-NAME-kafka
                topologyKey: "kubernetes.io/hostname"
    # ...
  zookeeper:
    # ...
    template:
      pod:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: strimzi.io/name
                      operator: In
                      values:
                        - CLUSTER-NAME-zookeeper
                topologyKey: "kubernetes.io/hostname"
    # ...
```

Where CLUSTER-NAME 是 Kafka 自定义资源的名称。

2.

如果您甚至希望确保 Kafka 代理和 ZooKeeper 节点不共享相同的 worker 节点，请使用 strimzi.io/cluster 标签。例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    template:
      pod:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: strimzi.io/cluster
                      operator: In
                      values:
                        - CLUSTER-NAME
                topologyKey: "kubernetes.io/hostname"
    # ...
  zookeeper:
    # ...
    template:
      pod:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: strimzi.io/cluster
                      operator: In
                      values:
                        - CLUSTER-NAME
                topologyKey: "kubernetes.io/hostname"
    # ...
```

Where CLUSTER-NAME 是 Kafka 自定义资源的名称。

3.

创建或更新资源。

```
oc apply -f KAFKA-CONFIG-FILE
```

2.7.3. 在 Kafka 组件中配置 pod 反关联性

Pod 反关联性配置有助于实现 Kafka 代理的稳定性和性能。通过使用 podAntiAffinity，OpenShift 不会将 Kafka 代理调度到与其他工作负载相同的节点上。通常，您想要避免在与其他网络或者存储密集型

应用程序（如数据库、存储或其他消息传递平台）相同的 worker 节点上运行 Kafka。

先决条件

- OpenShift 集群
- 一个正在运行的 Cluster Operator

流程

1. 编辑指定集群部署的资源中的 affinity 属性。使用标签指定不应调度到同一节点上的 pod。topologyKey 应设置为 kubernetes.io/hostname，以指定不应将所选 pod 调度到具有相同主机名的节点。例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    template:
      pod:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: application
                      operator: In
                      values:
                        - postgresql
                        - mongodb
                topologyKey: "kubernetes.io/hostname"
    # ...
  zookeeper:
    # ...
```

2. 创建或更新资源。

   这可以使用 oc apply 来完成：

```
oc apply -f KAFKA-CONFIG-FILE
```

2.7.4. 在 Kafka 组件中配置节点关联性

先决条件

- *OpenShift 集群*
- *一个正在运行的 Cluster Operator*

流程

1. *标记调度 AMQ Streams 组件的节点。*

   可以使用 oc label 来完成此操作：

   ```
   oc label node NAME-OF-NODE node-type=fast-network
   ```

   或者，某些现有标签可以被重复使用。

2. *编辑指定集群部署的资源中的 affinity 属性。例如：*

   ```
   apiVersion: kafka.strimzi.io/v1beta2
   kind: Kafka
   spec:
     kafka:
       # ...
       template:
         pod:
           affinity:
             nodeAffinity:
               requiredDuringSchedulingIgnoredDuringExecution:
                 nodeSelectorTerms:
                   - matchExpressions:
                     - key: node-type
                       operator: In
                       values:
                       - fast-network
       # ...
     zookeeper:
       # ...
   ```

3. *创建或更新资源。*

   这可以使用 oc apply 来完成：

```
oc apply -f KAFKA-CONFIG-FILE
```

2.7.5. 设置专用节点并在节点上调度 pod

先决条件

- OpenShift 集群
- 一个正在运行的 Cluster Operator

流程

1. 选择应当用作专用的节点。
2. 确保这些节点上未调度工作负载。
3. 在所选节点上设置污点：

   可以使用 oc adm taint 来完成此操作：

   ```
   oc adm taint node NAME-OF-NODE dedicated=Kafka:NoSchedule
   ```
4. 另外，也向所选节点添加标签。

   可以使用 oc label 来完成此操作：

   ```
   oc label node NAME-OF-NODE dedicated=Kafka
   ```
5. 编辑 指定集群部署的资源中的关联性 和 容限 属性。

   例如：

   ```
   apiVersion: kafka.strimzi.io/v1beta2
   kind: Kafka
   spec:
     kafka:
   ```

```
    # ...
    template:
      pod:
        tolerations:
          - key: "dedicated"
            operator: "Equal"
            value: "Kafka"
            effect: "NoSchedule"
        affinity:
          nodeAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              nodeSelectorTerms:
              - matchExpressions:
                - key: dedicated
                  operator: In
                  values:
                  - Kafka
    # ...
  zookeeper:
    # ...
```

6. 创建或更新资源。

   这可以使用 oc apply 来完成：

```
oc apply -f KAFKA-CONFIG-FILE
```

2.8. 日志记录配置

在 Kafka 组件和 AMQ Streams Operator 的自定义资源中配置日志级别。您可以在自定义资源的 spec.logging 属性中直接指定日志级别。或者，您可以使用 configMapKeyRef 属性在自定义资源中引用的 ConfigMap 中定义日志属性。

使用 ConfigMap 的优点是日志记录属性在一个位置维护，并可以被多个资源访问。您还可以为多个资源重复使用 ConfigMap。如果您使用 ConfigMap 指定 AMQ Streams Operator 的日志记录器，您也可以附加日志规格来添加过滤器。

您可以在日志记录规格中指定一个日志 类型 ：

- 直接指定日志级别时 内联
- 引用 ConfigMap 时 外部

内联 日志记录配置示例

```
spec:
  # ...
  logging:
    type: inline
    loggers:
      kafka.root.logger.level: "INFO"
```

外部 日志记录配置示例

```
spec:
  # ...
  logging:
    type: external
    valueFrom:
      configMapKeyRef:
        name: my-config-map
        key: my-config-map-key
```

ConfigMap 名称和 键 的值是必需的。如果未设置 名称或 密钥，则会使用默认日志记录。

其它资源

Kafka 组件日志

- *kafka 日志记录*
- *zookeeper 日志*
- *Kafka Connect 和 Mirror Maker 2.0 日志*

- *MirrorMaker 日志*
- *Kafka Bridge 日志*
- *精简控制日志*

Operator 日志

- *Cluster Operatror 日志*
- *主题 Operator 日志*
- *用户 Operator 日志*

2.8.1. 为日志记录创建 ConfigMap

要使用 ConfigMap 定义日志属性，您需要创建 ConfigMap，然后在资源 spec 中的日志记录定义中引用它。

ConfigMap 必须包含适当的日志记录配置。

- *Kafka 组件、ZooKeeper 和 Kafka Bridge 的 log4j.properties*
- *Topic Operator 和 User Operator 的 log4j2.properties*

配置必须置于这些属性下。

在这一流程中，ConfigMap 为 Kafka 资源定义根日志记录器。

流程

1. 创建 ConfigMap。

   您可以将 ConfigMap 作为 YAML 文件或属性文件创建。

   带有 Kafka 的根日志记录器定义的 ConfigMap 示例：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: logging-configmap
data:
  log4j.properties:
    kafka.root.logger.level="INFO"
```

   如果您使用属性文件，请在命令行中指定该文件：

```
oc create configmap logging-configmap --from-file=log4j.properties
```

   属性文件定义日志配置：

```
# Define the logger
kafka.root.logger.level="INFO"
# ...
```

2. 在资源的 spec 中定义 外部 日志记录，将 logging.valueFrom.configMapKeyRef.name 设置为 ConfigMap 的名称， valueFrom.configMapKeyRef.key 设置为此 ConfigMap 中的键。

```
spec:
  # ...
  logging:
    type: external
    valueFrom:
      configMapKeyRef:
        name: logging-configmap
        key: log4j.properties
```

3. 创建或更新资源。

```
oc apply -f KAFKA-CONFIG-FILE
```

2.8.2. 在 Operator 中添加日志记录过滤器

如果您使用 ConfigMap 为 AMQ Streams Operator 配置(log4j2)日志记录级别，您也可以定义日志记录过滤器来限制日志中返回的内容。

当您有大量日志消息时，日志记录过滤器很有用。假设您将日志记录器的日志级别设置为 DEBUG(rootLogger.level="DEBUG")。日志记录过滤器减少了该级别上日志记录器返回的日志数量，因此您可以专注于特定的资源。设置过滤器后，将仅记录与过滤器匹配的日志消息。

过滤器使用 标记 来指定要包含在日志中的内容。您可以为标记指定一个 kind、namespace 和 name。例如，如果 Kafka 集群失败，您可以通过将 kind 指定为 Kafka 来隔离日志，并使用失败集群的命名空间和名称。

本例显示了名为 my-kafka-cluster 的 Kafka 集群的标记过滤器。

基本日志记录过滤器配置

```
rootLogger.level="INFO"
appender.console.filter.filter1.type=MarkerFilter (1)
appender.console.filter.filter1.onMatch=ACCEPT (2)
appender.console.filter.filter1.onMismatch=DENY (3)
appender.console.filter.filter1.marker=Kafka(my-namespace/my-kafka-cluster) (4)
```

(1) MarkerFilter 类型比较了指定的过滤标记。

(2) 如果标志匹配，onMatch 属性接受日志。

(3) 如果标志不匹配，则 onMismatch 属性将拒绝日志。

(4)

您可以创建一个或多个过滤器。在这里，会针对两个 Kafka 集群过滤日志。

多个日志记录过滤器配置

```
appender.console.filter.filter1.type=MarkerFilter
appender.console.filter.filter1.onMatch=ACCEPT
appender.console.filter.filter1.onMismatch=DENY
appender.console.filter.filter1.marker=Kafka(my-namespace/my-kafka-cluster-1)
appender.console.filter.filter2.type=MarkerFilter
appender.console.filter.filter2.onMatch=ACCEPT
appender.console.filter.filter2.onMismatch=DENY
appender.console.filter.filter2.marker=Kafka(my-namespace/my-kafka-cluster-2)
```

为 Cluster Operator 添加过滤器

要向 Cluster Operator 添加过滤器，更新其日志记录 ConfigMap YAML 文件(install/cluster-operator/050-ConfigMap-strimzi-cluster-operator.yaml)。

流程

1. 更新 050-ConfigMap-strimzi-cluster-operator.yaml 文件，将过滤器属性添加到 ConfigMap 中。

   在本例中，过滤器属性仅为 my-kafka-cluster Kafka 集群返回日志：

   ```
   kind: ConfigMap
   apiVersion: v1
   metadata:
     name: strimzi-cluster-operator
   data:
     log4j2.properties:
       #...
       appender.console.filter.filter1.type=MarkerFilter
       appender.console.filter.filter1.onMatch=ACCEPT
       appender.console.filter.filter1.onMismatch=DENY
       appender.console.filter.filter1.marker=Kafka(my-namespace/my-kafka-cluster)
   ```

或者，直接编辑 ConfigMap：

```
oc edit configmap strimzi-cluster-operator
```

2. 如果您更新了 YAML 文件而不是直接编辑 ConfigMap，请通过部署 ConfigMap 来应用更改：

```
oc create -f install/cluster-operator/050-ConfigMap-strimzi-cluster-operator.yaml
```

为主题 Operator 或 User Operator 添加过滤器

要在 Topic Operator 或 User Operator 中添加过滤器，请创建或编辑日志记录 ConfigMap。

在这一流程中，使用 Topic Operator 的过滤器创建日志 ConfigMap。用户 Operator 也使用相同的方法。

流程

1. 创建 ConfigMap。

   您可以将 ConfigMap 作为 YAML 文件或属性文件创建。

   在本例中，过滤器属性仅为 my-topic 主题返回日志：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: logging-configmap
data:
  log4j2.properties:
    rootLogger.level="INFO"
    appender.console.filter.filter1.type=MarkerFilter
    appender.console.filter.filter1.onMatch=ACCEPT
    appender.console.filter.filter1.onMismatch=DENY
    appender.console.filter.filter1.marker=KafkaTopic(my-namespace/my-topic)
```

   如果您使用属性文件，请在命令行中指定该文件：

```
oc create configmap logging-configmap --from-file=log4j2.properties
```

属性文件定义日志配置：

```
# Define the logger
rootLogger.level="INFO"
# Set the filters
appender.console.filter.filter1.type=MarkerFilter
appender.console.filter.filter1.onMatch=ACCEPT
appender.console.filter.filter1.onMismatch=DENY
appender.console.filter.filter1.marker=KafkaTopic(my-namespace/my-topic)
# ...
```

2. 在资源的 spec 中定义 外部 日志记录，将 logging.valueFrom.configMapKeyRef.name 设置为 ConfigMap 的名称， valueFrom.configMapKeyRef.key 设置为此 ConfigMap 中的键。

   对于 Topic Operator，日志记录在 Kafka 资源的 topicOperator 配置中指定。

```
spec:
  # ...
  entityOperator:
    topicOperator:
      logging:
        type: external
        valueFrom:
          configMapKeyRef:
            name: logging-configmap
            key: log4j2.properties
```

3. 通过部署 Cluster Operator 来应用更改：

```
create -f install/cluster-operator -n my-cluster-operator-namespace
```

其它资源

- 配置 Kafka
- Cluster Operator 日志
- 主题 Operator 日志

- 用户 Operator 日志

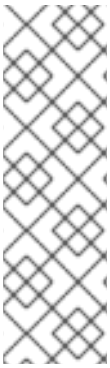
2.9. 从外部源加载配置值

使用 Kubernetes Configuration Provider 插件，从外部来源加载配置数据。从 OpenShift 机密或 ConfigMap 加载数据。

假设您在 Kafka 命名空间外或 Kafka 集群外管理的 Secret。提供程序允许您在配置中引用 Secret 的值，而无需提取文件。您只需告诉提供商要使用的 Secret 并提供访问权限。供应商会在不需要重启 Kafka 组件的情况下加载数据，即使使用新的 Secret 或 ConfigMap 时也是如此。当 Kafka Connect 实例托管多个连接器时，此功能可避免中断。

该供应商独立于 AMQ Streams 运行。您可以使用它来加载所有 Kafka 组件的配置数据，包括生产者和使用者。例如，使用它来提供 Kafka Connect 连接器配置的凭据。

在这一流程中，外部 ConfigMap 为连接器提供配置属性。



注意

OpenShift 配置提供程序无法使用挂载的文件。例如，它无法加载需要信任存储或密钥存储位置的值。相反，您可以将 ConfigMap 或 Secret 挂载到 Kafka Connect pod 中，作为环境变量或卷。您可以使用 KafkaConnect.spec 中的 externalConfiguration 属性添加配置。您无需通过这种方法设置访问权限。但是，当使用新 Secret 或 ConfigMap 作为连接器时，Kafka Connect 需要重启。这会导致所有 Kafka Connect 实例的连接器中断。

先决条件

- OpenShift 集群可用。
- Kafka 集群正在运行。
- Cluster Operator 正在运行。

流程

1. 创建包含配置属性的 ConfigMap 或 Secret。

   在本例中，名为 my-connector-configuration 的 ConfigMap 包含连接器属性：

带有连接器属性的 ConfigMap 示例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-connector-configuration
data:
  option1: value1
  option2: value2
```

2. 在 Kafka Connect 配置中指定 OpenShift 配置提供程序。

   此处显示的规格可支持从 Secret 和 ConfigMap 加载值。

将外部卷设置为 ConfigMap 中的值的示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  # ...
  config:
    # ...
    config.providers: secrets,configmaps (1)
    config.providers.secrets.class: io.strimzi.kafka.KubernetesSecretConfigProvider (2)
    config.providers.configmaps.class:
io.strimzi.kafka.KubernetesConfigMapConfigProvider (3)
  # ...
```

1 配置提供程序的别名用于定义其他配置参数。供应商参数使用 config.providers 中的别名，格式为 config.providers.${alias}.class。

2 KubernetesSecretConfigProvider 提供 Secret 的值。

3 KubernetesConfigMapConfigProvider 提供 ConfigMap 中的值。

3. 创建或更新资源以启用提供程序。

```
oc apply -f KAFKA-CONNECT-CONFIG-FILE
```

4. 创建允许访问外部 ConfigMap 中的值的角色。

从 ConfigMap 访问值的角色示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: connector-configuration-role
rules:
- apiGroups: [""]
  resources: ["configmaps"]
  resourceNames: ["my-connector-configuration"]
  verbs: ["get"]
# ...
```

该规则授予角色权限来访问 my-connector-configuration ConfigMap。

5. 创建角色绑定，以允许访问包含 ConfigMap 的命名空间。

用于访问包含 ConfigMap 的命名空间的角色绑定示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: connector-configuration-role-binding
subjects:
- kind: ServiceAccount
  name: my-connect-connect
  namespace: my-project
roleRef:
  kind: Role
  name: connector-configuration-role
  apiGroup: rbac.authorization.k8s.io
# ...
```

角色绑定授予访问 my-project 命名空间的角色权限。

该服务帐户必须与 Kafka Connect 部署使用的相同。服务帐户名称格式为 CLUSTER_NAME-connect，其中 CLUSTER_NAME 是 KafkaConnect 自定义资源的名称。

6. 在连接器配置中引用 ConfigMap。

引用 ConfigMap 的连接器配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-connector
  labels:
    strimzi.io/cluster: my-connect
spec:
  # ...
  config:
    option: ${configmaps:my-project/my-connector-configuration:option1}
    # ...
# ...
```

ConfigMap 中属性值的占位符在连接器配置中引用。占位符结构为 configmaps:PATH-AND-FILE-NAME:PROPERTY。KubernetesConfigMapConfigProvider 从外部 ConfigMap 中读取并提取 option1 属性值。

其它资源

- *Kafka Connect 连接器的外部配置*

第 3 章 配置外部监听程序

使用外部监听程序向 OpenShift 环境外的客户端公开您的 AMQ Streams Kafka 集群。

在外部监听程序配置中指定要公开 Kafka 的连接 类型。

- NodePort 使用 NodePort 类型 服务
- LoadBalancer 使用 Loadbalancer 类型 服务
- Ingress 使用 Kubernetes Ingress 和 Kubernetes 的 NGINX Ingress Controller
- 路由 使用 OpenShift 路由 和 HAProxy 路由器

有关监听器配置的更多信息，请参阅 GenericKafkaListener 模式参考。

注意

路由 只在 OpenShift 中被支持

其它资源

- 在 Strimzi 中访问 Apache Kafka

3.1. 使用节点端口访问 KAFKA

此流程描述了如何使用节点端口从外部客户端访问 AMQ Streams Kafka 集群。

要连接到代理，您需要 Kafka bootstrap 地址 的主机名和端口号，以及用于身份验证的证书。

先决条件

- *OpenShift 集群*
- *一个正在运行的 Cluster Operator*

流程

1. *配置 Kafka 资源，并将外部监听程序设置为 nodeport 类型。*

   例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    listeners:
      - name: external
        port: 9094
        type: nodeport
        tls: true
        authentication:
          type: tls
        # ...
    # ...
  zookeeper:
    # ...
```

2. *创建或更新资源。*

```
oc apply -f KAFKA-CONFIG-FILE
```

   为每个 Kafka 代理和外部 bootstrap 服务创建 NodePort 类型服务。bootstrap 服务将外部流量路由到 Kafka 代理。用于连接的节点地址被传播到 Kafka 自定义资源的状态。

   也使用与 Kafka 资源相同的名称来创建用于验证 kafka 代理身份的集群 CA 证书。

3. *从 Kafka 资源的状态检索您用来访问 Kafka 集群的 bootstrap 地址。*

```
oc get kafka KAFKA-CLUSTER-NAME -o=jsonpath='{.status.listeners[?
(@.type=="external")].bootstrapServers}{"\n"}'
```

4. 如果启用了 TLS 加密，请提取代理认证机构的公共证书。

```
oc get secret KAFKA-CLUSTER-NAME-cluster-ca-cert -o jsonpath='{.data.ca\.crt}' |
base64 -d > ca.crt
```

使用 Kafka 客户端中提取的证书来配置 TLS 连接。如果启用了任何身份验证，您还需要配置 SASL 或 TLS 身份验证。

3.2. 使用 LOADBALANCERS 访问 KAFKA

这个步骤描述了如何使用 loadbalancers 从外部客户端访问 AMQ Streams Kafka 集群。

要连接到代理，您需要 bootstrap 负载均衡器的地址， 以及用于 TLS 加密的证书。

先决条件

- OpenShift 集群
- 一个正在运行的 Cluster Operator

流程

1. 配置 Kafka 资源，并将外部监听程序设置为 loadbalancer 类型。

例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    listeners:
      - name: external
        port: 9094
        type: loadbalancer
        tls: true
        # ...
```

```
  # ...
zookeeper:
  # ...
```

2. 创建或更新资源。

```
oc apply -f KAFKA-CONFIG-FILE
```

LoadBalancer 类型服务和负载均衡器是为每个 Kafka 代理和一个外部 bootstrap 服务创建的。bootstrap 服务将外部流量路由到所有 Kafka 代理。用于连接的 DNS 名称和 IP 地址会传播到每个服务的状态。

也使用与 Kafka 资源相同的名称来创建用于验证 kafka 代理身份的集群 CA 证书。

3. 从 Kafka 资源的状态检索可用于访问 Kafka 集群的 bootstrap 服务地址。

```
oc get kafka KAFKA-CLUSTER-NAME -o=jsonpath='{.status.listeners[?
(@.type=="external")].bootstrapServers}{"\n"}'
```

4. 如果启用了 TLS 加密，请提取代理认证机构的公共证书。

```
oc get secret KAFKA-CLUSTER-NAME-cluster-ca-cert -o jsonpath='{.data.ca\.crt}' |
base64 -d > ca.crt
```

使用 Kafka 客户端中提取的证书来配置 TLS 连接。如果启用了任何身份验证，您还需要配置 SASL 或 TLS 身份验证。

3.3. 使用 INGRESS 访问 KAFKA

此流程演示了如何使用 Nginx Ingress 从 OpenShift 外部的外部客户端访问 AMQ Streams Kafka 集群。

要连接到代理，需要 Ingress bootstrap 地址的主机名（公告地址），以及用于身份验证的证书。

对于使用 Ingress 访问，端口始终为 443。

TLS passthrough

Kafka 通过 TCP 使用二进制协议，但 Kubernetes 的 NGINX Ingress Controller 旨在使用 HTTP 协议。为了能够通过 Ingress 传递 Kafka 连接，AMQ Streams 使用 Kubernetes NGINX Ingress Controller 的 TLS 直通功能。确保在 NGINX Ingress Controller 中为 Kubernetes 部署启用了 TLS passthrough。

由于它使用 TLS passthrough 功能，所以在使用 Ingress 公开 Kafka 时无法禁用 TLS 加密。

有关启用 TLS 直通的更多信息，请参阅 TLS 直通文档。

先决条件

- OpenShift 集群
- 为启用了 TLS passthrough 的 Kubernetes 部署 NGINX Ingress Controller
- 一个正在运行的 Cluster Operator

流程

1. 配置 Kafka 资源，并将外部监听程序设置为 ingress 类型。

   指定 bootstrap 服务和 Kafka 代理的 Ingress 主机。

   例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    listeners:
      - name: external
        port: 9094
        type: ingress
        tls: true
        authentication:
          type: tls
```

```
    configuration: (1)
      bootstrap:
        host: bootstrap.myingress.com
      brokers:
      - broker: 0
        host: broker-0.myingress.com
      - broker: 1
        host: broker-1.myingress.com
      - broker: 2
        host: broker-2.myingress.com
    # ...
  zookeeper:
    # ...
```

(1) bootstrap 服务和 Kafka 代理的 Ingress 主机。

2. 创建或更新资源。

```
oc apply -f KAFKA-CONFIG-FILE
```

为每个 Kafka 代理创建 ClusterIP 类型服务，以及一个额外的 bootstrap 服务。这些服务供 Ingress 控制器用于将流量路由到 Kafka 代理。还会为每个服务创建一个 Ingress 资源，以使用 Ingress 控制器公开它们。Ingress 主机传播到每个 服务的状态。

也使用与 Kafka 资源相同的名称来创建用于验证 kafka 代理身份的集群 CA 证书。

使用 Kafka 客户端中的 配置 和端口 443(BOOTSTRAP-HOST:443)中指定的 bootstrap 主机 地址，作为 bootstrap 地址 连接到 Kafka 集群。

3. 提取代理认证机构的公共证书。

```
oc get secret KAFKA-CLUSTER-NAME-cluster-ca-cert -o jsonpath='{.data.ca\.crt}' |
base64 -d > ca.crt
```

使用 Kafka 客户端中提取的证书来配置 TLS 连接。如果启用了任何身份验证，您还需要配置 SASL 或 TLS 身份验证。

3.4. 使用 OPENSHIFT 路由访问 KAFKA

此流程描述了如何使用路由从 OpenShift 外部的外部客户端访问 AMQ Streams Kafka 集群。

要连接到代理，您需要路由 bootstrap 地址 的主机名以及用于 TLS 加密的证书。

对于使用路由的访问，端口始终为 443。

先决条件

- OpenShift 集群
- 一个正在运行的 Cluster Operator

流程

1. 配置 Kafka 资源，并将外部监听程序设置为 路由 类型。

   例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  labels:
    app: my-cluster
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    listeners:
      - name: listener1
        port: 9094
        type: route
        tls: true
        # ...
    # ...
  zookeeper:
    # ...
```

警告

OpenShift Route 地址包含 Kafka 集群的名称、侦听器的名称以及在其中创建的命名空间的名称。例如，my -cluster-kafka-listener1-bootstrap-myproject (CLUSTER-NAME-kafka-LISTENER-NAME-bootstrap-NAMESPACE)。请注意，地址的长度不超过 63 个字符的最大限制。

2. *创建或更新资源。*

```
oc apply -f KAFKA-CONFIG-FILE
```

为每个 Kafka 代理和外部 bootstrap 服务 创建 ClusterIP 类型服务。服务将流量从 OpenShift 路由路由到 Kafka 代理。同时也为每个服务创建一个 OpenShift Route 资源，以利用 HAProxy 负载平衡器公开它们。用于连接的 DNS 地址会传播到每个 服务的状态。

也使用与 Kafka 资源相同的名称来创建用于验证 kafka 代理身份的集群 CA 证书。

3. *从 Kafka 资源 的状态检索可用于访问 Kafka 集群的 bootstrap 服务地址。*

```
oc get kafka KAFKA-CLUSTER-NAME -o=jsonpath='{.status.listeners[?(@.type=="external")].bootstrapServers}{"\n"}'
```

4. *提取代理认证机构的公共证书。*

```
oc get secret KAFKA-CLUSTER-NAME-cluster-ca-cert -o jsonpath='{.data.ca\.crt}' | base64 -d > ca.crt
```

使用 Kafka 客户端中提取的证书来配置 TLS 连接。如果启用了任何身份验证，您还需要配置 SASL 或 TLS 身份验证。

第 4 章 管理对 KAFKA 的安全访问

您可以通过管理每个客户端对 Kafka 代理的访问权限来保护 Kafka 集群。

Kafka 代理和客户端之间的安全连接可以包含：

- 为数据交换加密
- 验证以证明身份
- 允许或拒绝用户执行的操作的授权

本章解释了如何在 Kafka 代理和客户端之间设置安全连接，包含如下所述：

- Kafka 集群和客户端的安全选项
- 如何保护 Kafka 代理
- 如何使用授权服务器进行基于 OAuth 2.0 令牌的身份验证和授权

4.1. KAFKA 的安全选项

使用 Kafka 资源配置用于 Kafka 身份验证和授权的机制。

4.1.1. 监听程序验证

对于 OpenShift 集群内的客户端，您可以创建 普通 （无需加密）或 tls 内部 侦听器。

对于 OpenShift 集群外的客户端，您可以创建 外部 侦听器并指定连接机制，可以是 节点端口、负载均衡器、 入口 或 路由 （在 OpenShift 上）。

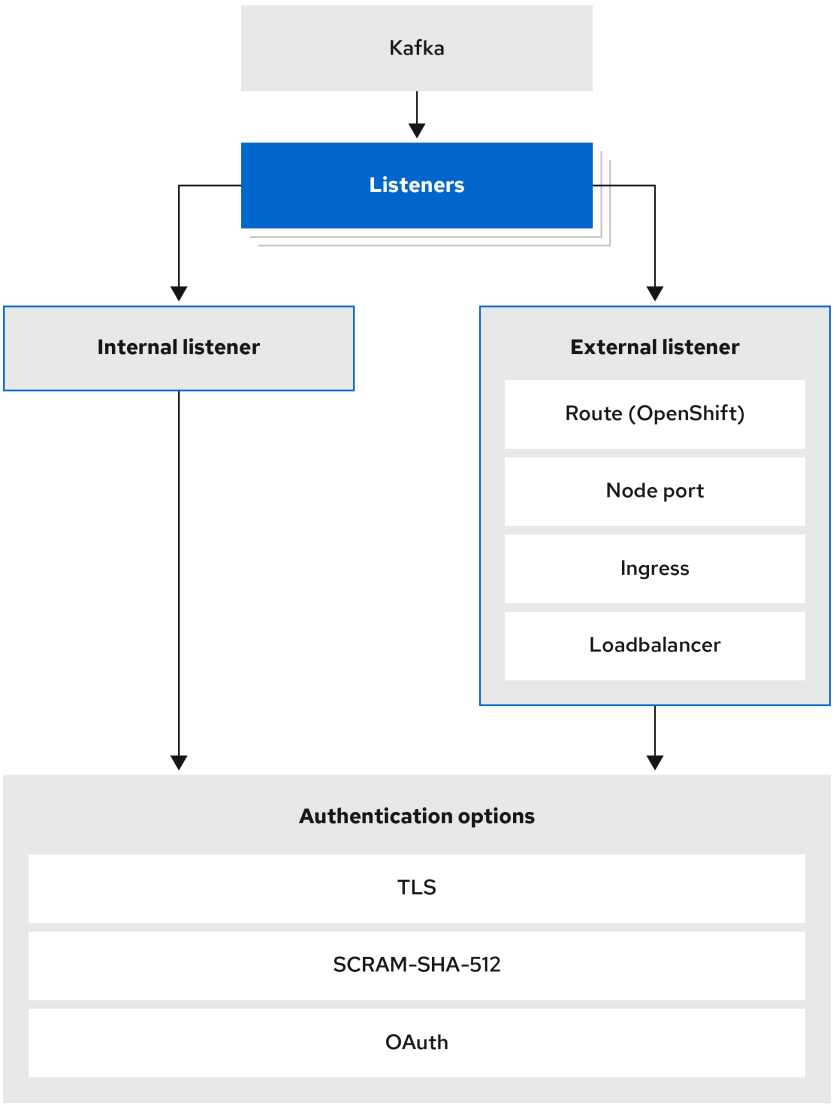
有关连接外部客户端的配置选项的更多信息，请参阅配置外部监听程序。

支持的验证选项：

1. 双向 TLS 身份验证（仅在启用了 TLS 加密的监听器中）
2. SCRAM-SHA-512 验证
3. 基于 OAuth 2.0 令牌的身份验证

您选择的身份验证选项取决于您要如何验证客户端对 Kafka 代理的访问。

图 4.1. Kafka 侦听程序验证选项



侦听器 身份验证 属性用于指定特定于该侦听器的身份验证机制。

如果没有指定 身份验证 属性，侦听器不会验证通过该侦听器进行连接的客户端。侦听器将接受所有连接，无需身份验证。

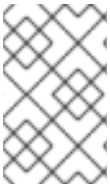
在使用 User Operator 管理 KafkaUsers 时，必须配置身份验证。

以下示例显示：

- 为 SCRAM-SHA-512 身份验证配置 的普通 监听器

- 具有 mutual TLS 身份验证的 A tls 侦听器
- 具有 mutual TLS 身份验证 的外部 监听程序

每个监听程序都使用 Kafka 集群中的唯一名称和端口进行配置。



注意

侦听器不能配置为使用为代理间通信保留的端口（9091 或 9090）和指标(9404)。

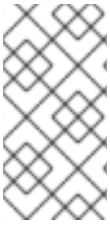
显示监听器身份验证配置的示例

```
# ...
listeners:
  - name: plain
    port: 9092
    type: internal
    tls: true
    authentication:
      type: scram-sha-512
  - name: tls
    port: 9093
    type: internal
    tls: true
    authentication:
      type: tls
  - name: external
    port: 9094
    type: loadbalancer
    tls: true
    authentication:
      type: tls
# ...
```

4.1.1.1. 双向 TLS 身份验证

双向 TLS 身份验证始终用于 Kafka 代理和 ZooKeeper pod 之间的通信。

AMQ Streams 可以配置 Kafka 以使用 TLS（传输层安全），以提供 Kafka 代理和客户端之间的加密通信，无论是否带有或不使用相互身份验证。为互相或双向身份验证，服务器和客户端都提供证书。当您配置相互身份验证时，代理会验证客户端（客户端身份验证），客户端验证代理（服务器身份验证）。



注意

TLS 身份验证更通常是单向，一个方验证另一方的身份。例如，当 Web 浏览器和 Web 服务器之间使用 HTTPS 时，浏览器会获得 Web 服务器身份验证。

4.1.1.2. SCRAM-SHA-512 验证

SCRAM（销售挑战响应身份验证机制）是一种身份验证协议，可以使用密码建立相互身份验证。AMQ Streams 可以配置 Kafka，以使用 SASL（简单身份验证和安全层） SCRAM-SHA-512 提供未加密和加密客户端连接的身份验证。

当 SCRAM-SHA-512 身份验证与 TLS 客户端连接一起使用时，TLS 协议提供加密，但不用于身份验证。

以下 SCRAM 属性可以安全地在未加密连接中使用 SCRAM-SHA-512：

- 密码不会通过通信通道在明文中发送。相反，客户端和服务器都受到另一用户的挑战，难以提供他们知道验证用户密码的证明。
- 服务器和客户端各自为每个身份验证交换生成新的挑战。这意味着交换具有应对重播攻击的弹性。

当使用 scr am-sha-512 配置 KafkaUser.spec.authentication.type 时，User Operator 将生成由大写和小写 ASCII 字母和数字组成的随机 12 个字符密码。

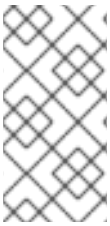
4.1.1.3. 网络策略

AMQ Streams 自动为每个在 Kafka 代理上启用的监听程序创建一个 NetworkPolicy 资源。默认情况下，NetworkPolicy 会 授予监听器访问所有应用程序和命名空间的权限。

如果要将网络级别的监听器访问限制为仅选择的应用程序或命名空间，请使用 networkPolicyPeers 属性。

使用网络策略作为监听器身份验证配置的一部分。每个监听器都可以具有不同的 networkPolicyPeers 配置。

如需更多信息，请参阅 Listener 网络策略 部分和 NetworkPolicyPeer API 参考。



注意

您的 OpenShift 配置必须支持 Ingress NetworkPolicies，以便在 AMQ Streams 中使用网络策略。

4.1.1.4. 其他监听程序配置选项

您可以使用 GenericKafkaListenerConfiguration 模式 的属性在监听器中添加更多配置。

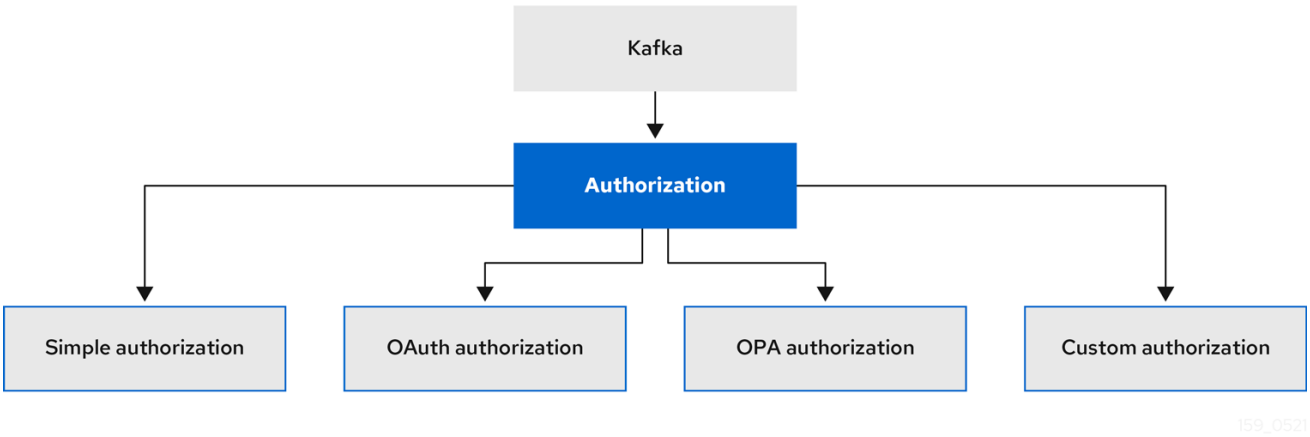
4.1.2. kafka 授权

您可以使用 Kafka. spec.kafka 资源中的 authorization 属性为 Kafka 代理配置授权。如果缺少 authorization 属性，则不启用授权，并且客户端没有限制。启用后，授权将应用到所有启用的监听程序。授权方法在 type 字段中定义。

支持的授权选项：

- 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）
- 打开策略代理(OPA)授权
- 自定义授权

图 4.2. Kafka 集群授权选项



4.1.2.1. 超级用户

无论访问限制如何，超级用户可以访问 Kafka 集群中的所有资源，并受所有授权机制的支持。

要为 Kafka 集群指定超级用户，请在 superUsers 属性中添加用户主体列表。如果用户使用 TLS 客户端身份验证，则其用户名是其证书主题的通用名称，前缀为 CN=。

带有超级用户的示例配置

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    authorization:
      type: simple
      superUsers:
        - CN=client_1
        - user_2
        - CN=client_3
    # ...
```

4.2. KAFKA 客户端的安全选项

使用 KafkaUser 资源，为 Kafka 客户端配置身份验证机制、授权机制和访问权限。在配置安全性方面，客户端以用户身份表示。

您可以验证并授权用户对 Kafka 代理的访问权限。身份验证允许访问，授权限制了对允许的操作的访问。

您还可以创建对 Kafka 代理具有不受约束访问权限的 超级用户。

身份验证和授权机制必须与 用于访问 Kafka 代理的监听程序规格相符。

4.2.1. 为用户处理识别 Kafka 集群

KafkaUser 资源包含一个标签，用于定义它所属的 Kafka 集群的名称（从 Kafka 资源的名称衍生）。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
```

该标签供 User Operator 用于标识 KafkaUser 资源、创建一个新用户，并在后续处理用户时也使用该标签。

如果标签与 Kafka 集群不匹配，User Operator 无法识别 KafkaUser，并且不会创建该用户。

如果 KafkaUser 资源的状态为空，请检查您的标签。

4.2.2. 用户身份验证

使用 KafkaUser.spec 中的 身份验证 属性配置用户身份验证。为用户启用的身份验证机制使用 type 字段指定。

支持的验证机制：

- TLS 客户端身份验证
- SCRAM-SHA-512 验证

如果没有指定身份验证机制，User Operator 不会创建用户或其凭证。

其它资源

- 何时对客户端使用 mutual TLS 身份验证或 SCRAM-SHA 身份验证身份验证

4.2.2.1. TLS 客户端身份验证

要使用 TLS 客户端身份验证，请将 type 字段设置为 tls。

启用了 TLS 客户端身份验证的 KafkaUser 示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: tls
  # ...
```

当 User Operator 创建用户时，它会创建一个名称与 KafkaUser 资源相同的新 Secret。Secret 包含用于 TLS 客户端身份验证的私钥和公钥。公钥包含在用户证书中，由客户端证书颁发机构(CA)签名。

所有键均采用 X.509 格式。

secret 以 PEM 和 PKCS #12 格式提供私钥和证书。

有关保护 Kafka 与 Secret 的通信的更多信息，请参阅 第 11 章 管理 TLS 证书。

带有用户凭证的 Secret 示例

```
apiVersion: v1
kind: Secret
metadata:
  name: my-user
  labels:
    strimzi.io/kind: KafkaUser
    strimzi.io/cluster: my-cluster
type: Opaque
data:
  ca.crt: # Public key of the client CA
  user.crt: # User certificate that contains the public key of the user
  user.key: # Private key of the user
  user.p12: # PKCS #12 archive file for storing certificates and keys
  user.password: # Password for protecting the PKCS #12 archive file
```

4.2.2.2. SCRAM-SHA-512 Authentication

要使用 SCRAM-SHA-512 身份验证机制，请将 type 字段设置为 scr am-sha-512。

启用了 SCRAM-SHA-512 身份验证的 KafkaUser 示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: scram-sha-512
  # ...
```

当 User Operator 创建用户时，它会创建一个名称与 KafkaUser 资源相同的新 secret。该机密包含

password 密钥中生成的密码，该密码使用 base64 编码。若要使用该密码，必须对其进行解码。

带有用户凭证的 Secret 示例

```
apiVersion: v1
kind: Secret
metadata:
  name: my-user
  labels:
    strimzi.io/kind: KafkaUser
    strimzi.io/cluster: my-cluster
type: Opaque
data:
  password: Z2VuZXJhdGVkcGFzc3dvcmQ= (1)
  sasl.jaas.config:
b3JnLmFwYWNoZS5rYWZrYS5jb21tb24uc2VjdXJpdHkuc2NyYW0uU2NyYW1Mb2dpbk1vZHVs
ZSByZXF1aXJlZCB1c2VybmFtZT0ibXktdXNlciIgcGFzc3dvcmQ9ImdlbmVyYXRlZHBhc3N3b3JkI
jsK (2)
```

(1) 生成的密码 base64 编码。

(2) 用于 SASL SCRAM-SHA-512 身份验证的 JAAS 配置字符串，采用 base64 编码。

解码生成的密码：

```
echo "Z2VuZXJhdGVkcGFzc3dvcmQ=" | base64 --decode
```

4.2.3. 用户授权

用户授权使用 KafkaUser.spec 中的 授权 属性进行配置。用户启用的授权类型通过 type 字段指定。

要使用简单授权，您可以在 KafkaUser.spec.authorization 中将 type 属性设置为 simple。简单授权使用默认的 Kafka 授权插件 AclAuthorizer。

或者，您可以使用 OPA 授权，或者已使用基于 OAuth 2.0 令牌的身份验证，您也可以使用 OAuth 2.0 授权。

如果没有指定授权，User Operator 不会为用户置备任何访问权限。这种 KafkaUser 是否可以访问资源取决于所使用的授权程序。例如，对于 AclAuthorizer，这由其 allow.everyone.if.no.acl.found 配置决定。

4.2.3.1. ACL 规则

AclAuthorizer 使用 ACL 规则来管理对 Kafka 代理的访问。

ACL 规则为用户授予访问权限，用户在 acls 属性中指定。

有关 AclRule 对象的更多信息，请参阅 AclRule 架构引用。

4.2.3.2. 超级用户访问 Kafka 代理

如果在 Kafka 代理配置中的超级用户列表中添加用户，则允许用户无限访问集群，无论 KafkaUser 中的 ACL 中定义的任何授权限制。

有关配置超级用户访问权限的更多信息，请参阅 Kafka 授权。

4.2.3.3. 用户配额

您可以配置 KafkaUser 资源的 spec 来强制配额，以便用户不会超过配置的 Kafka 代理访问级别。您可以设置基于大小的网络使用情况和基于时间的 CPU 使用率阈值。您还可以添加分区变异配额，以控制用户请求更改分区的请求的速率。

使用用户配额的 KafkaUser 示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
```

```
spec:
 # ...
 quotas:
  producerByteRate: 1048576 (1)
  consumerByteRate: 2097152 (2)
  requestPercentage: 55 (3)
  controllerMutationRate: 10 (4)
```

(1) 对用户可以推送到 Kafka 代理的数据量的字节每秒配额

(2) 对用户可从 Kafka 代理获取的数据量的字节/秒配额

(3) CPU 使用率限制为客户端组的时间百分比

(4) 每秒允许的并发分区创建和删除操作数（参数）

有关这些属性的更多信息，请参阅 KafkaUserQuotas 架构引用。

4.3. 保护对 KAFKA 代理的访问

要建立对 Kafka 代理的安全访问，您需要配置并应用：

- Kafka 资源，用于：
  - 使用指定验证类型创建监听程序
  - 为整个 Kafka 集群配置授权

- 通过监听程序安全访问 Kafka 代理的 Kafka User 资源

配置 Kafka 资源以设置：

- 监听程序验证
- 限制对 Kafka 侦听器的访问的网络策略
- kafka 授权
- 超级用户对代理进行无限制访问

身份验证是为每个监听器独立配置的。始终为整个 Kafka 集群配置授权。

Cluster Operator 创建监听程序并设置集群和客户端证书颁发机构(CA)证书，以便在 Kafka 集群中启用身份验证。

您可以通过 安装您自己的证书来替换 Cluster Operator 生成的证书。您还可以将 监听程序配置为使用由外部证书颁发机构管理的 Kafka 侦听器证书。证书以 PKCS #12 格式(.p12)和 PEM(.crt)格式提供。

使用 KafkaUser 启用特定客户端用来访问 Kafka 的身份验证和授权机制。

配置 KafkaUser 资源以设置：

- 与启用的监听器验证匹配的身份验证
- 与启用的 Kafka 授权匹配的授权
- 用于控制客户端资源使用的配额

User Operator 根据所选的身份验证类型，创建代表客户端以及用于客户端身份验证的安全凭证的用户。

其它资源

有关以下模式的更多信息：

- k afka，请参阅 Kafka 模式引用。
- KafkaUser，请参阅 KafkaUser 架构引用。

4.3.1. 保护 Kafka 代理

此流程演示了在运行 AMQ Streams 时保护 Kafka 代理的步骤。

为 Kafka 代理实施的安全性必须与为需要访问的客户端实施的安全性兼容。

- Kafka.spec.kafka.listeners[*].authentication matches KafkaUser.spec.authentication
- kafka.spec.kafka.authorization 与 KafkaUser.spec.authorization匹配

步骤演示了使用 TLS 身份验证的简单授权和监听器的配置。有关监听器配置的更多信息，请参阅 GenericKafkaListener 模式参考。

另外，您可以使用 SCRAM-SHA 或 OAuth 2.0 进行 监听程序身份验证，使用 OAuth 2.0 或 OPA 进行 Kafka 授权。

流程

1. 配置 Kafka 资源。
   a. 配置 授权 属性。

b.

配置 listens 属性，以创建具有身份验证的侦听器。

例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    authorization: 1
      type: simple
      superUsers: 2
        - CN=client_1
        - user_2
        - CN=client_3
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls 3
    # ...
  zookeeper:
    # ...
```

1

授权启用使用 AclAuthorizer Kafka 插件的 Kafka 代理 的简单 授权。

2

对 Kafka 拥有无限访问权限的用户主体列表。在使用 TLS 身份验证时，CN 是客户端证书的常用名称。

3

监听器身份验证机制可以为每个侦听器配置，并 指定为 mutual TLS、SCRAM-SHA-512 或基于令牌的 OAuth 2.0。

如果您要配置外部监听程序，其配置取决于所选的连接机制。

2.

创建或更新 Kafka 资源。

```
oc apply -f KAFKA-CONFIG-FILE
```

-

使用 TLS 身份验证配置 Kafka 代理监听程序。

为每个 Kafka 代理 pod 创建一个服务。

创建一个服务作为连接到 Kafka 集群的 bootstrap 地址。

也使用与 Kafka 资源相同的名称来创建用于验证 kafka 代理身份的集群 CA 证书。

4.3.2. 保护用户对 Kafka 的访问

使用 KafkaUser 资源的属性来配置 Kafka 用户。

您可以使用 oc apply 创建或修改用户，使用 oc delete 删除现有用户。

例如：

- oc apply -f USER-CONFIG-FILE
- oc delete KafkaUser USER-NAME

当您配置 KafkaUser 身份验证和授权机制时，请确保它们与等同的 Kafka 配置匹配：

- KafkaUser.spec.authentication matches Kafka.spec.kafka.listeners[*].authentication
- KafkaUser.spec.authorization 与 Kafka.spec.kafka.authorization匹配

此流程演示了如何使用 TLS 身份验证创建用户。您还可以创建具有 SCRAM-SHA 身份验证的用户。

所需的身份验证取决于为 Kafka 代理监听程序配置的身份验证类型。

注意

Kafka 用户和 Kafka 代理之间的身份验证取决于每个用户的身份验证设置。例如，如果 Kafka 配置中没有启用 TLS 的用户，则无法验证它。

先决条件

- 正在运行的 Kafka 集群使用 Kafka 代理监听程序配置使用 TLS 身份验证和加密。
- 一个正在运行的 User Operator（通常使用 Entity Operator 部署）。

KafkaUser 中的身份验证类型应与 Kafka 代理中配置的身份验证匹配。

流程

1. 配置 KafkaUser 资源。

例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication: 1
    type: tls
  authorization:
    type: simple 2
    acls:
      - resource:
          type: topic
          name: my-topic
          patternType: literal
        operation: Read
      - resource:
          type: topic
          name: my-topic
          patternType: literal
```

```
    operation: Describe
  - resource:
      type: group
      name: my-group
      patternType: literal
    operation: Read
```

1. 用户身份验证机制，定义为 mutual tls 或 scr am-sha-512。
2. 简单授权，这需要附带的 ACL 规则列表。

2. 创建或更新 KafkaUser 资源。

```
oc apply -f USER-CONFIG-FILE
```

已创建用户，以及名称与 KafkaUser 资源相同的 Secret。Secret 包含用于 TLS 客户端身份验证的私钥和公钥。

有关使用安全连接到 Kafka 代理的属性配置 Kafka 客户端的详情，请参考在 OpenShift 指南中为 OpenShift 之外的客户端设置访问权限。

4.3.3. 使用网络策略限制对 Kafka 侦听器的访问

您可以使用 networkPolicyPeers 属性将监听器的访问限制为所选应用程序。

先决条件

- 一个支持 Ingress NetworkPolicies 的 OpenShift 集群。
- Cluster Operator 正在运行。

流程

1. 打开 Kafka 资源。

2. 在 networkPolicyPeers 属性中，定义允许访问 Kafka 集群的应用程序 pod 或命名空间。

例如，将 a tls 侦听程序配置为只允许来自标签 app 设置为 kafka-client 的应用程序 pod 的连接：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
        networkPolicyPeers:
          - podSelector:
              matchLabels:
                app: kafka-client
    # ...
  zookeeper:
    # ...
```

3. 创建或更新资源。

使用 oc apply：

```
oc apply -f your-file
```

其它资源

- 有关该架构的更多信息，请参阅 NetworkPolicyPeer API 参考 和 GenericKafkaListener 模式参考。

4.4. 使用基于 OAUTH 2.0 令牌的身份验证

AMQ Streams 支持使用 SASL OAUTHBEARER 机制进行 OAuth 2.0 身份验证。

OAuth 2.0 支持应用之间基于令牌的标准化身份验证和授权，使用中央授权服务器发布对资源的有限访问权限的令牌。

您可以配置 OAuth 2.0 身份验证，然后配置 OAuth 2.0 授权。

OAuth 2.0 身份验证也可与 简单 或基于 OPA 的 Kafka 授权 一起使用。

通过使用基于 OAuth 2.0 令牌的身份验证，应用客户端可以访问应用服务器（称为 资源服务器）上的资源，而无需公开帐户凭据。

应用客户端通过访问令牌来进行身份验证，应用服务器也可以使用该令牌来确定要授予的访问权限级别。授权服务器处理关于访问权限的访问和咨询。

在 AMQ Streams 中：

- Kafka 代理充当 OAuth 2.0 资源服务器
- Kafka 客户端充当 OAuth 2.0 应用程序客户端

Kafka 客户端向 Kafka 代理进行身份验证。代理和客户端根据需要与 OAuth 2.0 授权服务器通信，以获取或验证访问令牌。

对于 AMQ Streams 的部署，OAuth 2.0 集成提供：

- 对 Kafka 代理的服务器端 OAuth 2.0 支持
- 客户端 OAuth 2.0 支持 Kafka MirrorMaker、Kafka Connect 和 Kafka Bridge

其它资源

- OAuth 2.0 站点

4.4.1. OAuth 2.0 身份验证机制

AMQ Streams 支持 OAUTHBEARER 和 PLAIN 机制进行 OAuth 2.0 身份验证。这两种机制都允许 Kafka 客户端通过 Kafka 代理建立经过身份验证的会话。客户端、授权服务器和 Kafka 代理之间的身份验证流程因每种机制而异。

建议您将客户端配置为尽可能使用 OAUTHBEARER。OAUTHBEARER 提供比 PLAIN 更高的安全级别，因为客户端凭证 永远不会 与 Kafka 代理共享。只考虑在不支持 OAUTHBEARER 的 Kafka 客户端中使用 PLAIN。

如果需要，可以在同一个 oauth 侦听器上同时启用 OAUTHBEARER 和 PLAIN。

OAUTHBEARER 概述

Kafka 支持 OAUTHBEARER 身份验证机制，但必须明确配置它。另外，很多 Kafka 客户端工具在协议级别使用为 OAUTHBEARER 提供基本支持的库。

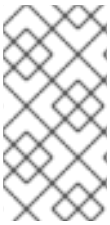
为简化应用程序开发，AMQ Streams 为上游 Kafka 客户端 Java 库（但不为其他库）提供 OAuth 回调处理程序。因此，您不需要为此类客户端自行编写回调处理程序。应用客户端可以使用回调处理程序来提供访问令牌。以其他语言编写的客户端（如 Go）必须使用自定义代码连接到授权服务器并获取访问令牌。

使用 OAUTHBEARER 时，客户端会启动与 Kafka 代理的会话进行凭证交换，其中凭证采用回调处理程序提供的 bearer 令牌的形式。通过使用回调，您可以通过以下三种方法之一配置令牌置备：

- 客户端 ID 和 Secret（使用 OAuth 2.0 客户端凭证机制）
- 长期访问令牌，在配置时手动获得
- 长期存在的刷新令牌，在配置时手动获取

OAUTHBEARER 在 Kafka 代理的 oauth 监听程序配置中自动启用。您可以将 enableOauthBearer 属性设置为 true，但这不是强制要求。

```
# ...
authentication:
  type: oauth
  # ...
  enableOauthBearer: true
```



注意

OAUTHBEARER 身份验证只能由支持协议级别的 OAUTHBEARER 机制的 Kafka 客户端使用。

PLAIN 概述

PLAIN 是供所有 Kafka 客户端工具（包括 kafkacat 等开发人员工具）使用的简单身份验证机制。为了启用 PLAIN 与 OAuth 2.0 身份验证一同使用，AMQ Streams 包含服务器端回调，并通过 PLAIN 调用这个 OAuth 2.0。

随着 PLAIN 的 AMQ Streams 实施，客户端凭据不会存储在 ZooKeeper 中。相反，客户端凭证在兼容授权服务器后面集中处理，这与使用 OAUTHBEARER 身份验证时类似。

当与 OAuth 2.0 over PLAIN 回调搭配使用时，Kafka 客户端使用以下方法之一向 Kafka 代理进行身份验证：

- **客户端 ID 和 secret（使用 OAuth 2.0 客户端凭证机制）**
- **长期访问令牌，在配置时手动获得**

客户端必须启用才能使用 PLAIN 身份验证，并提供 用户名和密码 。如果密码的前缀为 $accessToken:，后跟访问令牌的值，Kafka 代理会将密码解析为访问令牌。否则，Kafka 代理会将 用户名 解释为客户端 ID，密码 将解释为客户端 secret。

如果将 密码 设置为访问令牌，用户名 必须设置为 Kafka 代理从访问令牌获取的相同主体名称。这个过程取决于您如何使用 userNameClaim、fallbackUserNameClaim、fallbackUser namePrefix 或 userInfoEndpointUri 配置用户名提取。它还取决于您的授权服务器；特别是，如何将客户端 ID 映射到帐户名称。

要使用 PLAIN，您必须在 Kafka 代理的 oauth 监听程序配置中启用它。

在以下示例中，除了默认启用的 OAUTHBEARER 外，还启用了 PLAIN。如果只使用 PLAIN，可以通过将 enableOauthBearer 设置为 false 来禁用 OAUTHB EARER。

```
# ...
```

```
  authentication:
    type: oauth
    # ...
    enablePlain: true
    tokenEndpointUri: https://OAUTH-SERVER-
ADDRESS/auth/realms/external/protocol/openid-connect/token
```

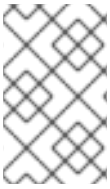
其它资源

- 第 4.4.6.2 节 “配置 Kafka 代理的 OAuth 2.0 支持”

4.4.2. OAuth 2.0 Kafka 代理配置

OAuth 2.0 的 Kafka 代理配置涉及：

- 在授权服务器中创建 OAuth 2.0 客户端
- 在 Kafka 自定义资源中配置 OAuth 2.0 身份验证



注意

对于授权服务器，Kafka 代理和 Kafka 客户端都被视为 OAuth 2.0 客户端。

4.4.2.1. 授权服务器上的 OAuth 2.0 客户端配置

要将 Kafka 代理配置为验证在会话启动过程中收到的令牌，推荐的方法是在授权服务器中创建 OAuth 2.0 客户端 定义（配置为 机密 ）并启用以下客户端凭证：

- kafka 的客户端 ID（例如）
- 客户端 ID 和机密作为身份验证机制

注意

您只需要在使用授权服务器的非公共内省端点时使用客户端 ID 和机密。在使用公共授权服务器端点时，通常不需要凭据，如进行快速本地 JWT 令牌验证一样。

4.4.2.2. Kafka 集群中的 OAuth 2.0 身份验证配置

要在 Kafka 集群中使用 OAuth 2.0 身份验证，例如，您可以使用身份验证方法 oauth 为 Kafka 集群自定义资源指定 TLS 侦听程序配置：

断言 OAuth 2.0 的身份验证方法类型

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: oauth
      #...
```

您可以配置 普通的、tls 和外部 监听程序，但建议您不要使用使用 OAuth 2.0 禁用 TLS 加密 的纯 监听程序 或外部监听程序，因为这会对通过令牌 Sft 进行网络窃取和未经授权的访问造成漏洞。

您可以使用 type: oauth 为安全传输层配置 外部 监听程序，以便与客户端通信。

使用带有外部监听器的 OAuth 2.0

```
# ...
listeners:
  - name: external
    port: 9094
    type: loadbalancer
    tls: true
    authentication:
      type: oauth
    #...
```

tls 属性默认为 false，因此必须启用。

当您将身份验证类型定义为 OAuth 2.0 后，您可以根据验证类型添加配置，或者作为 快速本地 JWT 验证 或使用内省端点进行令牌验证。

为监听程序配置 OAuth 2.0 的步骤以及描述和示例，请参考 为 Kafka 代理配置 OAuth 2.0 支持。

4.4.2.3. 快速本地 JWT 令牌验证配置

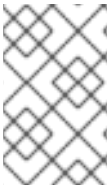
快速本地 JWT 令牌验证在本地检查 JWT 令牌签名。

本地检查可确保令牌：

- 通过包含访问令牌的 Bearer 声明值（拼写错误）符合类型
- 有效（未过期）
- 具有与 validIssuerURI匹配的签发者

在配置监听器时，您可以指定一个 validIssuerURI 属性，以便拒绝授权服务器未发布的任何令牌。

在快速本地 JWT 令牌验证期间，不需要联系授权服务器。您可以通过指定 jwksEndpointUri 属性（OAuth 2.0 授权服务器公开的端点）来激活快速本地 JWT 令牌验证。端点包含用于验证签名 JWT 令牌的公钥，这些令牌由 Kafka 客户端作为凭据发送。



注意

与授权服务器的所有通信都应使用 TLS 加密来执行。

您可以在 AMQ Streams 项目命名空间中将证书信任存储配置为 OpenShift Secret，并使用 a tlsTrustedCertificates 属性指向包含信任存储文件的 OpenShift Secret。

您可能希望配置 userNameClaim 以从 JWT 令牌正确提取用户名。如果要使用 Kafka ACL 授权，则需要在验证过程中根据用户的用户名进行识别。（JWT 令牌中的 子 声明通常是唯一的 ID，而不是用户名。）

用于快速本地 JWT 令牌验证的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    #...
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: oauth
          validIssuerUri: <https://<auth-server-address>/auth/realms/tls>
          jwksEndpointUri: <https://<auth-server-address>/auth/realms/tls/protocol/openid-
connect/certs>
          userNameClaim: preferred_username
          maxSecondsWithoutReauthentication: 3600
          tlsTrustedCertificates:
          - secretName: oauth-server-cert
            certificate: ca.crt
    #...
```

4.4.2.4. OAuth 2.0 内省端点配置

使用 OAuth 2.0 内省端点进行令牌验证，将收到的访问令牌视为不透明。Kafka 代理将访问令牌发送到内省端点，该端点使用验证所需的令牌信息进行响应。更重要的是，如果特定访问令牌有效，它将返回最新信息，以及关于令牌何时到期的信息。

要配置基于 OAuth 2.0 内省的验证，您需要指定一个 内省EndpointUri 属性，而不是为快速本地 JWT 令牌验证指定的 jwksEndpointUri 属性。根据授权服务器，您通常必须指定 clientId 和 client Secret，因为内省端点通常受到保护。

内省端点的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: oauth
          clientId: kafka-broker
          clientSecret:
            secretName: my-cluster-oauth
            key: clientSecret
          validIssuerUri: <https://<auth-server-address>/auth/realms/tls>
          introspectionEndpointUri: <https://<auth-server-
address>/auth/realms/tls/protocol/openid-connect/token/introspect>
          userNameClaim: preferred_username
          maxSecondsWithoutReauthentication: 3600
          tlsTrustedCertificates:
          - secretName: oauth-server-cert
            certificate: ca.crt
```

4.4.3. Kafka 代理的会话重新身份验证

您可以配置 oauth 侦听程序，在 Kafka 客户端和 Kafka 代理间的 OAuth 2.0 会话中使用 Kafka 会话重新身份验证。这种机制强制客户端与代理之间经过身份验证的会话在指定时间段后到期。会话过期时，客户端将通过重复使用现有连接而不是丢弃它来立即启动新会话。

会话重新身份验证默认为禁用。要启用它，您可以在 oauth 监听程序配置中为 maxSecondsWithoutReauthentication 设置一个时间值。同一属性用于为 OAUTHBEARER 和 PLAIN 身份验证配置会话重新身份验证。有关配置示例，请参阅 第 4.4.6.2 节 “配置 Kafka 代理的 OAuth 2.0 支持”。

客户端使用的 Kafka 客户端库必须支持会话重新身份验证。

会话重新身份验证可用于 快速本地 JWT 或 内省端点 令牌验证。

客户端重新身份验证

当代理经过身份验证的会话过期时，客户端必须通过向代理发送新的有效访问令牌来重新验证现有会话，而不丢弃连接。

如果令牌验证成功，则利用现有连接启动新的客户端会话。如果客户端无法重新验证，代理会在进一步尝试发送或接收消息时关闭连接。如果在代理中启用了重新身份验证机制，使用 Kafka 客户端库 2.2 或更高版本的 Java 客户端会自动重新验证。

会话重新身份验证也适用于刷新令牌（若使用）。会话过期时，客户端使用刷新令牌刷新访问令牌。然后，客户端使用新的访问令牌重新验证到现有会话。

OAUTHBEARER 和 PLAIN 的会话过期

当配置会话重新身份验证时，会话过期适用于 OAUTHBEARER 和 PLAIN 身份验证的不同。

对于 OAUTHBEARER 和 PLAIN，请使用客户端 ID 和 secret 方法：

- 代理通过身份验证的会话将在配置的 maxSecondsWithoutReauthentication 时过期。
- 如果访问令牌在配置的时间之前过期，会话将更早过期。

对于 PLAIN 使用长期访问令牌方法：

- 代理通过身份验证的会话将在配置的 maxSecondsWithoutReauthentication 时过期。
- 如果访问令牌在配置的时间之前过期，则重新身份验证会失败。虽然尝试重新身份验证会话，但 PLAIN 没有刷新令牌的机制。

如果没有 配置 maxSecondsWithoutReauthentication，OAUTHBEARER 和 PLAIN 客户端可以无限期地保持连接到代理，而无需重新身份验证。经过身份验证的会话不以访问令牌到期结尾。但是，在配置授权时需要考虑这一点，例如通过使用 keycloak 授权或安装自定义授权器。

其它资源

- 第 4.4.2 节 “OAuth 2.0 Kafka 代理配置”
- 第 4.4.6.2 节 “配置 Kafka 代理的 OAuth 2.0 支持”
- KafkaListenerAuthenticationOAuth 模式参考
- KIP-368

4.4.4. OAuth 2.0 Kafka 客户端配置

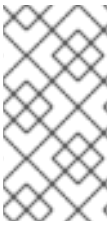
使用以下方法配置了 Kafka 客户端：

- 从授权服务器（客户端 ID 和 Secret）获取有效访问令牌所需的凭证
- 使用授权服务器提供的工具获取有效的长期访问令牌或刷新令牌

发送到 Kafka 代理的唯一信息是访问令牌。用于与授权服务器进行身份验证以获取访问令牌的凭证永远不会发送到代理。

当客户端获取访问令牌时，不需要进一步与授权服务器通信。

最简单的机制是使用客户端 ID 和机密进行身份验证。使用长期访问令牌或长期提供的刷新令牌增加了复杂性，因为对授权服务器工具存在额外的依赖性。



注意

如果您使用长期访问令牌，您可能需要在授权服务器中配置客户端，以增加令牌的最长生命周期。

如果 Kafka 客户端没有直接配置访问令牌，客户端会联系授权服务器在 Kafka 会话启动时交换访问令牌的凭证。Kafka 客户端交换：

- 客户端 ID 和机密
- 客户端 ID、刷新令牌和（可选）Secret

4.4.5. OAuth 2.0 客户端身份验证流

在本节中，我们在 Kafka 会话启动过程中解释和视觉化 Kafka 客户端、Kafka 代理和授权服务器之间的通信流。流程取决于客户端和服务器配置。

当 Kafka 客户端将访问令牌作为凭证发送到 Kafka 代理时，需要验证令牌。

根据所使用的授权服务器以及可用的配置选项，您可能更愿意使用：

- 基于 JWT 签名检查和本地令牌内省的快速本地令牌验证，而不联系授权服务器
- 授权服务器提供的 OAuth 2.0 内省端点

使用快速本地令牌验证需要授权服务器提供 JWKS 端点以及用于验证令牌签名的公共证书。

另一种选择是在授权服务器上使用 OAuth 2.0 内省端点。每次建立新的 Kafka 代理连接时，代理会将从客户端收到的访问令牌传递给授权服务器，并检查响应以确认令牌是否有效。

还可以为以下项配置 Kafka 客户端凭证：

- 使用之前生成的长期访问令牌直接进行本地访问
- 与授权服务器联系以获取要发布的新访问令牌

注意

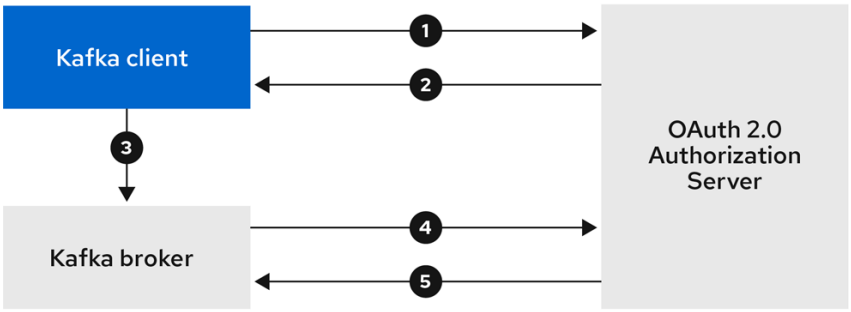
授权服务器可能只允许使用不透明访问令牌，这意味着无法进行本地令牌验证。

4.4.5.1. 客户端身份验证流示例

您可以在 Kafka 会话验证过程中看到用于不同配置的 Kafka 客户端和代理的通信流。

- 使用客户端 ID 和 secret 的客户端，代理将验证委派给授权服务器
- 使用客户端 ID 和 secret 的客户端，代理执行快速本地令牌验证
- 使用长期访问令牌的客户端，并将代理委派验证到授权服务器
- 使用长期访问令牌的客户端，代理执行快速本地验证

使用客户端 ID 和 secret 的客户端，代理将验证委派给授权服务器

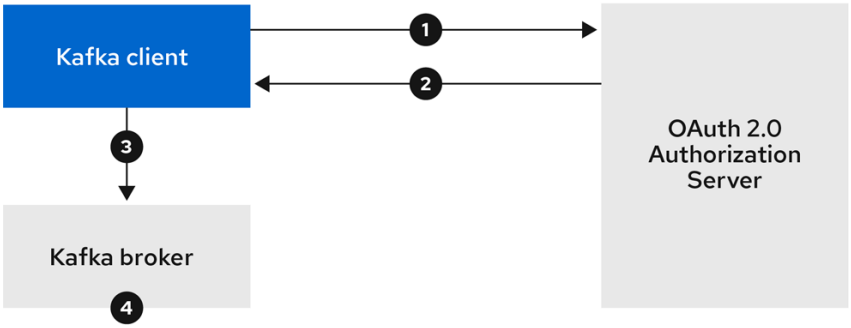


1. Kafka 客户端使用客户端 ID 和机密从授权服务器请求访问令牌，以及可选的刷新令牌。
2. 授权服务器生成新的访问令牌。
3. Kafka 客户端使用 SASL OAUTHBEARER 机制与 Kafka 代理进行身份验证来传递访问令牌。

4. Kafka 代理使用自己的客户端 ID 和 secret，在授权服务器上调用令牌内省端点来验证访问令牌。

5. 如果令牌有效，则创建 Kafka 客户端会话。

使用客户端 ID 和 secret 的客户端，代理执行快速本地令牌验证



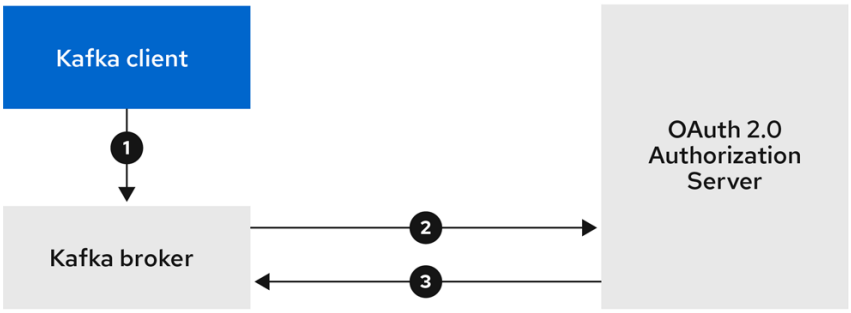
1. Kafka 客户端从令牌端点通过授权服务器进行身份验证，使用客户端 ID 和 secret 以及可选的刷新令牌进行身份验证。

2. 授权服务器生成新的访问令牌。

3. Kafka 客户端使用 SASL OAUTHBEARER 机制与 Kafka 代理进行身份验证来传递访问令牌。

4. Kafka 代理使用 JWT 令牌签名检查和本地令牌内省在本地验证访问令牌。

使用长期访问令牌的客户端，并将代理委派验证到授权服务器



1. *Kafka 客户端使用 SASL OAUTHBEARER 机制与 Kafka 代理进行身份验证，以传递长期访问令牌。*

2. *Kafka 代理使用自己的客户端 ID 和 secret，在授权服务器上调用令牌内省端点来验证访问令牌。*

3. *如果令牌有效，则创建 Kafka 客户端会话。*

使用长期访问令牌的客户端，代理执行快速本地验证

1. *Kafka 客户端使用 SASL OAUTHBEARER 机制与 Kafka 代理进行身份验证，以传递长期访问令牌。*

2. *Kafka 代理使用 JWT 令牌签名检查和本地令牌内省在本地验证访问令牌。*

警告

快速本地 JWT 令牌签名验证仅适用于短期的令牌，因为如果令牌被撤销，则不会与授权服务器检查。令牌到期将写入令牌中，但取消可以在任何时间发生，因此在不联系授权服务器的情况下无法考虑该令牌。任何发布的令牌都将被视为有效，直到该令牌过期为止。

4.4.6. 配置 OAuth 2.0 身份验证

OAuth 2.0 用于 Kafka 客户端和 AMQ Streams 组件之间的交互。

要将 OAuth 2.0 用于 AMQ Streams，您必须：

1. *部署授权服务器并配置部署以与 AMQ 流集成*
2. *使用配置为使用 OAuth 2.0 的 Kafka 代理监听程序部署或更新 Kafka 集群*
3. *更新基于 Java 的 Kafka 客户端以使用 OAuth 2.0*
4. *更新 Kafka 组件客户端以使用 OAuth 2.0*

4.4.6.1. 将红帽单点登录配置为 OAuth 2.0 授权服务器

这个步骤描述了如何将 Red Hat Single Sign-On 部署为授权服务器，并将其配置为与 AMQ Streams 集成。

授权服务器为身份验证和授权以及用户、客户端和权限的管理提供中央点。红帽单点登录有一个域概念，其中 域 代表一组单独的用户、客户端、权限和其他配置。您可以使用默认的 master 域，或者创建一个新域。每个域公开自己的 OAuth 2.0 端点，这意味着应用客户端和应用服务器都需要使用相同的域。

要将 OAuth 2.0 与 AMQ Streams 搭配使用，您可以使用 Red Hat Single Sign-On 部署来创建和管理身份验证域。

注意

如果您已经部署了红帽单点登录，您可以跳过部署步骤并使用当前的部署。

开始前

您需要熟悉使用红帽单点登录。

有关部署和管理说明，请参阅：

- OpenShift 的红帽单点登录
- 服务器管理指南

先决条件

- AMQ Streams 和 Kafka 正在运行

对于 Red Hat Single Sign-On 部署：

- 检查 Red Hat Single Sign-On 支持的配置
- 安装需要具有 cluster-admin 角色的用户，如 system:admin

流程

1. 将红帽单点登录部署到您的 OpenShift 集群。

   在 OpenShift Web 控制台中检查部署的进度。

2. 登录红帽单点登录管理控制台，为 AMQ Streams 创建 OAuth 2.0 策略。

部署 Red Hat Single Sign-On 时会提供登录详情。

3. 创建和启用域。

   您可以使用现有的 master 域。

4. 如果需要，调整域的会话和令牌超时。

5. 创建名为 kafka-broker 的客户端。

6. 在 Settings 选项卡中设置：

   - 访问类型 到 机密
   - 标准流启用 至 OFF，以禁用此客户端的 Web 登录
   - 支持服务于 ON，允许此客户端使用自己的名称进行身份验证

7. 单击 Save，然后再继续。

8. 在 Credentials 选项卡中，记下 AMQ Streams Kafka 集群配置中使用的 secret。

9. 为要连接到 Kafka 代理的任何应用程序客户端重复客户端创建步骤。

   为每个新客户端创建定义。

   在配置中，您将使用名称作为客户端 ID。

接下来要做什么

部署和配置授权服务器后，将 Kafka 代理配置为使用 OAuth 2.0。

4.4.6.2. 配置 Kafka 代理的 OAuth 2.0 支持

这个步骤描述了如何配置 Kafka 代理，以便代理监听程序被启用为使用授权服务器使用 OAuth 2.0 身份验证。

我们建议通过配置 TLS 侦听器在加密接口上使用 OAuth 2.0。不建议纯监听器。

如果授权服务器使用由可信 CA 签名的证书并匹配 OAuth 2.0 服务器主机名，则 TLS 连接可以使用默认设置。否则，您可能需要使用探测器证书或禁用证书主机名验证来配置信任存储。

在配置 Kafka 代理时，在新连接的 Kafka 客户端的 OAuth 2.0 验证过程中，您有两个用于验证访问令牌的机制：

- 配置快速本地 JWT 令牌验证
- 使用内省端点配置令牌验证

开始前

有关为 Kafka 代理监听程序配置 OAuth 2.0 身份验证的更多信息，请参阅：

- KafkaListenerAuthenticationOAuth 模式参考
- 管理对 Kafka 的访问

先决条件

- AMQ Streams 和 Kafka 正在运行
- 部署了 OAuth 2.0 授权服务器

流程

1. *在编辑器中更新Kafka 资源的 Kafka 代理配置(Kafka.spec.kafka)。*

```
oc edit kafka my-cluster
```

2. *配置 Kafka 代理 监听程序 配置。*

   每种侦听器的配置不一定是独立的，因为它们是独立的。

   此处的示例显示了为外部侦听器配置的配置选项。

示例 1：配置快速本地 JWT 令牌验证

```
#...
- name: external
  port: 9094
  type: loadbalancer
  tls: true
  authentication:
    type: oauth (1)
    validIssuerUri: <https://<auth-server-address>/auth/realms/external> (2)
    jwksEndpointUri: <https://<auth-server-
address>/auth/realms/external/protocol/openid-connect/certs> (3)
    userNameClaim: preferred_username (4)
    maxSecondsWithoutReauthentication: 3600 (5)
    tlsTrustedCertificates: (6)
    - secretName: oauth-server-cert
      certificate: ca.crt
    disableTlsHostnameVerification: true (7)
    jwksExpirySeconds: 360 (8)
    jwksRefreshSeconds: 300 (9)
    jwksMinRefreshPauseSeconds: 1 (10)
```

(1) *侦听器类型设置为 oauth。*

(2)

用于身份验证的令牌签发者的 URI。

3

用于本地 JWT 验证的 JWKS 证书端点 URI。

4

在令牌中包含实际用户名的令牌声明（或密钥）。用户名 是用于 标识用户的主体。userNameClaim 值将取决于身份验证流和使用的授权服务器。

5

（可选）激活 Kafka 重新身份验证机制，强制会话过期的时间与访问令牌相同。如果指定的值小于访问令牌过期的时间，客户端必须在实际令牌到期之前重新进行身份验证。默认情况下，会话不会在访问令牌过期时过期，客户端也不会尝试重新身份验证。

6

（可选）用于 TLS 连接到授权服务器的受信任证书。

7

（可选）禁用 TLS 主机名验证。默认为 false。

8

JWKS 证书在过期前被视为有效的时间。默认值为 360 秒。如果您指定了更长的时间，请考虑允许访问撤销的证书的风险。

9

JWKS 证书刷新间隔的时间段。间隔必须至少比到期间隔短 60 秒。默认值为 300 秒。

10

连续尝试刷新 JWKS 公钥之间的最小暂停（以秒为单位）。当遇到未知签名密钥时，将在常规定期计划外调度 JWKS 密钥刷新，并且自上次刷新尝试后至少使用指定的暂停。刷新键遵循 exponential backoff 规则，不成功刷新会一直增加暂停，直到它到达 jwksRefreshSeconds。默认值为 1。

示例 2：使用内省端点配置令牌验证

```
- name: external
  port: 9094
  type: loadbalancer
  tls: true
  authentication:
    type: oauth
    validIssuerUri: <https://<auth-server-address>/auth/realms/external>
    introspectionEndpointUri: <https://<auth-server-
address>/auth/realms/external/protocol/openid-connect/token/introspect> 1
    clientId: kafka-broker 2
    clientSecret: 3
      secretName: my-cluster-oauth
      key: clientSecret
    userNameClaim: preferred_username 4
    maxSecondsWithoutReauthentication: 3600 5
```

1 令牌内省端点的 URI。

2 用于标识客户端的客户端 ID。

3 客户端机密和客户端 ID 用于身份验证。

4 在令牌中包含实际用户名的令牌声明（或密钥）。用户名 是用于 标识用户的主体。userNameClaim 值将取决于所使用的授权服务器。

5 （可选）激活 Kafka 重新身份验证机制，强制会话过期的时间与访问令牌相同。如果指定的值小于访问令牌过期的时间，客户端必须在实际令牌到期之前重新进行身份验证。默认情况下，会话不会在访问令牌过期时过期，客户端也不会尝试重新身份验证。

根据您应用 OAuth 2.0 身份验证的方式，以及授权服务器的类型，您可以使用额外的（可选）配置设置：

```
# ...
```

```
authentication:
  type: oauth
  # ...
  checkIssuer: false (1)
  checkAudience: true (2)
  fallbackUserNameClaim: client_id (3)
  fallbackUserNamePrefix: client-account- (4)
  validTokenType: bearer (5)
  userInfoEndpointUri: https://OAUTH-SERVER-ADDRESS/auth/realms/external/protocol/openid-connect/userinfo (6)
  enableOauthBearer: false (7)
  enablePlain: true (8)
  tokenEndpointUri: https://OAUTH-SERVER-ADDRESS/auth/realms/external/protocol/openid-connect/token (9)
  customClaimCheck: "@.custom == 'custom-value'" (10)
  clientAudience: AUDIENCE (11)
  clientScope: SCOPE (12)
```

(1)

如果您的授权服务器不提供 iss 声明，则无法执行签发者检查。在这种情况下，把 checkIssuer 设置为 false，且不指定 validIssuerUri。默认值为 true。

(2)

如果您的授权服务器提供 ud （严重）申索，并且您希望强制进行使用者检查，请将 checkAudience 设置为 true。使用者检查可识别令牌的预期接收者。因此，Kafka 代理将拒绝在其 aud 声明中没有其 clientId 的令牌。默认为 false。

(3)

授权服务器不能提供单个属性来标识常规用户和客户端。当客户端在其自己的名称中进行身份验证时，服务器可能会提供 客户端 ID。当用户使用用户名和密码进行身份验证时，为了获取刷新令牌或访问令牌，服务器可能会提供除客户端 ID 之外 的用户名 属性。使用此回退选项指定用户名声明(attribute)，以便在主用户 ID 属性不可用时使用。

(4)

在适用 fallbackUserNameClaim 的情况下，可能还需要防止用户名声明的值与回退用户名声明的值冲突。请考虑存在名为 producer 的客户端，但也存在名为 producer 的常规用户。为了区分这两者，您可以使用此属性向客户端的用户 ID 添加前缀。

(5)

（仅在使用 内省EndpointUri时）取决于您使用的授权服务器，内省端点可能会也可能不会返回 令牌类型 属性，或者它可能包含不同的值。您可以指定内省端点必须包含的有效令牌类型值。

(6)

（仅在使用 内省EndpointUri时）可以配置或实施授权服务器，以免在内省端点响应中提供任何可识别的信息。若要获取用户 ID，您可以将 userinfo 端点的 URI 配置为回退。userNameClaim、fallbackUserNameClaim 和 fallbackUserNamePrefix 设置应用于 userinfo 端点的响应。

7

将其设置为 false' 以禁用监听器上的 OAUTHBEARER 机制。必须至少启用 PLAIN 或 OAUTHBEARER 中的一个。默认为 'true。

8

设置为 true，以便在监听器上启用 PLAIN 身份验证，这是所有平台上所有客户端支持的。Kafka 客户端必须启用 PLAIN 机制并设置 用户名和密码 。PLAIN 可用于使用 OAuth 访问令牌或 OAuth 客户端Id 和 secret （客户端凭据）进行身份验证。该行为还由是否指定 tokenEndpointUri 来控制。默认为 false。如果指定了 tokenEndpointUri，并且客户端将 密码 设置为以字符串 $accessToken: 开头，服务器会将密码解析为访问令牌，用户名 为帐户用户名。否则，用户名 将解释为 clientId，密码 解析为客户端 secret，代理用于在客户端名称中获取访问令牌。如果没有指定 tokenEndpointUri，密码 始终解析为访问令牌，并且 用户名 始终被解释为帐户用户名，它必须与从令牌提取的主体 ID 匹配。这称为"no-client-credentials"模式，因为客户端必须始终自行获取访问令牌，并且无法使用 clientId 和 secret。

9

PLAIN 机制的其他配置，允许客户端进行身份验证，方法是传递 clientId 和 secret 作为 用户名和密码 ，如上点所述。如果没有指定，客户端只能通过传递访问令牌作为 密码 参数通过 PLAIN 进行身份验证。

10

可以通过将此规则设置为 JsonPath 过滤器查询，在验证期间对 JWT 访问令牌应用其他自定义规则。如果访问令牌不包含必要的数据，则会被拒绝。使用 内省EndpointUri 时，自定义检查应用到内省端点响应 JSON。

11

（可选） 传递到 令牌端点的使用者参数。在获取用于代理身份验证的访问令牌时，会使用 使用者。它也用于 OAuth 2.0 的客户端名称，通过 PLAIN 客户端身份验证使用 clientId 和 secret。这只会影响获取令牌的能力，以及令牌的内容，具体取决于授权服务器。它不会影响侦听器的令牌验证规则。

12

（可选）传递到令牌端点 的范围 参数。在获取用于代理身份验证的访问令牌时，会使用 范围。它也用于 OAuth 2.0 的客户端名称，通过 PLAIN 客户端身份验证使用 clientId 和 secret。这只会影响获取令牌的能力，以及令牌的内容，具体取决于授权服务器。它不会影响侦听器的令牌验证规则。

3. 保存并退出编辑器，然后等待滚动更新完成。

4. 检查日志中的更新，或者查看 pod 状态转换：

```
oc logs -f ${POD_NAME} -c ${CONTAINER_NAME}
oc get pod -w
```

滚动更新将代理配置为使用 OAuth 2.0 身份验证。

接下来要做什么

- 将 Kafka 客户端配置为使用 OAuth 2.0

4.4.6.3. 将 Kafka Java 客户端配置为使用 OAuth 2.0

这个步骤描述了如何配置 Kafka 制作者和使用者 API，以使用 OAuth 2.0 与 Kafka 代理交互。

将客户端回调插件添加到 pom.xml 文件，并配置系统属性。

先决条件

- AMQ Streams 和 Kafka 正在运行
- 为对 Kafka 代理的 OAuth 访问部署并配置了 OAuth 授权服务器
- 为 OAuth 2.0 配置 Kafka 代理

流程

1. 将带有 OAuth 2.0 支持的客户端库添加到 Kafka 客户端的 pom.xml 文件中：

```
<dependency>
 <groupId>io.strimzi</groupId>
 <artifactId>kafka-oauth-client</artifactId>
```

```
 <version>{oauth-version}</version>
</dependency>
```

2. 为回调配置系统属性：

   例如：

```
System.setProperty(ClientConfig.OAUTH_TOKEN_ENDPOINT_URI, “https://<auth-server-address>/auth/realms/master/protocol/openid-connect/token”); (1)
System.setProperty(ClientConfig.OAUTH_CLIENT_ID, "<client-name>"); (2)
System.setProperty(ClientConfig.OAUTH_CLIENT_SECRET, "<client-secret>"); (3)
```

(1) 授权服务器令牌端点的 URI。

(2) 客户端 ID，这是在授权服务器中创建 客户端 时使用的名称。

(3) 在授权服务器中创建 客户端 时创建的客户端机密。

3. 在 Kafka 客户端配置中的 TLS 加密连接中启用 SASL OAUTHBEARER 机制：

   例如：

```
props.put("sasl.jaas.config",
"org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required;");
props.put("security.protocol", "SASL_SSL"); (1)
props.put("sasl.mechanism", "OAUTHBEARER");
props.put("sasl.login.callback.handler.class",
"io.strimzi.kafka.oauth.client.JaasClientOauthLoginCallbackHandler");
```

(1) 在这里，我们使用 SASL_SSL 通过 TLS 连接使用。对未加密的连接使用 SASL_PLAINTEXT。

4. 验证 Kafka 客户端可以访问 Kafka 代理。

接下来要做什么

- 将 Kafka 组件配置为使用 OAuth 2.0

4.4.6.4. 为 Kafka 组件配置 OAuth 2.0

这个步骤描述了如何将 Kafka 组件配置为使用授权服务器的 OAuth 2.0 身份验证。

您可以为以下对象配置身份验证：

- Kafka Connect
- Kafka MirrorMaker
- Kafka Bridge

在这种情况下，Kafka 组件和授权服务器会在同一集群中运行。

开始前

有关为 Kafka 组件配置 OAuth 2.0 身份验证的更多信息，请参阅：

- KafkaClientAuthenticationOAuth 模式参考

先决条件

- AMQ Streams 和 Kafka 正在运行
- 为对 Kafka 代理的 OAuth 访问部署并配置了 OAuth 授权服务器

- 为 OAuth 2.0 配置 Kafka 代理

流程

1. 创建客户端机密，并将它作为环境变量挂载到组件上。

   例如，此处我们为 Kafka Bridge 创建客户端 Secret ：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Secret
metadata:
 name: my-bridge-oauth
type: Opaque
data:
 clientSecret: MGQ1OTRmMzYtZTllZS00MDY2LWI5OGEtMTM5MzM2NjdlZjQw (1)
```

   (1) clientSecret 键必须采用 base64 格式。

2. 为 Kafka 组件创建或编辑资源，以便为身份验证属性配置 OAuth 2.0 身份验证。

   对于 OAuth 2.0 身份验证，您可以使用：

   - 客户端 ID 和 secret
   - 客户端 ID 和刷新令牌
   - 访问令牌
   - TLS

   KafkaClientAuthenticationOAuth 模式参考提供了每个模式的示例。

   例如，此处的 OAuth 2.0 使用客户端 ID 和 secret 分配给 Kafka Bridge 客户端，以及

TLS：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  authentication:
    type: oauth (1)
    tokenEndpointUri: https://<auth-server-address>/auth/realms/master/protocol/openid-connect/token (2)
    clientId: kafka-bridge
    clientSecret:
      secretName: my-bridge-oauth
      key: clientSecret
    tlsTrustedCertificates: (3)
    - secretName: oauth-server-cert
      certificate: tls.crt
```

(1) 验证类型设置为 oauth。

(2) 用于身份验证的令牌端点的 URI。

(3) TLS 到授权服务器的可信证书。

根据您应用 OAuth 2.0 身份验证的方式以及授权服务器的类型，您可以使用额外的配置选项：

```
# ...
spec:
  # ...
  authentication:
    # ...
    disableTlsHostnameVerification: true (1)
    checkAccessTokenType: false (2)
    accessTokenIsJwt: false (3)
    scope: any (4)
    audience: kafka (5)
```

(1)

（可选）禁用 TLS 主机名验证。默认为 false。

2

如果授权服务器没有在 JWT 令牌中返回 拼写错误 （类型）声明，您可以应用 checkAccessTokenType: false 来跳过令牌类型检查。默认值为 true。

3

如果您使用不透明令牌，您可以应用 accessTokenIsJwt: false，以便访问令牌不被视为 JWT 令牌。

4

（可选）从令牌端点请求令牌 的范围。授权服务器可能需要客户端指定范围。本例中为 任意.

5

（可选）从令牌端点请求令牌的 受众。授权服务器可能需要客户端指定受众。本例中为 kafka。

3. 将更改应用到 Kafka 资源的部署。

```
oc apply -f your-file
```

4. 检查日志中的更新，或者查看 pod 状态转换：

```
oc logs -f ${POD_NAME} -c ${CONTAINER_NAME}
oc get pod -w
```

滚动更新配置组件，以使用 OAuth 2.0 身份验证与 Kafka 代理交互。

4.5. 使用基于 OAUTH 2.0 令牌的授权

如果您在 Red Hat Single Sign-On 中使用 OAuth 2.0 进行基于令牌的身份验证，您还可以使用 Red Hat Single Sign-On 配置授权规则来限制客户端对 Kafka 代理的访问。身份验证建立用户的身份。授权决定该用户的访问级别。

AMQ Streams 支持通过 Red Hat Single Sign-On Authorization Services 使用基于 OAuth 2.0 令牌的授权，它允许您集中管理安全策略和权限。

Red Hat Single Sign-On 中定义的安全策略和权限用于授予对 Kafka 代理上资源的访问权限。用户和客户端与允许对 Kafka 代理执行特定操作的策略进行匹配。

默认情况下，Kafka 允许所有用户对代理进行完全访问，并提供 AclAuthorizer 插件以根据访问控制列表(ACL)配置授权。

zookeeper 存储允许或拒绝基于 用户名 访问资源的 ACL 规则。但是，使用红帽单点登录的 OAuth 2.0 基于令牌的授权为您要对 Kafka 代理实施访问控制提供了极大的灵活性。另外，您可以将 Kafka 代理配置为使用 OAuth 2.0 授权和 ACL。

其它资源

- 使用基于 OAuth 2.0 令牌的身份验证
- Kafka 授权
- Red Hat Single Sign-On 文档

4.5.1. OAuth 2.0 授权机制

AMQ Streams 中的 OAuth 2.0 授权使用 Red Hat Single Sign-On 服务器授权服务 REST 端点通过对特定用户应用定义的安全策略，并为该用户提供不同资源获得的权限列表，从而通过红帽单点登录来扩展基于令牌的身份验证。策略使用角色和组来匹配用户的权限。OAuth 2.0 授权根据红帽单点登录授权服务中用户收到的授权列表，在本地强制实施权限。

4.5.1.1. Kafka 代理自定义授权器

一个 Red Hat Single Sign-On 授权程序 (KeycloakRBACAuthorizer)由 AMQ Streams 提供。为了能够将 Red Hat Single Sign-On REST 端点用于 Red Hat Single Sign-On 提供的授权服务，您可以在 Kafka 代理上配置自定义授权程序。

授权者根据需要从授权服务器获取授予权限的列表，并在 Kafka Broker 上强制本地实施授权，从而对每个客户端请求做出快速的授权决策。

4.5.2. 配置 OAuth 2.0 授权支持

这个步骤描述了如何使用 Red Hat Single Sign-On Authorization Services 将 Kafka 代理配置为使用 OAuth 2.0 授权服务。

开始前

考虑您需要或希望限制某些用户的访问权限。您可以使用红帽单点登录 组、角色、客户端 和用户 组合在红帽单点登录中配置访问权限。

通常，组用于根据组织部门或地理位置匹配用户。和 角色用于根据用户的功能匹配用户。

通过红帽单点登录，您可以将用户和组存储在 LDAP 中，而客户端和角色则无法以这种方式存储。存储和用户数据访问可能是您选择配置授权策略的一个因素。

注意

无论 Kafka 代理中实施的授权如何，超级用户 始终都对 Kafka 代理具有未经约束的访问权限。

先决条件

- AMQ Streams 必须配置为使用 OAuth 2.0 和 Red Hat Single Sign-On 基于 令牌的身份验证。设置授权时，您可以使用相同的 Red Hat Single Sign-On 服务器端点。
- OAuth 2.0 身份验证必须使用 maxSecondsWithoutReauthentication 选项进行配置，才能启用重新身份验证。

流程

1. 访问 Red Hat Single Sign-On Admin 控制台，或使用 Red Hat Single Sign-On Admin CLI 为您在设置 OAuth 2.0 身份验证时创建的 Kafka 代理客户端启用授权服务。
2. 使用授权服务为客户端定义资源、授权范围、策略和权限。
3. 通过为用户和组分配角色和组，将权限绑定至用户和组。

4.

通过在 编辑器 中更新 Kafka 代理配置(Kafka.spec.kafka)，将 Kafka代理配置为使用 Red Hat Single Sign-On 授权。

```
oc edit kafka my-cluster
```

5.

将 Kafka 代理 kafka 配置配置为使用 keycloak 授权，并可以访问授权服务器和授权服务。

例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    authorization:
      type: keycloak (1)
      tokenEndpointUri: <https://<auth-server-address>/auth/realms/external/protocol/openid-connect/token> (2)
      clientId: kafka (3)
      delegateToKafkaAcls: false (4)
      disableTlsHostnameVerification: false (5)
      superUsers: (6)
      - CN=fred
      - sam
      - CN=edward
      tlsTrustedCertificates: (7)
      - secretName: oauth-server-cert
        certificate: ca.crt
      grantsRefreshPeriodSeconds: 60 (8)
      grantsRefreshPoolSize: 5 (9)
    #...
```

(1)

类型 keycloak 启用红帽单点登录授权。

(2)

红帽单点登录令牌端点的 URI。对于生产环境，始终使用 HTTPs。当您配置基于令牌的 oauth 身份验证时，您可以将 jwksEndpointUri 指定为本地 JWT 验证的 URI。tokenEndpointUri URI 的主机名必须相同。

(3)

4 （可选）如果 Red Hat Single Sign-On Authorization Services 策略拒绝访问，则会向 Kafka AclAuthorizer 委派授权。默认为 false。

5 （可选）禁用 TLS 主机名验证。默认为 false。

6 （可选）指定的超级用户.

7 （可选）用于 TLS 连接到授权服务器的受信任证书。

8 （可选）两个连续的时间允许刷新运行。这是活跃会话的最大时间，用于检测 Red Hat Single Sign-On 上用户的任何权限更改。默认值为 60。

9 （可选）用于（并行）活动会话授权的线程数量。默认值为 5。

6. 保存并退出编辑器，然后等待滚动更新完成。

7. 检查日志中的更新，或者查看 pod 状态转换：

```
oc logs -f ${POD_NAME} -c kafka
oc get pod -w
```

滚动更新将代理配置为使用 OAuth 2.0 授权。

8. 以具有特定角色的客户端或用户访问 Kafka 代理，确保他们具有必要的访问权限，或者没有他们应该拥有的访问权限，来验证配置的权限。

4.5.3. 在 Red Hat Single Sign-On Authorization Services 中管理策略和权限

本节论述了 Red Hat Single Sign-On Authorization Services 和 Kafka 使用的授权模型，并定义了每个模型中的重要概念。

要授予访问 Kafka 的权限，您可以通过在 Red Hat Single Sign-On Sign-On 登陆到 Red Hat Single Sign-On 中创建 OAuth 客户端规格 将 Red Hat Single Sign-On Authorization Services 对象映射到 Kafka 资源。使用 Red Hat Single Sign-On Authorization Services 规则为用户帐户或服务帐户授予 Kafka 权限。

显示了 常见 Kafka 操作所需的不同用户权限，如创建和列出主题。

4.5.3.1. Kafka 和红帽单点登录授权模型概述

Kafka 和红帽单点登录授权服务使用不同的授权模型。

Kafka 授权模型

Kafka 的授权模型使用 资源类型。当 Kafka 客户端对代理执行操作时，代理使用配置的 KeycloakRBACAuthorizer 检查客户端的权限，具体取决于操作和资源类型。

Kafka 使用五个资源类型来控制访问： Topic、group、 Cluster、transactionalId 和 DelegationToken。每一资源类型具有一组可用权限。

主题

- create
- 写
- 读
- 删除
- describe

- *DescribeConfigs*
- *更改*
- *AlterConfigs*

组

- *读*
- *describe*
- *删除*

Cluster

- *create*
- *describe*
- *更改*
- *DescribeConfigs*
- *AlterConfigs*
- *IdempotentWrite*

- *ClusterAction*

TransactionalId

- *describe*
- *写*

DelegationToken

- *describe*

Red Hat Single Sign-On Authorization Services 模型

红帽单点登录授权服务模型具有四个概念，用于定义和授予权限：资源、授权范围、 策略 和权限。

资源

资源是一组资源定义，用于将资源与允许的操作匹配。例如，资源可以是单独的主题，或者名称以同一前缀开头的所有主题。资源定义与一组可用的授权范围相关联，后者代表一组资源上可用的所有操作。通常，只有一部分操作实际被允许。

授权范围

授权范围是针对特定资源定义的一组所有可用操作。当您定义新资源时，您可以添加所有范围集中的范围。

策略（policy）

策略是使用条件与帐户列表匹配的授权规则。策略可以匹配：

- 基于 客户端 ID 或角色的服务帐户
- 基于用户名、组或角色的用户帐户。

权限

权限将特定资源定义的一组授权范围授予一组用户。

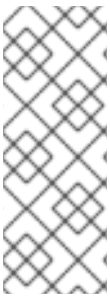
其它资源

- Kafka 授权模型

4.5.3.2. 将 Red Hat Single Sign-On Authorization Services 映射到 Kafka 授权模型

Kafka 授权模型用作定义控制对 Kafka 访问的红帽单点登录角色和资源的基础。

要为用户帐户或服务帐户授予 Kafka 权限，请首先为 Kafka 代理在 Red Hat Single Sign-On 中创建 OAuth 客户端规格。然后，在客户端上指定红帽单点登录授权服务规则。通常，代表代理的 OAuth 客户端的客户端 ID 是 kafka （AMQ Streams 提供 的示例文件 使用 kafka 作为 OAuth 客户端 ID）。



注意

如果您有多个 Kafka 集群，您可以在所有这些集群中使用单个 OAuth 客户端 (kafka)。这为您提供了一个统一空间，用于定义和管理授权规则。但是，您也可以使用不同的 OAuth 客户端 ID（如 my-cluster-kafka 或 cluster-dev-kafka），并为每个客户端配置中为每个集群定义授权规则。

kafka 客户端定义必须在红帽单点管理控制台中启用 Authorization Enabled 选项。

所有权限都存在于 kafka 客户端范围内。如果您为不同的 Kafka 集群配置了不同的 OAuth 客户端 ID，它们各自需要一组独立的权限，即使它们属于同一红帽单点登录域。

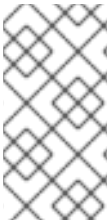
当 Kafka 客户端使用 OAUTHBEARER 身份验证时，Red Hat Single Sign-On 授权器 (KeycloakRBACAuthorizer)使用当前会话的访问令牌来检索 Red Hat Single Sign-On 服务器中的授权列表。若要检索授权，授权者将评估红帽单点登录授权服务策略和权限。

Kafka 权限的授权范围

初始红帽单点登录配置通常涉及上传授权范围，以创建在每个 Kafka 资源类型上执行的所有可能操作的列表。此步骤仅执行一次，然后再定义任何权限。您可以手动添加授权范围，而不是上传它们。

授权范围必须包含所有可能的 Kafka 权限，而不考虑资源类型：

- *create*
- *写*
- *读*
- *删除*
- *describe*
- *更改*
- *DescribeConfig*
- *AlterConfig*
- *ClusterAction*
- *IdempotentWrite*



注意

如果您确定不需要权限（如 IdempotentWrite），您可以从授权范围列表中省略它。但是，该权限将无法在 Kafka 资源上作为目标。

权限检查的资源模式

在执行权限检查时，资源模式用于与目标资源进行模式匹配。常规模式格式为 RESOURCE-TYPE:PATTERN-NAME。

资源类型镜像 Kafka 授权模型。模式允许两个匹配选项：

- 精确匹配（当模式不以 * 结尾 时）
- 前缀匹配（模式以 * 结尾 时）

资源模式示例

```
Topic:my-topic
Topic:orders-*
Group:orders-*
Cluster:*
```

此外，常规模式格式可以加上 kafka-cluster:CLUSTER-NAME 前缀，后面添加一个逗号，其中 CLUSTER-NAME 是指 Kafka 自定义资源中的 metadata.name。

具有集群前缀的资源的模式示例

```
kafka-cluster:my-cluster,Topic:*
kafka-cluster:*,Group:b_*
```

当缺少 kafka-cluster 前缀时，会假定为 kafka-cluster:*。

在定义资源时，您可以将它与与资源相关的可能授权范围列表关联。设置任何对目标资源类型有用的操作。

虽然您可以向任何资源添加任何授权范围，但只有资源类型支持的范围才会被视为访问控制。

应用访问权限的策略

策略用于将权限定向到一个或多个用户帐户或服务帐户。目标可指：

- *特定用户或服务帐户*
- *域角色或客户端角色*
- *用户组*
- *与客户端 IP 地址匹配的 JavaScript 规则*

策略被赋予唯一名称，并可以重复利用以将多个权限作为多个资源的目标。

授予访问权限

使用细粒度权限将策略、资源和授权范围提取到一起，从而授予用户访问权限。

每个权限的名称应明确定义它授予哪些用户的权限。例如，开发团队 B 可以从以 x 开头的主题中读取内容。

其它资源

- *有关如何通过 Red Hat Single Sign-On Authorization Services 配置权限的详情请参考 第 4.5.4 节 “试用红帽单点登录授权服务”。*

4.5.3.3. Kafka 操作所需的权限示例

以下示例演示了在 Kafka 上执行常见操作所需的用户权限。

创建主题

要创建主题，特定主题或 Cluster:kafka-cluster 需要 Create 权限。

```
bin/kafka-topics.sh --create --topic my-topic \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-
config=/tmp/config.properties
```

列出主题

如果用户对指定主题具有 描述 权限，则会列出该主题。

```
bin/kafka-topics.sh --list \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-
config=/tmp/config.properties
```

显示主题详细信息

要显示主题的详细信息，主题需要 Description scribe 和 DescribeConfigs 权限。

```
bin/kafka-topics.sh --describe --topic my-topic \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-
config=/tmp/config.properties
```

生成到一个主题的消息

要生成到某个主题的消息，主题需要Describe 和 Write 权限。

如果尚未创建该主题，并且已启用主题自动创建，则需要有创建主题的权限。

```
bin/kafka-console-producer.sh  --topic my-topic \
  --broker-list my-cluster-kafka-bootstrap:9092 --producer.config=/tmp/config.properties
```

使用来自主题的消息

若要使用某个主题的消息，主题需要 描述 和 读取 权限。从主题中消耗的消耗通常取决于将消费者偏移存储在消费者组中，这需要消费者组具有额外的 描述 和 读取 权限。

匹配需要两个 资源。例如：

```
Topic:my-topic
Group:my-group-*
```

```
bin/kafka-console-consumer.sh --topic my-topic --group my-group-1 --from-beginning \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --consumer.config
/tmp/config.properties
```

使用幂等生成器生成到主题的消息

除了生成主题的权限外，集群资源需要有额外的 IdempotentWrite 权限 。

匹配需要两个 资源。例如：

```
Topic:my-topic
Cluster:kafka-cluster
```

```
bin/kafka-console-producer.sh  --topic my-topic \
  --broker-list my-cluster-kafka-bootstrap:9092 --producer.config=/tmp/config.properties --
producer-property enable.idempotence=true --request-required-acks -1
```

列出消费者组

在列出消费者组时，仅返回用户具有 Describe 权限的 组。或者，如果用户对 Cluster:kafka-cluster 具有 Describe 权限，则返回所有消费者组。

```
bin/kafka-consumer-groups.sh --list \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-
config=/tmp/config.properties
```

显示消费者组详情

要显示消费者组的详细信息，需要对组具有 描述 权限，以及与该组关联的主题。

```
bin/kafka-consumer-groups.sh --describe --group my-group-1 \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-
config=/tmp/config.properties
```

更改主题配置

要更改主题的配置，主题必须具有 描述 和 Alter 权限。

```
bin/kafka-topics.sh --alter --topic my-topic --partitions 2 \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-
config=/tmp/config.properties
```

显示 Kafka 代理配置

要使用 kafka-configs.sh 获取代理的配置，需要在 Cluster:kafka-cluster 上具有 Describe Configs 权限。

```
bin/kafka-configs.sh --entity-type brokers --entity-name 0 --describe --all \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-
config=/tmp/config.properties
```

更改 Kafka 代理配置

要更改 Kafka 代理的配置，Cluster:kafka-cluster 需要 Describe Configs 和 AlterConfigs 权限。

```
bin/kafka-configs --entity-type brokers --entity-name 0 --alter --add-config
log.cleaner.threads=2 \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-
config=/tmp/config.properties
```

删除主题

要删除某个主题，主题上需要描述和删除 权限 。

```
bin/kafka-topics.sh --delete --topic my-topic \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-
config=/tmp/config.properties
```

选择潜在分区

要为主题分区运行领导选择，需要在 Cluster:kafka-cluster 上具有 Alter 权限。

```
bin/kafka-leader-election.sh --topic my-topic --partition 0 --election-type PREFERRED  /
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --admin.config /tmp/config.properties
```

重新分配分区

要生成分区重新分配文件，需要对涉及的主题 描述 权限。

```
bin/kafka-reassign-partitions.sh --topics-to-move-json-file /tmp/topics-to-move.json --broker-
list "0,1" --generate \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-config
/tmp/config.properties > /tmp/partition-reassignment.json
```

要执行分区重新分配，需要在 Cluster:kafka-cluster 上具有 Describe 和 Alter 权限。此外，描述 涉及的主题的权限是必需的。

```
bin/kafka-reassign-partitions.sh --reassignment-json-file /tmp/partition-reassignment.json --
execute \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-config
/tmp/config.properties
```

要在 Cluster:kafka-cluster 以及每个涉及的主题上验证分区重新分配、Describe 和 AlterConfigs 权限是必需的。

```
bin/kafka-reassign-partitions.sh --reassignment-json-file /tmp/partition-reassignment.json --
verify \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-config
/tmp/config.properties
```

4.5.4. 试用红帽单点登录授权服务

本例介绍了如何使用带有 keycloak 授权的红帽单点登录授权服务。使用 Red Hat Single Sign-On Authorization Services 对 Kafka 客户端实施访问限制。红帽单点登录授权服务使用授权范围、策略和权限来定义并应用对资源的访问控制。

红帽单点登录授权服务 REST 端点为经过身份验证的用户提供所授予资源权限的列表。授权列表（权限）从红帽单点登录服务器获取，作为 Kafka 客户端建立身份验证会话后的第一个操作。这个列表会在后台刷新，以便检测到对授权的更改。授权会在每个用户会话的 Kafka 代理上进行缓存和强制实施，以提供快速的授权决策。

AMQ Streams 提供了两个示例文件，其中包含用于设置 Red Hat Single Sign-On 的部署工件：

kafka-ephemeral-oauth-single-keycloak-authz.yaml

使用红帽单点登录为基于 OAuth 2.0 令牌的授权配置 Kafka 自定义资源示例。您可以使用自定义资源来部署使用 keycloak 授权和基于令牌的 oauth 身份验证的 Kafka 集群。

kafka-authz-realm.json

配置有示例组、用户、角色和客户端的红帽单点登录域示例。您可以将该域导入到 Red Hat Single Sign-On 实例，以设置精细的权限来访问 Kafka。

如果要通过 Red Hat Single Sign-On 尝试示例，请使用这些文件按照所示的顺序执行本节中概述的任务。

1. 访问红帽单点登录管理控制台
2. 使用 Red Hat Single Sign-On 授权部署 Kafka 集群

3. 为 CLI Kafka 客户端会话准备 TLS 连接

4. 使用 CLI Kafka 客户端会话检查对 Kafka 的授权访问权限

Authentication

当您配置基于令牌的 oauth 身份验证时，您可以将 jwksEndpointUri 指定为本地 JWT 验证的 URI。在配置 keycloak 授权时，您可以将 tokenEndpointUri 指定为 Red Hat Single Sign-On 令牌端点的 URI。两个 URI 的主机名必须相同。

使用组或角色策略定向权限

在红帽单点登录中，启用服务帐户的机密客户端可以使用客户端 ID 和机密以自己的名称向服务器进行身份验证。对于通常使用自有名称的微服务，而非作为特定用户的代理（如网站）而言，这非常方便。服务帐户可以像普通用户一样分配角色。但是，他们不能分配有组。因此，如果要使用服务帐户为微服务设置权限，则无法使用组策略，而是应使用角色策略。相反，如果您只想将某些权限限制为需要使用用户名和密码进行身份验证的普通用户帐户，您可以实现这一点，作为使用组策略而不是角色策略的一个副作用。本例中用于以 ClusterManager 开头的权限。通常使用 CLI 工具以交互方式执行集群管理。在使用生成的访问令牌验证 Kafka 代理前，要求用户登录是合适的。在这种情况下，访问令牌代表特定的用户，而不是客户端应用。

4.5.4.1. 访问红帽单点登录管理控制台

设置红帽单点登录，然后连接到其管理控制台，再添加预配置的域。使用示例 kafka-authz-realm.json 文件导入该域。您可以在管理控制台中检查为域定义的授权规则。规则授予对配置为使用 Red Hat Single Sign-On 域示例的 Kafka 集群上资源的访问权限。

先决条件

- 正在运行的 OpenShift 集群。
- AMQ Streams 示例/security/keycloak-authorization/kafka-authz-realm.json 文件包含预配置域。

流程

1. 使用 Red Hat Single Sign-On Operator 安装 Red Hat Single Sign-On 服务器，如 Red Hat Single Sign-On 文档中的服务器安装和配置 中所述。

2. 等待红帽单点登录实例运行。

3. 获取外部主机名，以便能够访问管理控制台。

```
NS=sso
oc get ingress keycloak -n $NS
```

在本例中，我们假定红帽单点登录服务器正在 sso 命名空间中运行。

4. 获取 admin 用户的密码。

```
oc get -n $NS pod keycloak-0 -o yaml | less
```

密码存储为 secret，因此获取 Red Hat Single Sign-On 实例的配置 YAML 文件，以标识 secret 的名称(secretKeyRef.name)。

5. 使用机密的名称获取明文密码。

```
SECRET_NAME=credential-keycloak
oc get -n $NS secret $SECRET_NAME -o yaml | grep PASSWORD | awk '{print $2}' |
base64 -D
```

在本例中，我们假设 secret 的名称是 credentials -keycloak。

6. 使用用户名 admin 和您获取的密码登录管理控制台。

使用 https://HOSTNAME 访问 OpenShift 入口。

现在，您可以使用管理控制台将示例域上传到红帽单点登录。

7. 单击 Add Realm 以 导入示例域。

8. 添加 example /security/keycloak-authorization/kafka-authz-realm.json 文件，然后单击 Create。

现在，在管理控制台中，kafka-authz 作为当前域。

默认视图显示 Master 域。

9. 在 Red Hat Single Sign-On Admin Console 中，进入 Clients > kafka > Authorization > Settings，并检查 Decision Strategy 是否已设置为 Affirmative。

一个策略意味着，客户端必须至少满足一个策略才能访问 Kafka 集群。

10. 在红帽单点管理控制台中，前往 组、用户、角色和 客户端 以查看域配置。

组

组用于创建用户组 和设置用户权限。组是分配了名称的用户组。它们用于将用户划分到地理、组织或部门单元中。组可以链接到 LDAP 身份提供程序。您可以通过自定义 LDAP 服务器 admin 用户界面让用户成为组的成员，例如，授予 Kafka 资源的权限。

用户

用户 用于创建用户。本例中定义了 alice 和 bob。Alice 是 ClusterManager 组的成员，bob 是 ClusterManager-my-cluster 组的成员。用户可以存储在 LDAP 身份提供商中。

角色

角色 将用户或客户端标记为具有特定权限。角色是类似于组的概念。它们通常用于为用户 添加 组织角色，并具有必要的权限。角色不能存储在 LDAP 身份提供商中。如果 LDAP 是必需的，您可以改为使用组，并将红帽单点登录角色添加到组，以便在为用户分配组时也获得相应的角色。

客户端

客户端 可以具有特定的配置。在本例中，配置了 kafka、kafka-cli、team-a-client 和 team-b-client 客户端。

- kafka 客户端供 Kafka 代理用于执行必要的 OAuth 2.0 通信，以进行访问令牌验证。此客户端还包含授权服务资源定义、策略和授权范围，用于在 Kafka 代理上执行授权。授权配置在 Authorization 选项卡的 kafka 客户端中定义，在 Settings 选项卡中打开 授权启用 时可看到该配置。

- kafka-cli 客户端是一个公共客户端，在使用用户名和密码进行身份验证时供

Kafka 命令行工具使用，以获取访问令牌或刷新令牌。

- team-a-client 和 team-b-client 客户端是机密客户端，代表对某些 Kafka 主题有部分访问权限的服务。

11. 在 Red Hat Single Sign-On Admin Console 中，进入 Authorization > Permissions，以查看使用为域定义的资源和策略所授予的权限。

例如，kafka 客户端有以下权限：

```
Dev Team A can write to topics that start with x_ on any cluster
Dev Team B can read from topics that start with x_ on any cluster
Dev Team B can update consumer group offsets that start with x_ on any cluster
ClusterManager of my-cluster Group has full access to cluster config on my-cluster
ClusterManager of my-cluster Group has full access to consumer groups on my-cluster
ClusterManager of my-cluster Group has full access to topics on my-cluster
```

Dev 团队 A

Dev 团队 A 域角色可以写入任何群集上以 x_ 开头的主题。这结合了名为 Topic:x_*、Describe 和 Write 范围以及 Dev Team A 策略 的资源。Dev 团队 A 策略 与具有名为 Dev Team A 的域角色的所有用户匹配。

Dev 团队 B

Dev 团队 B 域角色可以从任何群集上以 x_ 开头的主题中读取。这结合了 主题：x_*、Group:x_* 资源、Describe 和 Read 范围以及 Dev Team B 策略。Dev 团队 B 策略匹配具有名为 Dev Team B 的域角色的所有用户。匹配的用户和客户端能够从主题中读取，并为名称以 x_ 开头的主题和消费者组更新所消耗的偏移量。

4.5.4.2. 使用 Red Hat Single Sign-On 授权部署 Kafka 集群

部署一个 Kafka 集群以连接到 Red Hat Single Sign-On 服务器。使用 kafka-ephemeral-oauth-single-keycloak-authz.yaml 文件将 Kafka 集群部署为 Kafka 自定义资源。这个示例使用 keycloak 授权和 oauth 身份验证部署单节点 Kafka 集群。

先决条件

- 红帽单点登录授权服务器部署到 OpenShift 集群，并加载了示例域。

- Cluster Operator 已部署到 OpenShift 集群中。

- AMQ Streams 示例/security/keycloak-authorization/kafka-ephemeral-oauth-single-keycloak-authz.yaml 自定义资源。

流程

1. 使用您部署的 Red Hat Single Sign-On 实例的主机名，为 Kafka 代理准备信任存储证书，以便与红帽单点登录服务器通信。

```
SSO_HOST=SSO-HOSTNAME
SSO_HOST_PORT=$SSO_HOST:443
STOREPASS=storepass

echo "Q" | openssl s_client -showcerts -connect $SSO_HOST_PORT 2>/dev/null | awk
' /BEGIN CERTIFICATE/,/END CERTIFICATE/ { print $0 } ' > /tmp/sso.crt
```

   证书是必需的，因为 OpenShift ingress 用于进行安全(HTTPS)连接。

2. 将证书作为机密部署到 OpenShift。

```
oc create secret generic oauth-server-cert --from-file=/tmp/sso.crt -n $NS
```

3. 将主机名设置为环境变量

```
SSO_HOST=SSO-HOSTNAME
```

4. 创建并部署示例 Kafka 集群。

```
cat examples/security/keycloak-authorization/kafka-ephemeral-oauth-single-keycloak-
authz.yaml | sed -E 's#\${SSO_HOST}"'#$SSO_HOST#" | oc create -n $NS -f -
```

4.5.4.3. 为 CLI Kafka 客户端会话准备 TLS 连接

为交互式 CLI 会话创建新 pod。使用红帽单点登录证书为 TLS 连接设置信任存储。truststore 是连接到 Red Hat Single Sign-On 和 Kafka 代理。

先决条件

- 红帽单点登录授权服务器部署到 OpenShift 集群，并加载了示例域。

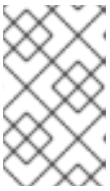
  在 Red Hat Single Sign-On Admin 控制台中，检查分配给客户端的角色是否显示在 Clients > Service Account Roles 中。

- 配置为连接到 Red Hat Single Sign-On 的 Kafka 集群已部署到 OpenShift 集群。

流程

1. 使用 AMQ Streams Kafka 镜像运行一个新的交互式 pod 容器，以连接到正在运行的 Kafka 代理。

```
NS=sso
oc run -ti --restart=Never --image=registry.redhat.io/amq7/amq-streams-kafka-28-rhel7:1.8.0 kafka-cli -n $NS -- /bin/sh
```



注意

如果 oc 超时等待镜像下载，后续尝试可能会导致 AlreadyExists 错误。

2. 附加到 pod 容器。

```
oc attach -ti kafka-cli -n $NS
```

3. 使用红帽单点登录实例的主机名，使用 TLS 为客户端连接准备证书。

```
SSO_HOST=SSO-HOSTNAME
SSO_HOST_PORT=$SSO_HOST:443
STOREPASS=storepass

echo "Q" | openssl s_client -showcerts -connect $SSO_HOST_PORT 2>/dev/null | awk ' /BEGIN CERTIFICATE/,/END CERTIFICATE/ { print $0 } ' > /tmp/sso.crt
```

4. 为 TLS 连接到 Kafka 代理创建信任存储。

```
keytool -keystore /tmp/truststore.p12 -storetype pkcs12 -alias sso -storepass $STOREPASS -import -file /tmp/sso.crt -noprompt
```

5. 使用 Kafka bootstrap 地址作为 Kafka 代理的主机名，使用 tls 侦听器端口(9093)为 Kafka 代理准备证书。

```
KAFKA_HOST_PORT=my-cluster-kafka-bootstrap:9093
STOREPASS=storepass

echo "Q" | openssl s_client -showcerts -connect $KAFKA_HOST_PORT 2>/dev/null |
awk ' /BEGIN CERTIFICATE/,/END CERTIFICATE/ { print $0 } ' > /tmp/my-cluster-
kafka.crt
```

6. 将 Kafka 代理的证书添加到信任存储中。

```
keytool -keystore /tmp/truststore.p12 -storetype pkcs12 -alias my-cluster-kafka -
storepass $STOREPASS -import -file /tmp/my-cluster-kafka.crt -noprompt
```

保持 会话处于打开状态，以检查授权访问权限。

4.5.4.4. 使用 CLI Kafka 客户端会话检查对 Kafka 的授权访问权限

利用交互式 CLI 会话，检查通过红帽单点登录域应用的授权规则。使用 Kafka 的示例制作者和消费者客户端应用检查，以创建具有不同访问权限级别的用户和服务帐户的主题。

使用 team-a-client 和 team-b-client 客户端检查授权规则。使用 alice admin 用户对 Kafka 执行额外的管理任务。

本例中使用的 AMQ Streams Kafka 镜像包含 Kafka 制作者和使用者二进制文件。

先决条件

- zookeeper 和 Kafka 在 OpenShift 集群中运行，以便能够发送和接收消息。
- 交互式 CLI Kafka 客户端会话 已启动。

  Apache Kafka 下载.

设置客户端和管理员用户配置

1.

I.

使用 team-a-client 客户端的身份验证属性准备 Kafka 配置文件。

```
SSO_HOST=SSO-HOSTNAME

cat > /tmp/team-a-client.properties << EOF
security.protocol=SASL_SSL
ssl.truststore.location=/tmp/truststore.p12
ssl.truststore.password=$STOREPASS
ssl.truststore.type=PKCS12
sasl.mechanism=OAUTHBEARER
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginM
odule required \
  oauth.client.id="team-a-client" \
  oauth.client.secret="team-a-client-secret" \
  oauth.ssl.truststore.location="/tmp/truststore.p12" \
  oauth.ssl.truststore.password="$STOREPASS" \
  oauth.ssl.truststore.type="PKCS12" \
  oauth.token.endpoint.uri="https://$SSO_HOST/auth/realms/kafka-
authz/protocol/openid-connect/token" ;
sasl.login.callback.handler.class=io.strimzi.kafka.oauth.client.JaasClientOauthLoginC
allbackHandler
EOF
```

使用 SASL OAUTHBEARER 机制。这种机制需要客户端 ID 和客户端 secret，这意味着客户端首先连接到红帽单点登录服务器来获取访问令牌。然后，客户端连接到 Kafka 代理并使用访问令牌进行身份验证。

2.

使用 team-b-client 客户端的身份验证属性准备 Kafka 配置文件。

```
cat > /tmp/team-b-client.properties << EOF
security.protocol=SASL_SSL
ssl.truststore.location=/tmp/truststore.p12
ssl.truststore.password=$STOREPASS
ssl.truststore.type=PKCS12
sasl.mechanism=OAUTHBEARER
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginM
odule required \
  oauth.client.id="team-b-client" \
  oauth.client.secret="team-b-client-secret" \
  oauth.ssl.truststore.location="/tmp/truststore.p12" \
  oauth.ssl.truststore.password="$STOREPASS" \
  oauth.ssl.truststore.type="PKCS12" \
  oauth.token.endpoint.uri="https://$SSO_HOST/auth/realms/kafka-
authz/protocol/openid-connect/token" ;
sasl.login.callback.handler.class=io.strimzi.kafka.oauth.client.JaasClientOauthLoginC
allbackHandler
EOF
```

3.

通过使用 curl 并执行密码授权身份验证来获取刷新令牌，从而对 admin 用户 alice 进行身

份验证。

```
USERNAME=alice
PASSWORD=alice-password

GRANT_RESPONSE=$(curl -X POST "https://$SSO_HOST/auth/realms/kafka-
authz/protocol/openid-connect/token" -H 'Content-Type: application/x-www-form-
urlencoded' -d
"grant_type=password&username=$USERNAME&password=$PASSWORD&client_id=
kafka-cli&scope=offline_access" -s -k)

REFRESH_TOKEN=$(echo $GRANT_RESPONSE | awk -F "refresh_token\":\"" '{printf
$2}' | awk -F "\"" '{printf $1}')
```

刷新令牌是一个离线令牌，寿命很长，且不会过期。

4. *使用 admin 用户 alice 的身份验证属性准备 Kafka 配置文件。*

```
cat > /tmp/alice.properties << EOF
security.protocol=SASL_SSL
ssl.truststore.location=/tmp/truststore.p12
ssl.truststore.password=$STOREPASS
ssl.truststore.type=PKCS12
sasl.mechanism=OAUTHBEARER
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginM
odule required \
  oauth.refresh.token="$REFRESH_TOKEN" \
  oauth.client.id="kafka-cli" \
  oauth.ssl.truststore.location="/tmp/truststore.p12" \
  oauth.ssl.truststore.password="$STOREPASS" \
  oauth.ssl.truststore.type="PKCS12" \
  oauth.token.endpoint.uri="https://$SSO_HOST/auth/realms/kafka-
authz/protocol/openid-connect/token" ;
sasl.login.callback.handler.class=io.strimzi.kafka.oauth.client.JaasClientOauthLoginC
allbackHandler
EOF
```

kafka-cli 公共客户端用于 sasl .jaas.config 中的 oauth.client. id。由于它是公共客户端，因此不需要机密。客户端使用上一步中验证的刷新令牌进行身份验证。刷新令牌在后台请求访问令牌，然后发送到 Kafka 代理进行身份验证。

生成具有授权访问权限的消息

使用 team-a-client 配置检查您可以向以 a_ 或 x_ 开头的主题生成消息。

1. 写入到 主题 my 主题。

```
bin/kafka-console-producer.sh --broker-list my-cluster-kafka-bootstrap:9093 --topic
my-topic \
  --producer.config=/tmp/team-a-client.properties
First message
```

此请求 返回未授权访问主题：[my-topic] 错误。

team-a-client 具有开发团队 A 角色，允许它对以 a_ 开头的主题执行任何受支持的操作，但只能写入 x_ 开头的主题。名为 my-topic 的主题都不匹配这些规则。

2. 写入主题 a_messages。

```
bin/kafka-console-producer.sh --broker-list my-cluster-kafka-bootstrap:9093 --topic
a_messages \
  --producer.config /tmp/team-a-client.properties
First message
Second message
```

成功生成消息到 Kafka。

3. 按 CTRL+C 退出 CLI 应用。

4. 检查 Kafka 容器日志中请求的 Authorization GRANTED 的调试日志。

```
oc logs my-cluster-kafka-0 -f -n $NS
```

使用具有授权访问权限的信息

使用 team-a-client 配置来使用来自主题 a_messages 的消息。

1. 从主题 a_messages 获取消息.

```
bin/kafka-console-consumer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --
topic a_messages \
  --from-beginning --consumer.config /tmp/team-a-client.properties
```

请求返回一个错误，因为 team-a-client 的 Dev 团队 A 角色仅有权访问名称以 a_ 开头的消费者组。

2. 更新 team-a-client 属性，以指定允许使用的自定义使用者组。

```
bin/kafka-console-consumer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --topic a_messages \
  --from-beginning --consumer.config /tmp/team-a-client.properties --group a_consumer_group_1
```

使用者从 a_messages 主题接收所有消息。

使用授权访问权限管理 Kafka

team-a-client 是一个没有集群级别访问权限的帐户，但可以用于一些管理操作。

1. 列出主题.

```
bin/kafka-topics.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --command-config /tmp/team-a-client.properties --list
```

a_messages 主题返回。

2. 列出消费者组。

```
bin/kafka-consumer-groups.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --command-config /tmp/team-a-client.properties --list
```

a_consumer_group_1 consumer 组返回。

获取集群配置详情。

```
bin/kafka-configs.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --command-config /tmp/team-a-client.properties \
  --entity-type brokers --describe --entity-default
```

请求返回一个错误，因为操作需要 team-a-client 没有集群级别权限。

使用具有不同权限的客户端

使用 team-b-client 配置向以 b_ 开头的主题生成消息。

1. 写入主题 a_messages。

```
bin/kafka-console-producer.sh --broker-list my-cluster-kafka-bootstrap:9093 --topic
a_messages \
  --producer.config /tmp/team-b-client.properties
Message 1
```

此请求将返回未授权访问主题：[a_messages] 错误。

2. 写入到主题 b_messages。

```
bin/kafka-console-producer.sh --broker-list my-cluster-kafka-bootstrap:9093 --topic
b_messages \
  --producer.config /tmp/team-b-client.properties
Message 1
Message 2
Message 3
```

成功生成消息到 Kafka。

3. 写入主题 x_messages。

```
bin/kafka-console-producer.sh --broker-list my-cluster-kafka-bootstrap:9093 --topic
x_messages \
  --producer.config /tmp/team-b-client.properties
Message 1
```

未授权访问主题：[x_messages] 错误返回，team-b-client 只能从主题 x_messages 中读取。

4. 使用 team-a-client 写入主题 x_messages。

```
bin/kafka-console-producer.sh --broker-list my-cluster-kafka-bootstrap:9093 --topic
x_messages \
  --producer.config /tmp/team-a-client.properties
Message 1
```

此请求 返回未授权访问主题：[x_messages] 错误。team-a-client 可以写入 x_messages 主题，但是如果主题尚不存在，则无权创建该主题。在 team-a-client 可以写入 x_messages 主题之前，管理员 高级用户 必须使用正确的配置创建它，如分区和副本的数量。

使用授权 admin 用户管理 Kafka

使用 admin 用户 alice 管理 Kafka。Alice 完全有权管理任何 Kafka 集群中的所有内容。

1. 创建 x_messages 主题，作为 alice。

```
bin/kafka-topics.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --command-
config /tmp/alice.properties \
  --topic x_messages --create --replication-factor 1 --partitions 1
```

主题创建成功。

2. 将所有主题列为 alice.

```
bin/kafka-topics.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --command-
config /tmp/alice.properties --list
bin/kafka-topics.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --command-
config /tmp/team-a-client.properties --list
bin/kafka-topics.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --command-
config /tmp/team-b-client.properties --list
```

admin 用户 alice 可以列出所有主题，而 team-a-client 和 team-b-client 只能列出他们有权访问的主题。

Dev 团队 A 和 Dev 团队 B 角色均对以 x_ 开头的主题具有 权限，但它们无法看到其他团队的主题，因为它们没有描述它们 的权限。

3. 使用 team-a-client 生成到 x_messages 主题的信息：

```
bin/kafka-console-producer.sh --broker-list my-cluster-kafka-bootstrap:9093 --topic
x_messages \
```

```
 --producer.config /tmp/team-a-client.properties
Message 1
Message 2
Message 3
```

当 alice 创建 x_messages 主题时，会成功生成消息到 Kafka。

4. 使用 team-b-client 生成消息到 x_messages 主题。

```
bin/kafka-console-producer.sh --broker-list my-cluster-kafka-bootstrap:9093 --topic
x_messages \
 --producer.config /tmp/team-b-client.properties
Message 4
Message 5
```

此请求返回未授权访问主题：[x_messages] 错误。

5. 使用 team-b-client 使用来自 x_messages 主题的信息：

```
bin/kafka-console-consumer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --
topic x_messages \
 --from-beginning --consumer.config /tmp/team-b-client.properties --group
x_consumer_group_b
```

使用者从 x_messages 主题接收所有消息。

6. 使用 team-a-client 使用来自 x_messages 主题的消息。

```
bin/kafka-console-consumer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --
topic x_messages \
 --from-beginning --consumer.config /tmp/team-a-client.properties --group
x_consumer_group_a
```

此请求返回未授权访问主题：[x_messages] 错误。

7. 使用 team-a-client 使用来自以 a_ 开头的消费者组的消息。

```
bin/kafka-console-consumer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --
topic x_messages \
```

```
  --from-beginning --consumer.config /tmp/team-a-client.properties --group
a_consumer_group_a
```

此请求返回未授权访问主题：[x_messages] 错误。

Dev 团队 A 对以 x_ 开头的主题没有读取访问权限。

8. 使用 alice 生成指向 x_messages 主题的消息。

```
bin/kafka-console-consumer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --
topic x_messages \
  --from-beginning --consumer.config /tmp/alice.properties
```

成功生成消息到 Kafka。

Alice 可以从任何主题读取或写入。

9. 使用 alice 读取集群配置。

```
bin/kafka-configs.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --command-
config /tmp/alice.properties \
  --entity-type brokers --describe --entity-default
```

本例的集群配置为空。

其它资源

- 服务器安装和配置
- 将 Red Hat Single Sign-On Authorization Services 映射到 Kafka 授权模型

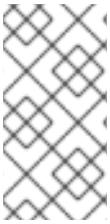
第 5 章 使用 AMQ STREAMS OPERATOR

使用 AMQ Streams 操作器来管理您的 Kafka 集群，以及 Kafka 主题和用户。

5.1. 使用 CLUSTER OPERATOR

Cluster Operator 用于部署 Kafka 集群和其他 Kafka 组件。

Cluster Operator 使用 YAML 安装文件进行部署。



注意

在 OpenShift 中，Kafka Connect 部署可以纳入 Source2Image 功能，以提供添加额外连接器的便捷方式。

其它资源

- 在 OpenShift 的 Deploying and upgradeding AMQ Streams 中部署 Cluster Operator。
- Kafka 集群配置.

5.1.1. Cluster Operator 配置

您可以使用支持的环境变量及其日志记录配置来配置 Cluster Operator。

环境变量与用于部署 Cluster Operator 镜像的容器配置相关。有关 镜像 配置的详情请参考 第 13.1.6 节 “image”。

STRIMZI_NAMESPACE

Operator 应操作的命名空间的逗号分隔列表。如果没有设置、设置为空字符串或设置为 *，Cluster Operator 将在所有命名空间中运行。Cluster Operator 部署可能会使用 OpenShift Downward API 自动将其设置为 Cluster Operator 部署到的命名空间。

Cluster Operator 命名空间的配置示例

```
env:
  - name: STRIMZI_NAMESPACE
    valueFrom:
      fieldRef:
        fieldPath: metadata.namespace
```

STRIMZI_FULL_RECONCILIATION_INTERVAL_MS

可选，默认为 120000 ms。定期协调之间的间隔，以毫秒为单位。

STRIMZI_OPERATION_TIMEOUT_MS

可选，默认为 300000 ms。内部操作的超时，以毫秒为单位。当在常规 OpenShift 操作需要比平时更长的集群中使用 AMQ Streams 时，这个值应该会增加（例如，因为 Docker 镜像下载速度较慢）。

STRIMZI_OPERATIONS_THREAD_POOL_SIZE

可选，默认为 10 worker 线程池大小，用于集群 Operator 运行的各种异步和阻止操作。

STRIMZI_OPERATOR_NAMESPACE

运行 AMQ Streams Cluster Operator 的命名空间的名称。不要手动配置这个变量。使用 OpenShift Downward API。

```
env:
  - name: STRIMZI_OPERATOR_NAMESPACE
    valueFrom:
      fieldRef:
        fieldPath: metadata.namespace
```

STRIMZI_OPERATOR_NAMESPACE_LABELS

可选。运行 AMQ Streams Cluster Operator 的命名空间的标签。命名空间标签用于在网络策略中配置命名空间选择器，以允许 AMQ Streams Cluster Operator 只能使用这些标签从命名空间中访问操作对象。如果没有设置，网络策略中的命名空间选择器会被配置为允许从 OpenShift 集群的任何命名空间访问 AMQ Streams Cluster Operator。

```
env:
  - name: STRIMZI_OPERATOR_NAMESPACE_LABELS
    value: label1=value1,label2=value2
```

STRIMZI_CUSTOM_RESOURCE_SELECTOR

可选。指定标签选择器，用于过滤操作器处理的自定义资源。Operator 仅在设置了指定标签的自定义资源上运行。没有这些标签的资源不会被 Operator 看到。标签选择器适用于 Kafka、Kafka Connect、KafkaConnectS2I、KafkaBridge、KafkaMirrorMaker 和 KafkaMirrorMaker2 资源。只有在对应的 Kafka 和 Kafka Connect 集群具有匹配的标签时，才会执行 Kafka Rebalance 和 KafkaConnector 资源。

```
env:
  - name: STRIMZI_CUSTOM_RESOURCE_SELECTOR
    value: label1=value1,label2=value2
```

STRIMZI_LABELS_EXCLUSION_PATTERN

可选，默认的 regex 模式是 ^app.kubernetes.io/(?!part-of)。指定 regex 排除模式，用于将标签传播从主自定义资源过滤到其子资源。排除过滤器的标签不会应用到模板部分中的标签，如 spec.kafka.template.pod.metadata.labels。*

```
env:
  - name: STRIMZI_LABELS_EXCLUSION_PATTERN
    value: "^key1.*"
```

STRIMZI_KAFKA_IMAGES

必需。这提供了从 Kafka 版本到相应 Docker 镜像的映射，其中包含该版本的 Kafka 代理。所需语法为空格或用逗号分开的 <version>=<image> 对。例如 2.7.0=registry.redhat.io/amq7/amq-streams-kafka-27-rhel7:1.8.0, 2.8.0=registry.redhat.io/amq7/amq-streams-kafka-28-rhel7:1.8.0。当指定 Kafka.spec.kafka.version 属性但没有指定 Kafka 资源 中的 Kafka.spec.kafka.image 时会使用此方法。

STRIMZI_DEFAULT_KAFKA_INIT_IMAGE

可选，默认 registry.redhat.io/amq7/amq-streams-rhel7-operator:1.8.0。如果没有将镜像指定为 Kafka 资源中的 kafka-init-image，则在代理之前启动的 init 容器的镜像名称（即机架支持）没有指定为 kafka-init-image。

STRIMZI_KAFKA_CONNECT_IMAGES

必需。这提供了从 Kafka 版本到相应 Docker 镜像的映射，其中包含该版本的 Kafka 连接。所需语法为空格或用逗号分开的 <version>=<image> 对。例如 2.7.0=registry.redhat.io/amq7/amq-streams-kafka-27-rhel7:1.8.0, 2.8.0=registry.redhat.io/amq7/amq-streams-kafka-28-rhel7:1.8.0。这在指定 KafkaConnect.spec.version 属性而不是 KafkaConnect.spec.image 时使用。

STRIMZI_KAFKA_CONNECT_S2I_IMAGES

必需。这提供了从 Kafka 版本到相应 Docker 镜像的映射，其中包含该版本的 Kafka 连接。所需语法为空格或用逗号分开的 <version>=<image> 对。例如 2.7.0=registry.redhat.io/amq7/amq-streams-kafka-27-rhel7:1.8.0, 2.8.0=registry.redhat.io/amq7/amq-streams-kafka-28-

rhel7:1.8.0。这在指定 KafkaConnectS2I.spec.version 属性而不是 KafkaConnectS2I.spec.image 时使用。

STRIMZI_KAFKA_MIRROR_MAKER_IMAGES

必需。这提供了从 Kafka 版本到相应 Docker 镜像的映射，其中包含该版本的 Kafka 镜像制作器。所需语法为空格或用逗号分开的 <version>=<image> 对。例如 2.7.0=registry.redhat.io/amq7/amq-streams-kafka-27-rhel7:1.8.0, 2.8.0=registry.redhat.io/amq7/amq-streams-kafka-28-rhel7:1.8.0。这在指定 KafkaMirrorMaker.spec.version 属性而不是 KafkaMirrorMaker.spec.image 时使用。

STRIMZI_DEFAULT_TOPIC_OPERATOR_IMAGE

可选，默认 registry.redhat.io/amq7/amq-streams-rhel7-operator:1.8.0。部署主题 Operator 时要使用的镜像名称，如果没有将镜像指定为 Kafka 资源 中的 Kafka.spec.entityOperator.topicOperator.image。

STRIMZI_DEFAULT_USER_OPERATOR_IMAGE

可选，默认 registry.redhat.io/amq7/amq-streams-rhel7-operator:1.8.0。在部署用户 operator 时要使用的镜像名称，如果没有在 Kafka 资源中指定为 Kafka.spec.entityOperator.userOperator.image 镜像名称 。

STRIMZI_DEFAULT_TLS_SIDECAR_ENTITY_OPERATOR_IMAGE

可选，默认 registry.redhat.io/amq7/amq-streams-kafka-28-rhel7:1.8.0。在部署 sidecar 容器时用作默认的镜像名称，该容器为 Entity Operator 提供 TLS 支持，如果没有将镜像指定为 Kafka 资源 中的 Kafka.spec.entityOperator.tlsSidecar.image。

STRIMZI_IMAGE_PULL_POLICY

可选。ImagePullPolicy，它将应用到由 AMQ Streams Cluster Operator 管理的所有 pod 中的容器。有效值有 Always、ifNotPresent 和 Never。如果未指定，则将使用 OpenShift 的默认值。更改策略会导致所有 Kafka、Kafka Connect 和 Kafka MirrorMaker 集群滚动更新。

STRIMZI_IMAGE_PULL_SECRETS

可选。以逗号分隔的 Secret 名称列表。此处引用的 secret 包含从中提取容器镜像的容器注册表的凭据。secret 在 imagePullSecrets 字段中用于 Cluster Operator 创建的所有 Pod。更改此列表会导致所有 Kafka、Kafka Connect 和 Kafka MirrorMaker 集群进行滚动更新。

STRIMZI_KUBERNETES_VERSION

可选。覆盖从 API 服务器检测到的 Kubernetes 版本信息。

Kubernetes 版本覆盖的配置示例

```
env:
  - name: STRIMZI_KUBERNETES_VERSION
    value: |
           major=1
           minor=16
           gitVersion=v1.16.2
           gitCommit=c97fe5036ef3df2967d086711e6c0c405941e14b
           gitTreeState=clean
           buildDate=2019-10-15T19:09:08Z
           goVersion=go1.12.10
           compiler=gc
           platform=linux/amd64
```

KUBERNETES_SERVICE_DNS_DOMAIN

可选。覆盖默认的 OpenShift DNS 域名后缀。

默认情况下，OpenShift 集群中分配的服务的 DNS 域名使用默认的后缀 cluster.local。

例如，对于代理 kafka-0 ：

```
<cluster-name>-kafka-0.<cluster-name>-kafka-brokers.<namespace>.svc.cluster.local
```

DNS 域名被添加到用于主机名验证的 Kafka 代理证书中。

如果您在集群中使用不同的 DNS 域名后缀，请将 KUBERNETES_SERVICE_DNS_DOMAIN 环境变量从默认值改为您用来与 Kafka 代理建立连接的变量。

STRIMZI_CONNECT_BUILD_TIMEOUT_MS

可选，默认为 300000 ms。使用额外连接点构建新 Kafka Connect 镜像的超时时间，以毫秒为单位。使用 AMQ Streams 构建包含多个连接器的容器镜像或使用速度较慢的容器注册表时，应增加这个值。

STRIMZI_FEATURE_GATES

可选。启用或禁用由功能门控制的功能和功能。有关每个功能门的详情请参考 第 5.1.1.1 节 “功能门”。

5.1.1.1. 功能门

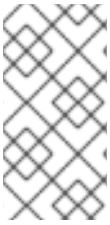
AMQ Streams 操作器支持 功能门来 启用或禁用某些功能。启用功能门会更改相关 Operator 的行为，并将该功能引入您的 AMQ Streams 部署。

功能门的默认状态为 enabled 或 disabled。要修改功能门的默认状态，请在操作器的配置中使用 STRIMZI_FEATURE_GATES 环境变量。您可以使用这个环境变量修改多个功能门。

功能门有三个成熟度阶段：

- alpha - 通常默认禁用
- Beta - 通常默认启用
- 正式发行(GA)- 通常默认启用

Alpha 阶段功能可能实验性或不稳定，可能会有所变化，或者没有经过足够的测试以供生产环境使用。测试版阶段功能经过了良好的测试，其功能不太可能改变。GA 阶段的功能是稳定的，未来不应改变。如果 alpha 和 beta 阶段功能不有用，则会删除它们。



注意

功能门可能会在到达 GA 时被删除。这意味着该功能已合并到 AMQ Streams 核心功能中，无法再禁用。

表 5.1. 当所有功能门和 AMQ Streams 版本移到 alpha、beta 或 GA 时

功能门	alpha	Beta	GA
ControlPlaneListener	1.8	-	-
ServiceAccountPatching	1.8	-	-

配置功能门

您可以使用操作器配置中的 STRIMZI_FEATURE_GATES 环境变量来配置功能门。指定以逗号分隔的功能门名称和前缀列表。+ 前缀启用功能门，而 - 前缀可禁用它。

启用 FeatureGate1 并禁用 FeatureGate2 的功能门配置示例

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: +FeatureGate1,-FeatureGate2
```

5.1.1.1.1. control plane 侦听程序功能门

使用 ControlPlaneListener 功能门更改 Kafka 集群中用于代理间通信的通信路径。

OpenShift 控制平面管理在工作程序节点上运行的工作负载。Kubernetes API 服务器和控制器管理器等服务在 control plane 上运行。OpenShift 数据平面向容器（包括 CPU、内存、网络和存储）提供资源。

在 AMQ Streams 中，control plane 流量由维护 Kafka 集群所需状态的控制器连接组成。数据平面流量主要由领导代理和后续代理之间的数据复制组成。

当禁用 ControlPlaneListener 功能门时，control plane 和数据平面流量会通过端口 9091 上的相同内部监听程序。这是引入功能门前的默认行为。

当启用 ControlPlaneListener 时，control plane 流量通过端口 9090 上的专用 control plane 侦听程序进行。数据平面流量在端口 9091 上继续使用内部监听程序。

使用 control plane 侦听器可能会提高性能，因为重要的控制器连接（如分区领导更改）不会因为代理间数据复制而延迟。

启用 control plane 侦听程序功能门

ControlPlaneListener 功能门处于 alpha 阶段，默认状态为 disabled。要启用它，请在 Cluster Operator 配置中的 STRIMZI_FEATURE_GATES 环境变量中指定 +ControlPlaneListener。

在以下情况下必须禁用此功能门：

- 从 AMQ Streams 1.7 及更早版本升级
- 降级到 AMQ Streams 1.7 及更早版本

注意

ControlPlaneListener 功能门是在 AMQ Streams 1.8 中引入的，它应该在进入 beta 阶段前保留在 alpha 阶段。

5.1.1.1.2. 服务帐户补丁功能门

默认情况下，Cluster Operator 不会更新服务帐户。要允许 Cluster Operator 应用更新，请启用 ServiceAccountPatching 功能门。

将 +ServiceAccountPatching 添加到 Cluster Operator 配置中的 STRIMZI_FEATURE_GATES 环境变量中。

功能门当前处于 alpha 阶段并默认禁用。启用功能门后，Cluster Operator 会在每次协调中对服务帐户配置应用更新。例如，您可以在已创建操作对象后更改服务帐户标签和注解。

注意

ServiceAccountPatching 功能门是在 AMQ Streams 1.8 中引入的，它应该会一直处于多个发行版本的 alpha 阶段，然后进入 beta 阶段并默认启用。

5.1.1.2. 通过 ConfigMap 日志记录配置

Cluster Operator 的日志记录由 strimzi-cluster-operator ConfigMap 配置。

安装 Cluster Operator 时会创建一个包含日志配置的 ConfigMap。此 ConfigMap 在文件 install/cluster-operator/050-ConfigMap-strimzi-cluster-operator.yaml 中描述。您可以通过更改此 ConfigMap 中的 data 字段 log4j2.properties 来配置 Cluster Operator 日志。

要更新日志记录配置，您可以编辑 050-ConfigMap-strimzi-cluster-operator.yaml 文件，然后运行以下命令：

```
oc create -f install/cluster-operator/050-ConfigMap-strimzi-cluster-operator.yaml
```

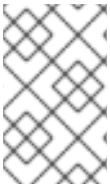
或者，直接编辑 ConfigMap：

```
oc edit configmap strimzi-cluster-operator
```

要更改重新加载间隔的频率，请在创建的 ConfigMap 中的 monitorInterval 选项中设置一个时间（以秒为单位）。

如果在部署 Cluster Operator 时缺少 ConfigMap，则会使用默认的日志记录值。

如果在部署 Cluster Operator 后意外删除 ConfigMap，则会使用最新载入的日志配置。创建新 ConfigMap 以 加载新的日志配置。



注意

不要从 ConfigMap 中删除 monitorInterval 选项。

5.1.1.3. 使用网络策略限制 Cluster Operator 访问

Cluster Operator 可以和它管理的资源在同一命名空间中运行，也可以在单独的命名空间中运行。默认情况下，STRIMZI_OPERATOR_NAMESPACE 环境变量被配置为使用 OpenShift Downward API 来查找 Cluster Operator 在哪个命名空间中运行。如果 Cluster Operator 在与资源相同的命名空间中运行，则只需要本地访问权限，并且 AMQ Streams 允许。

如果 Cluster Operator 在一个独立的命名空间中运行，则 OpenShift 集群中的任何命名空间都可以访问 Cluster Operator，除非配置了网络策略。使用可选的 STRIMZI_OPERATOR_NAMESPACE_LABELS 环境变量，使用命名空间标签为 Cluster Operator 建立网络策略。通过添加命名空间标签，Cluster Operator 的访问权限仅限于指定的命名空间。

为 Cluster Operator 部署配置网络策略

```
#...
env:
  # ...
  - name: STRIMZI_OPERATOR_NAMESPACE_LABELS
    value: label1=value1,label2=value2
  #...
```

–

5.1.1.4. 定期协调

虽然 Cluster Operator 会响应从 OpenShift 集群收到所需集群资源的所有通知，但如果操作器没有运行，或者因任何原因未收到通知，则所需的资源将与正在运行的 OpenShift 集群的状态保持同步。

为了正确处理故障切换，由 Cluster Operator 执行定期协调过程，以便它可以将所需资源的状态与当前集群部署进行比较，以便在所有这些部署中保持一致状态。您可以使用 [STRIMZI_FULL_RECONCILIATION_INTERVAL_MS] 变量为定期协调设置时间间隔。

5.1.2. 调配基于角色的访问控制(RBAC)

要使 Cluster Operator 正常工作，需要在 OpenShift 集群中与资源（如 Kafka、Kafka Connect 等）以及受管资源（如 ConfigMap、 Pod、 Deployment、 StatefulSet 和 Services ）交互。这种权限在 OpenShift 基于角色的访问控制(RBAC)资源中描述：

- ServiceAccount,
- 角色 和 ClusterRole,
- role binding 和 ClusterRoleBinding。

除了使用 ClusterRoleBinding 在其自身 ServiceAccount 中运行外，Cluster Operator 还为需要访问 OpenShift 资源的组件管理一些 RBAC 资源。

OpenShift 还包含特权升级保护，可防止在一个 ServiceAccount 下操作的组件授予授予授权 ServiceAccount 没有的其他 ServiceAccount s 特权。因为 Cluster Operator 必须能够创建 ClusterRoleBindings，以及它管理的资源所需的 RoleBindings，所以 Cluster Operator 还必须具有相同的权限。

5.1.2.1. 委派的权限

当 Cluster Operator 为所需的 Kafka 资源部署资源时，它还会创建 ServiceAccount、RoleBindings 和 ClusterRoleBindings，如下所示：

- *Kafka 代理 pod 使用一个名为 cluster-name-kafka的 ServiceAccount*
  - *使用机架功能时，strimzi-cluster-name-kafka-init ClusterRoleBinding 用来通过 ClusterRole （名为 strimzi-kafka-broker）对集群中的节点授予这个 ServiceAccount 访问权限*
  - *如果没有使用机架功能，且集群没有通过 nodeport 公开，则不会创建绑定*
- *ZooKeeper pod 使用名为 cluster-name-zookeeper的 ServiceAccount*
- *Entity Operator pod 使用名为 cluster-name-entity-operator的 ServiceAccount*
  - *Topic Operator 生成带有状态信息的 OpenShift 事件，因此 ServiceAccount 绑定到名为 strimzi-entity-operator 的 ClusterRole，它通过 strimzi-entity-operator RoleBinding授予此访问权限*
- *KafkaConnect 和 KafkaConnect S2I 资源的 pod 使用名为 cluster-name-cluster-connect的 ServiceAccount*
- *KafkaMirrorMaker 的 pod 使用名为 cluster-name-mirror-maker的 ServiceAccount*
- *KafkaMirrorMaker2 的 pod 使用名为 cluster-name-mirrormaker2的 ServiceAccount*
- *KafkaBridge 的 pod 使用名为 cluster-name-bridge的 ServiceAccount*

5.1.2.2. ServiceAccount

Cluster Operator 最好使用 ServiceAccount 运行：

Cluster Operator 的 ServiceAccount 示例

```
apiVersion: v1
```

```
kind: ServiceAccount
metadata:
  name: strimzi-cluster-operator
  labels:
    app: strimzi
```

然后，Operator 的 Deployment 需要在其 spec.template.spec.serviceAccountName 中指定 ：

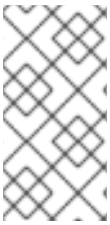
Cluster Operator 部署 的部分示例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-cluster-operator
  labels:
    app: strimzi
spec:
  replicas: 1
  selector:
    matchLabels:
      name: strimzi-cluster-operator
      strimzi.io/kind: cluster-operator
  template:
      # ...
```

注意 12 行，其中 strimzi-cluster-operator ServiceAccount 指定为 serviceAccountName。

5.1.2.3. ClusterRoles

Cluster Operator 需要使用 提供所需资源访问的 ClusterRole 运行。根据 OpenShift 集群设置，可能需要集群管理员来创建 ClusterRole。



注意

集群管理员权限只在创建 ClusterRole 时需要。Cluster Operator 不会在集群管理员帐户下运行。

ClusterRole 遵循最小 权限， 仅包含 Cluster Operator 运行 Kafka、Kafka Connect 和 ZooKeeper 集群所需的权限。第一组分配的权限允许 Cluster Operator 管理 OpenShift 资源，如 StatefulSets、Deployment、 Pod 和 ConfigMap。

Cluster Operator 使用 ClusterRole 在命名空间范围的资源级别和集群范围的资源级别授予权限：

具有 Cluster Operator 资源命名空间资源的 ClusterRole

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: strimzi-cluster-operator-namespaced
  labels:
    app: strimzi
rules:
  - apiGroups:
      - "rbac.authorization.k8s.io"
    resources:
      # The cluster operator needs to access and manage rolebindings to grant Strimzi components cluster permissions
      - rolebindings
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
  - apiGroups:
      - "rbac.authorization.k8s.io"
    resources:
      # The cluster operator needs to access and manage roles to grant the entity operator permissions
      - roles
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
  - apiGroups:
      - ""
    resources:
      # The cluster operator needs to access and delete pods, this is to allow it to monitor pod health and coordinate rolling updates
      - pods
```

```
    # The cluster operator needs to access and manage service accounts to grant Strimzi components cluster permissions
    - serviceaccounts
    # The cluster operator needs to access and manage config maps for Strimzi components configuration
    - configmaps
    # The cluster operator needs to access and manage services and endpoints to expose Strimzi components to network traffic
    - services
    - endpoints
    # The cluster operator needs to access and manage secrets to handle credentials
    - secrets
    # The cluster operator needs to access and manage persistent volume claims to bind them to Strimzi components for persistent data
    - persistentvolumeclaims
  verbs:
    - get
    - list
    - watch
    - create
    - delete
    - patch
    - update
- apiGroups:
    - "kafka.strimzi.io"
  resources:
    # The cluster operator runs the KafkaAssemblyOperator, which needs to access and manage Kafka resources
    - kafkas
    - kafkas/status
    # The cluster operator runs the KafkaConnectAssemblyOperator, which needs to access and manage KafkaConnect resources
    - kafkaconnects
    - kafkaconnects/status
    # The cluster operator runs the KafkaConnectS2IAssemblyOperator, which needs to access and manage KafkaConnectS2I resources
    - kafkaconnects2is
    - kafkaconnects2is/status
    # The cluster operator runs the KafkaConnectorAssemblyOperator, which needs to access and manage KafkaConnector resources
    - kafkaconnectors
    - kafkaconnectors/status
    # The cluster operator runs the KafkaMirrorMakerAssemblyOperator, which needs to access and manage KafkaMirrorMaker resources
    - kafkamirrormakers
    - kafkamirrormakers/status
    # The cluster operator runs the KafkaBridgeAssemblyOperator, which needs to access and manage BridgeMaker resources
    - kafkabridges
    - kafkabridges/status
    # The cluster operator runs the KafkaMirrorMaker2AssemblyOperator, which needs to access and manage KafkaMirrorMaker2 resources
    - kafkamirrormaker2s
    - kafkamirrormaker2s/status
    # The cluster operator runs the KafkaRebalanceAssemblyOperator, which needs to access and manage KafkaRebalance resources
```

```
    - kafkarebalances
    - kafkarebalances/status
  verbs:
    - get
    - list
    - watch
    - create
    - delete
    - patch
    - update
- apiGroups:
    # The cluster operator needs the extensions api as the operator supports Kubernetes
version 1.11+
    # apps/v1 was introduced in Kubernetes 1.14
    - "extensions"
  resources:
    # The cluster operator needs to access and manage deployments to run deployment
based Strimzi components
    - deployments
    - deployments/scale
    # The cluster operator needs to access replica sets to manage Strimzi components and to
determine error states
    - replicasets
    # The cluster operator needs to access and manage replication controllers to manage
replicasets
    - replicationcontrollers
    # The cluster operator needs to access and manage network policies to lock down
communication between Strimzi components
    - networkpolicies
    # The cluster operator needs to access and manage ingresses which allow external
access to the services in a cluster
    - ingresses
  verbs:
    - get
    - list
    - watch
    - create
    - delete
    - patch
    - update
- apiGroups:
    - "apps"
  resources:
    # The cluster operator needs to access and manage deployments to run deployment
based Strimzi components
    - deployments
    - deployments/scale
    - deployments/status
    # The cluster operator needs to access and manage stateful sets to run stateful sets based
Strimzi components
    - statefulsets
    # The cluster operator needs to access replica-sets to manage Strimzi components and to
determine error states
    - replicasets
  verbs:
    - get
```

```
    - list
    - watch
    - create
    - delete
    - patch
    - update
- apiGroups:
    - ""
  resources:
    # The cluster operator needs to be able to create events and delegate permissions to do so
    - events
  verbs:
    - create
- apiGroups:
    # OpenShift S2I requirements
    - apps.openshift.io
  resources:
    - deploymentconfigs
    - deploymentconfigs/scale
    - deploymentconfigs/status
    - deploymentconfigs/finalizers
  verbs:
    - get
    - list
    - watch
    - create
    - delete
    - patch
    - update
- apiGroups:
    # OpenShift S2I requirements
    - build.openshift.io
  resources:
    - buildconfigs
    - buildconfigs/instantiate
    - builds
  verbs:
    - get
    - list
    - watch
    - create
    - delete
    - patch
    - update
- apiGroups:
    # OpenShift S2I requirements
    - image.openshift.io
  resources:
    - imagestreams
    - imagestreams/status
  verbs:
    - get
    - list
    - watch
    - create
    - delete
```

```
      - patch
      - update
  - apiGroups:
      - networking.k8s.io
    resources:
      # The cluster operator needs to access and manage network policies to lock down
communication between Strimzi components
      - networkpolicies
      # The cluster operator needs to access and manage ingresses which allow external
access to the services in a cluster
      - ingresses
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
  - apiGroups:
      - route.openshift.io
    resources:
      # The cluster operator needs to access and manage routes to expose Strimzi components
for external access
      - routes
      - routes/custom-host
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
  - apiGroups:
      - policy
    resources:
      # The cluster operator needs to access and manage pod disruption budgets this limits the
number of concurrent disruptions
      # that a Strimzi component experiences, allowing for higher availability
      - poddisruptionbudgets
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
```

第二个包括集群范围资源所需的权限。

为 Cluster Operator 带有集群范围资源的 ClusterRole

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: strimzi-cluster-operator-global
  labels:
    app: strimzi
rules:
  - apiGroups:
      - "rbac.authorization.k8s.io"
    resources:
      # The cluster operator needs to create and manage cluster role bindings in the case of an
install where a user
      # has specified they want their cluster role bindings generated
      - clusterrolebindings
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
  - apiGroups:
      - storage.k8s.io
    resources:
      # The cluster operator requires "get" permissions to view storage class details
      # This is because only a persistent volume of a supported storage class type can be
resized
      - storageclasses
    verbs:
      - get
  - apiGroups:
      - ""
    resources:
      # The cluster operator requires "list" permissions to view all nodes in a cluster
      # The listing is used to determine the node addresses when NodePort access is
configured
      # These addresses are then exposed in the custom resource states
      - nodes
    verbs:
      - list
```

The strimzi-kafka-broker ClusterRole 代表 Kafka pod 中 init 容器所需的访问权限，用于机架功能。如 委派的特权 部分所述，Cluster Operator 还需要此角色才能委派此访问权限。

Cluster Operator 的 ClusterRole，允许它将 OpenShift 节点的访问权限委派给 Kafka 代理 pod

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: strimzi-kafka-broker
  labels:
    app: strimzi
rules:
  - apiGroups:
      - ""
    resources:
      # The Kafka Brokers require "get" permissions to view the node they are on
      # This information is used to generate a Rack ID that is used for High Availability
configurations
      - nodes
    verbs:
      - get
```

The strimzi-topic-operator ClusterRole 代表 Topic Operator 所需的访问权限。如 委派的特权 部分所述，Cluster Operator 还需要此角色才能委派此访问权限。

Cluster Operator 的 ClusterRole，允许它将对事件的访问委托给主题 Operator

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: strimzi-entity-operator
  labels:
    app: strimzi
rules:
  - apiGroups:
      - "kafka.strimzi.io"
    resources:
      # The entity operator runs the KafkaTopic assembly operator, which needs to access and
manage KafkaTopic resources
      - kafkatopics
      - kafkatopics/status
      # The entity operator runs the KafkaUser assembly operator, which needs to access and
manage KafkaUser resources
```

```
      - kafkausers
      - kafkausers/status
    verbs:
      - get
      - list
      - watch
      - create
      - patch
      - update
      - delete
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      # The entity operator needs to be able to create events
      - create
  - apiGroups:
      - ""
    resources:
      # The entity operator user-operator needs to access and manage secrets to store
generated credentials
      - secrets
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
```

The strimzi-kafka-client ClusterRole 代表组件所需的访问，这些客户端使用客户端机架感知能力。如 委派的特权 部分所述，Cluster Operator 还需要此角色才能委派此访问权限。

Cluster Operator 的 ClusterRole，允许它将对 OpenShift 节点的访问委托给基于 Kafka 客户端的 pod

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: strimzi-kafka-client
  labels:
    app: strimzi
rules:
  - apiGroups:
      - ""
    resources:
```

```
    # The Kafka clients (Connect, Mirror Maker, etc.) require "get" permissions to view the
node they are on
    # This information is used to generate a Rack ID (client.rack option) that is used for
consuming from the closest
    # replicas when enabled
    - nodes
   verbs:
    - get
```

5.1.2.4. ClusterRoleBindings

对于包含集群范围资源的 ClusterRole，Operator 需要 Cluster RoleBindings 和RoleBindings 将其 ClusterRole 与其 ServiceAccount: ClusterRoleBinding s 相关联。

Cluster Operator 的 ClusterRoleBinding 示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: strimzi-cluster-operator
  labels:
    app: strimzi
subjects:
  - kind: ServiceAccount
    name: strimzi-cluster-operator
    namespace: myproject
roleRef:
  kind: ClusterRole
  name: strimzi-cluster-operator-global
  apiGroup: rbac.authorization.k8s.io
```

委派 所需的 ClusterRole 还需要 ClusterRoleBinding ：

Kafka 代理的 Cluster Operator 的 Cluster RoleBinding 示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
```

```
  name: strimzi-cluster-operator-kafka-broker-delegation
  labels:
    app: strimzi
# The Kafka broker cluster role must be bound to the cluster operator service account so that
it can delegate the cluster role to the Kafka brokers.
# This must be done to avoid escalating privileges which would be blocked by Kubernetes.
subjects:
  - kind: ServiceAccount
    name: strimzi-cluster-operator
    namespace: myproject
roleRef:
  kind: ClusterRole
  name: strimzi-kafka-broker
  apiGroup: rbac.authorization.k8s.io
```

和

用于 Kafka 客户端识别的 Cluster Operator 的 Cluster RoleBinding 示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: strimzi-cluster-operator-kafka-client-delegation
  labels:
    app: strimzi
# The Kafka clients cluster role must be bound to the cluster operator service account so that
it can delegate the
# cluster role to the Kafka clients using it for consuming from closest replica.
# This must be done to avoid escalating privileges which would be blocked by Kubernetes.
subjects:
  - kind: ServiceAccount
    name: strimzi-cluster-operator
    namespace: myproject
roleRef:
  kind: ClusterRole
  name: strimzi-kafka-client
  apiGroup: rbac.authorization.k8s.io
```

仅 包含有命名空间资源的 ClusterRole 只会使用 RoleBindings 绑定。

```
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: RoleBinding
metadata:
  name: strimzi-cluster-operator
  labels:
    app: strimzi
subjects:
  - kind: ServiceAccount
    name: strimzi-cluster-operator
    namespace: myproject
roleRef:
  kind: ClusterRole
  name: strimzi-cluster-operator-namespaced
  apiGroup: rbac.authorization.k8s.io
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: strimzi-cluster-operator-entity-operator-delegation
  labels:
    app: strimzi
# The Entity Operator cluster role must be bound to the cluster operator service account so
that it can delegate the cluster role to the Entity Operator.
# This must be done to avoid escalating privileges which would be blocked by Kubernetes.
subjects:
  - kind: ServiceAccount
    name: strimzi-cluster-operator
    namespace: myproject
roleRef:
  kind: ClusterRole
  name: strimzi-entity-operator
  apiGroup: rbac.authorization.k8s.io
```

5.1.3. 使用默认代理设置配置 Cluster Operator

如果您在 HTTP 代理上运行 Kafka 集群，您仍然可以传递数据。例如，您可以使用从代理外部推送和拉取数据的连接器运行 Kafka Connect。或者，您可以使用代理来连接授权服务器。

配置 Cluster Operator 部署以指定代理环境变量。Cluster Operator 接受标准代理配置（HTTP_PROXY、HTTPS_PROXY 和 NO_PROXY）作为环境变量。代理设置应用到所有 AMQ Streams 容器。

代理地址的格式为 http://IP-ADDRESS:PORT-NUMBER。要使用名称和密码设置代理，其格式为 http://USERNAME:PASSWORD@IP-ADDRESS:PORT-NUMBER。

先决条件

此流程需要使用 OpenShift 用户帐户，该帐户可以创建自定义 ResourceDefinition、 ClusterRole 和 ClusterRoleBinding。在 OpenShift 集群中使用 Role Base Access Control(RBAC)通常意味着只有

system:admin 等 OpenShift 集群管理员才有创建、编辑和删除这些资源的权限。

流程

1. *要向 Cluster Operator 添加代理环境变量，更新其 部署配置 (install/cluster-operator/060-Deployment-strimzi-cluster-operator.yaml)。*

   Cluster Operator 的代理配置示例

```
apiVersion: apps/v1
kind: Deployment
spec:
  # ...
  template:
    spec:
      serviceAccountName: strimzi-cluster-operator
      containers:
        # ...
        env:
        # ...
        - name: "HTTP_PROXY"
          value: "http://proxy.com" 1
        - name: "HTTPS_PROXY"
          value: "https://proxy.com" 2
        - name: "NO_PROXY"
          value: "internal.com, other.domain.com" 3
  # ...
```

1. *代理服务器的地址。*
2. *代理服务器的安全地址.*
3. *直接访问的服务器的地址，作为代理服务器的例外。URL 用逗号分开。*

或者，直接编辑 Deployment ：

```
oc edit deployment strimzi-cluster-operator
```

2. *如果您更新了 YAML 文件而不是直接编辑 Deployment，请应用更改：*

```
oc create -f install/cluster-operator/060-Deployment-strimzi-cluster-operator.yaml
```

其它资源

- *主机别名*
- *指定 AMQ Streams 管理员*

5.2. 使用主题 OPERATOR

当您使用 KafkaTopic 资源创建、修改或删除主题时，Tpic Operator 可确保这些更改反映在 Kafka 集群中。

OpenShift 指南中的部署和升级 AMQ Streams 提供了部署 Topic Operator 的说明：

- *使用 Cluster Operator（推荐）*
- *使用不由 AMQ Streams 管理的 Kafka 集群独立运行*

5.2.1. Kafka 主题资源

KafkaTopic 资源用于配置主题，包括分区和副本的数量。

KafkaTopic 的完整 schema 包括在 KafkaTopic schema 引用 中。

5.2.1.1. 为主题处理识别 Kafka 集群

KafkaTopic 资源包含一个标签，用于定义它所属的 Kafka 集群名称（从 Kafka 资源的名称衍生）。

例如：

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

该标签供主题 Operator 用于标识 KafkaTopic 资源、创建一个新主题，以及后续处理该主题。

如果标签与 Kafka 集群不匹配，主题 Operator 无法识别 KafkaTopic，且不会创建该主题。

5.2.1.2. Kafka 主题使用建议

处理主题时，应保持一致。始终在 OpenShift 中直接对 KafkaTopic 资源或主题运行。对于给定主题，避免在这两种方法之间进行定期切换。

使用反映主题性质的主题名称，并记住以后无法更改名称。

如果在 Kafka 中创建主题，请使用有效的 OpenShift 资源名称，否则 Topic Operator 将需要使用符合 OpenShift 规则的名称创建对应的 KafkaTopic。

注意

OpenShift 中的标识符和名称建议在 OpenShift 社区文章中概述了标识符和名称。

5.2.1.3. Kafka 主题命名约定

Kafka 和 OpenShift 分别为 Kafka 和 KafkaTopic.metadata.name 中的主题命名实施自己的验证规则。每个名称都无效。

使用 spec.topicName 属性，可以在 Kafka 中创建有效的主题，其名称对 OpenShift 中的 Kafka 主题无效。

spec.topicName 属性继承 Kafka 命名验证规则：

- *名称不能超过 249 个字符。*
- *Kafka 主题的有效字符为 ASCII 字母数字、.、_ 和 -。*
- *该名称不能为. 或 . 。 但是， 可以在 名称中使用，如 exampleTopic. or .exampleTopic。*

不得更改 spec.topicName。

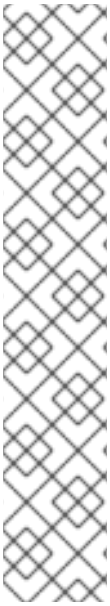
例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: topic-name-1
spec:
  topicName: topicName-1 (1)
  # ...
```

(1) *在 OpenShift 中，大写无效。*

不能改为：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: topic-name-1
spec:
  topicName: name-2
  # ...
```



注意

一些 Kafka 客户端应用程序（如 Kafka Streams）可以以编程方式在 Kafka 中创建主题。如果这些主题的名称具有无效的 OpenShift 资源名称，Topic Operator 会根据 Kafka 名称为其提供有效的 metadata.name。将替换无效字符，并在名称中附加一个哈希值。例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: mytopic---c55e57fe2546a33f9e603caf57165db4072e827e
spec:
  topicName: myTopic
  # ...
```

5.2.2. 主题 Operator 主题存储

Topic Operator 使用 Kafka 将主题元数据描述为键值对。主题存储 基于 Kafka Streams 键值机制，机制使用 Kafka 主题来持久保留状态。

主题元数据缓存在内存中，并在 Topic Operator 中本地访问。从应用到本地内存中缓存的操作的更新会保留到磁盘上的备份主题存储中。主题存储将与来自 Kafka 主题或 OpenShift KafkaTopic 自定义资源的更新持续同步。以这种方式设置主题存储会快速处理操作，但如果内存缓存崩溃，它会自动从持久性存储中重新填充。

5.2.2.1. 内部主题存储主题

内部主题支持处理主题元数据。

__strimzi_store_topic

存储主题元数据的输入主题

__strimzi-topic-operator-kstreams-topic-store-changelog

保留压缩主题存储值的日志

警告

不要删除这些主题，因为它们是运行 Topic Operator 的关键。

5.2.2.2. 从 ZooKeeper 迁移主题元数据

在之前的 AMQ Streams 发行版中，主题元数据存储在 ZooKeeper 中。新进程删除了这个要求，将元数据放入 Kafka 集群，并在 Topic Operator 控制下。

当升级到 AMQ Streams 1.8 时，对主题存储的 Topic Operator 控制会无缝转换。元数据可以从 ZooKeeper 找到并迁移，旧存储也会被删除。

5.2.2.3. 降级到使用 ZooKeeper 存储主题元数据的 AMQ Streams 版本

如果您要恢复到早于 0.22 的 AMQ Streams 版本（使用 ZooKeeper 存储主题元数据），您仍然会将 Cluster Operator 降级到以前的版本，然后将 Kafka 代理和客户端应用程序降级到以前的 Kafka 版本。

但是，您还必须使用 kafka-admin 命令删除为主题存储创建的主题，并指定 Kafka 集群的 bootstrap 地址。例如：

```
oc run kafka-admin -ti --image=registry.redhat.io/amq7/amq-streams-kafka-28-rhel7:1.8.0 --rm=true --restart=Never -- ./bin/kafka-topics.sh --bootstrap-server localhost:9092 --topic __strimzi-topic-operator-kstreams-topic-store-changelog --delete && ./bin/kafka-topics.sh --bootstrap-server localhost:9092 --topic __strimzi_store_topic --delete
```

命令必须与用于访问 Kafka 集群的监听程序和身份验证对应。

Topic Operator 将从 Kafka 中主题的状态重建 ZooKeeper 主题元数据。

5.2.2.4. 主题 Operator 主题复制和扩展

推荐由 Topic Operator 管理的主题配置是主题复制因素 3，至少为 2 个以同步副本。

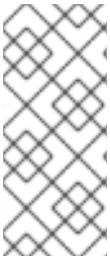
```
apiVersion: kafka.strimzi.io/v1beta2
```

```
kind: KafkaTopic
metadata:
  name: my-topic
  labels:
    strimzi.io/cluster: my-cluster
spec:
  partitions: 1 (1)
  replicas: 3 (2)
  config:
    min.insync.replicas=2 (3)
  #...
```

(1) 主题的分区数量。通常情况下，1 分区就足够了。

(2) 副本主题分区的数量。目前，这无法在 KafkaTopic 资源中 更改，但可以使用 kafka-reassign-partitions.sh 工具进行修改。

(3) 消息必须成功写入或引发异常的最小副本分区数量。



注意

同步内副本与制作者应用的 ack 配置 结合使用。acks 配置决定消息必须复制到的追随者分区的数量，然后确认消息被确认为成功接收。Topic Operator 使用 acks=all 运行，在此情况下，所有内同步副本都必须确认消息。

当通过添加或删除代理来扩展 Kafka 集群时，不会更改复制因素配置，并且不会自动重新分配副本。但是，您可以使用 kafka-reassign-partitions.sh 工具更改复制因素，并手动将副本分配给代理。

另外，尽管 AMQ Streams 的 Cruise Control 集成无法更改主题的复制因素，但它为重新平衡 Kafka 而生成的优化结果包括传输分区副本并更改分区领导地位的命令。

5.2.2.5. 处理主题的更改

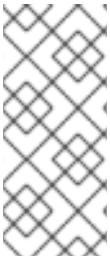
Topic Operator 需要解决的一个根本问题是没有单一的事实来源： KafkaTopic 资源和 Kafka 主题都可以独立于主题 Operator 进行修改。复杂的情况是，Topic Operator 可能无法实时观察每个末尾的更改。例如，当主题 Operator 停机时。

为解决这个问题，主题 Operator 会维护主题存储中每个主题的信息。当 Kafka 集群或 OpenShift 中发生更改时，它会同时查看其他系统和主题存储的状态，以确定需要更改什么内容才能保持所有同步。只要 Topic Operator 启动并在运行期间定期进行同样的操作。

例如，假设 Topic Operator 没有运行，并创建一个名为 my-topic 的 KafkaTopic。当主题 Operator 启动时，主题存储不包含有关 my-topic 的信息，因此它可以推断 KafkaTopic 是在上次运行后创建的。Topic Operator 会创建与 my-topic 对应的主题，并将 my-topic 的元数据存储在主题存储中。

如果您更新 Kafka 主题配置或通过 KafkaTopic 自定义资源应用更改，则在 Kafka 集群协调后会更新主题存储。

主题存储还允许主题 Operator 管理在 Kafka 主题中更改主题配置并通过 OpenShift KafkaTopic 自定义资源 进行更新 的场景，只要更改不兼容。例如，可以对同一主题配置键进行更改，但更改不同的值。对于不兼容的更改，Kafka 配置将具有优先权，并且会相应地更新 KafkaTopic。



注意

您还可以通过 oc delete -f KAFKA-TOPIC-CONFIG-FILE 命令使用 KafkaTopic 资源删除主题。要做到这一点，在 Kafka 资源的 spec.kafka.config 中必须将 delete.topic.enable 设置为 true （默认）。

其它资源

- 降级 AMQ Streams
- 生产者配置调整和数据持久性
- 扩展集群和分区分配
- 用于集群重新平衡的精简控制

5.2.3. 配置 Kafka 主题

使用 KafkaTopic 资源的属性来配置 Kafka 主题。

您可以使用 oc apply 创建或修改主题，使用 oc delete 删除现有主题。

例如：

- *oc apply -f <topic-config-file>*
- *oc delete KafkaTopic <topic-name>*

此流程演示了如何创建带有 10 个分区和 2 个副本的主题。

开始前

在进行更改前请考虑以下几点：

- *Kafka 不支持 通过 KafkaTopic 资源进行以下更改：*
  - *使用 spec.topicName更改主题名称*
  - *使用 spec.partitions减少分区大小*
- *您不能使用 spec.replicas 来更改最初指定的副本数量。*
- *使用键为主题增加 spec.partitions 将更改记录的分区方式，这在主题使用 语义分区 时特别有问题。*

先决条件

- *正在运行的 Kafka 集群使用 Kafka 代理监听程序配置使用 TLS 身份验证和加密。*
- *一个正在运行的主题 Operator（通常使用 Entity Operator 部署）。*

- *要删除主题，请在 Kafka 资源的 spec.kafka.config 中 delete.topic. enable=true (default)。*

流程

1. *准备包含要创建的 KafkaTopic 的文件。*

   KafkaTopic 示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: orders
  labels:
    strimzi.io/cluster: my-cluster
spec:
  partitions: 10
  replicas: 2
```

   提示

   在修改主题时，您可以使用 oc get kafkatopic Order -o yaml 获取资源的当前版本。

2. *在 OpenShift 中创建 KafkaTopic 资源。*

```
oc apply -f TOPIC-CONFIG-FILE
```

5.2.4. 使用资源请求和限值配置主题 Operator

您可以将资源（如 CPU 和内存）分配给 Topic Operator，并为它消耗的资源量设置限制。

先决条件

- *Cluster Operator 正在运行。*

流程

1. *根据需要更新编辑器中的 Kafka 集群配置：*

```
oc edit kafka MY-CLUSTER
```

2. *在 Kafka 资源中的 spec.entityOperator.topicOperator.resources 属性中，为 Topic Operator 设置资源请求和限值。*

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
 # Kafka and ZooKeeper sections...
 entityOperator:
   topicOperator:
     resources:
       requests:
         cpu: "1"
         memory: 500Mi
       limits:
         cpu: "1"
         memory: 500Mi
```

3. *应用新配置以创建或更新资源。*

```
oc apply -f KAFKA-CONFIG-FILE
```

5.3. 使用 USER OPERATOR

当使用 KafkaUser 资源创建、修改或删除用户时，User Operator 可确保这些更改反映在 Kafka 集群中。

OpenShift 指南中的部署和升级 AMQ Streams 提供了部署 User Operator 的说明：

- *使用 Cluster Operator（推荐）*
- *使用不由 AMQ Streams 管理的 Kafka 集群独立运行*

有关模式的更多信息，请参阅 KafkaUser schema 参考。

验证和授权对 Kafka 的访问

使用 KafkaUser 启用特定客户端用来访问 Kafka 的身份验证和授权机制。

有关使用 KafkUser 管理用户并保护对 Kafka 代理的访问的更多信息，请参阅保护对 Kafka 代理的访问。

5.3.1. 使用资源请求和限值配置 User Operator

您可以将资源（如 CPU 和内存）分配给 User Operator，并为它消耗的资源量设置限制。

先决条件

- Cluster Operator 正在运行。

流程

1. 根据需要更新编辑器中的 Kafka 集群配置：

```
oc edit kafka MY-CLUSTER
```

2. 在 Kafka 资源中的 spec.entityOperator.userOperator.resources 属性中，为 User Operator 设置资源请求和限值。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # Kafka and ZooKeeper sections...
  entityOperator:
    userOperator:
      resources:
        requests:
          cpu: "1"
          memory: 500Mi
        limits:
          cpu: "1"
          memory: 500Mi
```

保存文件并退出编辑器。Cluster Operator 会自动应用更改。

5.4. 使用 PROMETHEUS 指标监控 OPERATOR

AMQ Streams 操作器公开 Prometheus 指标数据。指标数据会被自动启用，其中包含以下相关信息：

- 协调数
- Operator 正在处理的自定义资源数量
- 协调的持续时间
- 来自操作器的 JVM 指标

另外，我们提供了一个 Grafana 仪表板示例。

如需有关 Prometheus 的更多信息，请参阅 OpenShift 指南中的 Deploying and upgrade AMQ Streams 中的 引入 Metrics to Kafka。

第 6 章 KAFKA BRIDGE

本章概述了 AMQ Streams Kafka Bridge，并帮助您开始使用 REST API 与 AMQ Streams 交互。

- 要在您的本地环境中尝试 Kafka 网桥，请参阅本章后面的 第 6.2 节 “Kafka Bridge quickstart”。
- 有关配置步骤的详情请参考 第 2.5 节 “Kafka Bridge 集群配置”。
- 要查看 API 文档，请参阅 Kafka Bridge API 参考。

6.1. KAFKA 网桥概述

您可以使用 AMQ Streams Kafka Bridge 作为接口，向 Kafka 集群发出特定类型的 HTTP 请求。

6.1.1. Kafka Bridge 接口

Kafka Bridge 提供了一个 RESTful 接口，允许基于 HTTP 的客户端与 Kafka 集群交互。 它提供了与 AMQ Streams 的 Web API 连接的优势，不需要客户端应用程序来解释 Kafka 协议。

API 有两个主要资源（ 使用者和 主题 ），它们通过端点公开并可访问，以便与 Kafka 集群中的用户和生产者交互。资源仅与 Kafka 网桥相关，而不是与 Kafka 直接连接的消费者和生产者。

6.1.1.1. HTTP 请求

Kafka Bridge 支持对 Kafka 集群的 HTTP 请求，其方法如下：

- 发送消息到一个主题。
- 从主题检索消息.
- 检索主题的分区列表。

- 创建和删除消费者.
- 订阅消费者了解主题，以便他们开始接收来自这些主题的信息。
- 检索消费者订阅的主题列表。
- 取消订阅消费者的主题.
- 将分区分配给消费者.
- 提交消费者偏移列表。
- 寻找分区，以便使用者开始接受来自第一个或最后一个偏移位置的信息，或者给定的偏移位置。

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

客户端可以生成和使用消息，而无需使用原生 Kafka 协议。

其它资源

- 要查看 API 文档，包括请求和响应示例，请参阅 Kafka Bridge API 参考。

6.1.2. 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。

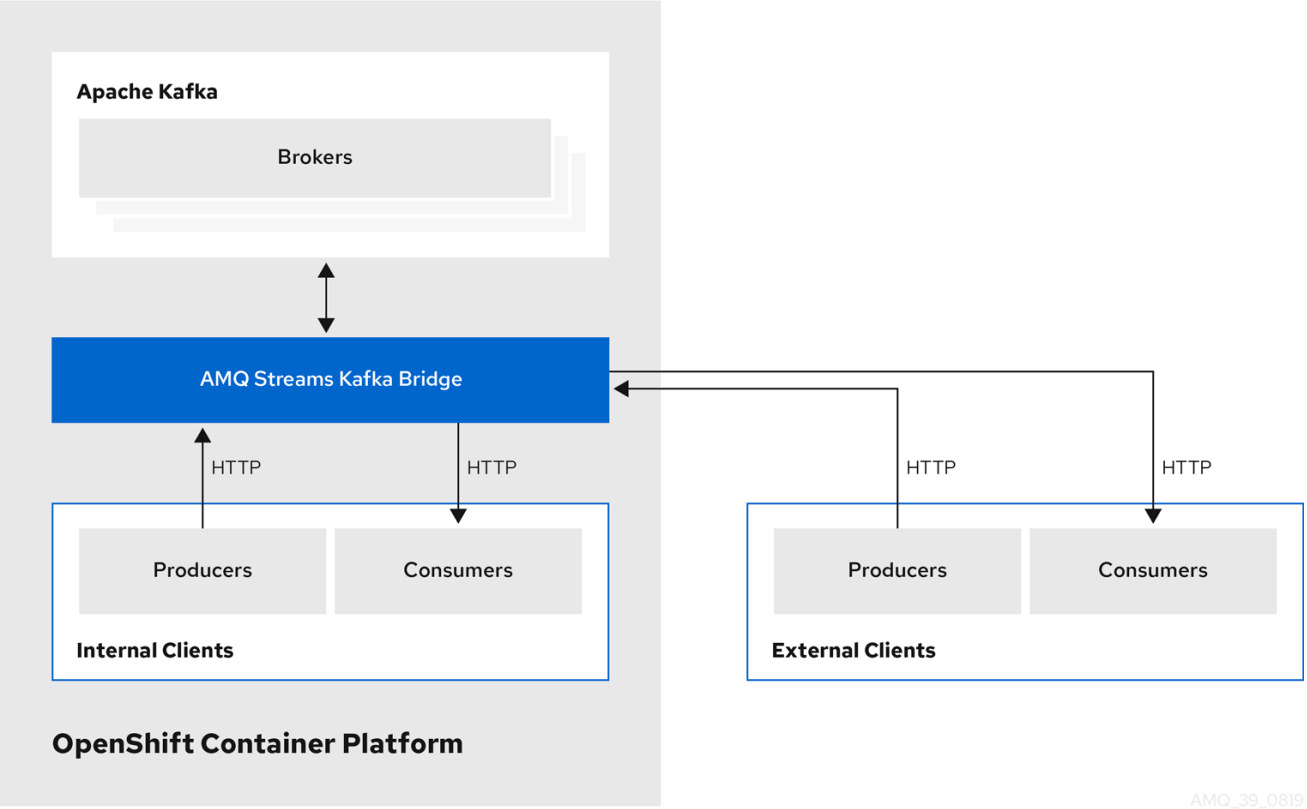
内部客户端

内部客户端是基于容器的 HTTP 客户端，与 Kafka Bridge 本身 在同一个 OpenShift 集群中运行。内部客户端可以访问在 Kafka Bridge 自定义资源中定义的主机上的 Kafka Bridge 和端口。

外部客户端

外部客户端是在 OpenShift 集群 外部 运行的 HTTP 客户端，其中部署并运行 Kafka Bridge。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



6.1.3. 保护 Kafka 网桥

AMQ Streams 目前不为 Kafka Bridge 提供任何加密、身份验证或授权。这意味着，从外部客户端发送到 Kafka Bridge 的请求是：

- 未加密，且必须使用 HTTP 而不是 HTTPS
- 在没有验证的情况下发送

但是，您可以使用其他方法保护 Kafka 网桥的安全，例如：

- 定义哪些容器集可以访问 Kafka 网桥的 OpenShift Network 策略。
- 带有身份验证或授权的反向代理，如 OAuth2 代理。
- API 网关.
- 具有 TLS 终止的入口或 OpenShift 路由。

当连接到 Kafka Broker 时，Kafka Bridge 支持 TLS 加密以及 TLS 和 SASL 身份验证。在 OpenShift 集群中，您可以配置：

- Kafka Bridge 和 Kafka 集群间的 TLS 或基于 SASL 的身份验证
- Kafka 网桥和 Kafka 集群之间的 TLS 加密连接。

如需更多信息，请参阅 第 2.5.1 节 “配置 Kafka 网桥”。

您可以使用 Kafka 代理中的 ACL 来限制可通过 Kafka Bridge 使用和生成的主题。

6.1.4. 访问 OpenShift 外部的 Kafka 网桥

部署后，AMQ Streams Kafka Bridge 只能由同一 OpenShift 集群中运行的应用程序访问。这些应用程序使用 kafka-bridge-name-bridge-service 服务来访问 API。

如果要使 Kafka Bridge 可供在 OpenShift 集群外运行的应用程序访问，您可以使用以下功能之一手动公开它：

- LoadBalancer 或 NodePort 类型的服务
- Ingress 资源

- *OpenShift 路由*

如果您决定创建服务，请使用 选择器 中的以下标签来配置服务要将流量路由到的 pod:

```
# ...
selector:
  strimzi.io/cluster: kafka-bridge-name 1
  strimzi.io/kind: KafkaBridge
#...
```

1 *OpenShift 集群中 Kafka Bridge 自定义资源的名称。*

6.1.5. 对 Kafka Bridge 的请求

指定数据格式和 HTTP 标头，以确保向 Kafka Bridge 提交有效的请求。

6.1.5.1. 内容类型标头

API 请求和响应正文始终编码为 JSON。

- *在执行消费者操作时，如果存在非空正文，POST 请求必须提供以下 Content-Type 标头：*

```
Content-Type: application/vnd.kafka.v2+json
```

- *在执行制作者操作时，POST 请求 必须提供 Content-Type 标头，指定生成的消息的 嵌入式数据格式。这可以是 json 或 二进制。*

嵌入式数据格式	content-Type 标头
JSON	**content-Type: application/vnd.kafka.json.v2+json**
二进制	**content-Type: application/vnd.kafka.binary.v2+json**

嵌入式数据格式为每个消费者设置，如下一节所述。

如果 POST 请求具有空正文，则不能设置 Content-Type。空正文可用于创建具有默认值的消费者。

6.1.5.2. 嵌入式数据格式

嵌入式数据格式是 Kafka 消息通过 HTTP 从生产者传输到使用 Kafka 网桥的消费者的格式。支持两种嵌入的数据格式：JSON 和二进制。

在使用 /consumers/groupid 端点创建消费者时，POST 请求正文必须指定嵌入的数据格式（JSON 或二进制）。这在 format 字段中指定，例如：

```
{
  "name": "my-consumer",
  "format": "binary", 1
...
}
```

1

二进制嵌入式数据格式.

创建使用者时指定的嵌入式数据格式必须与它将使用的 Kafka 消息的数据格式匹配。

如果您选择指定二进制嵌入式数据格式，后续制作者请求必须在请求正文中以 Base64 编码的字符串形式提供二进制数据。例如，在使用 /topics/topicname 端点发送信息时，records.value 必须采用 Base64 编码：

```
{
  "records": [
    {
      "key": "my-key",
      "value": "ZWR3YXJkdGhldGhyZWVsZWdnZWRjYXQ="
    },
  ]
}
```

制作者请求还必须提供与嵌入式数据格式对应的 Content-Type 标头，如 Content-Type: application/vnd.kafka.binary.v2+json。

6.1.5.3. 消息格式

使用 /topics 端点发送消息时，您将在请求正文中的 record 参数中输入消息有效负载。

records 参数可以包含任何这些可选字段：

- 消息标头
- 消息键
- 消息值
- 目标分区

到 /topics 的 POST 请求示例

```
curl -X POST \
  http://localhost:8080/topics/my-topic \
  -H 'content-type: application/vnd.kafka.json.v2+json' \
  -d '{
    "records": [
       {
          "key": "my-key",
          "value": "sales-lead-0001"
          "partition": 2
          "headers": [
            {
              "key": "key1",
              "value": "QXBhY2hlIEthZmthIGlzIHRoZSBib21iIQ==" 1
            }
          ]
       },
    ]
}'
```

1

二进制格式的标头值，编码为 Base64。

6.1.5.4. 接受标头

创建消费者后，所有后续 GET 请求都必须以以下格式提供 Accept 标头：

```
Accept: application/vnd.kafka.EMBEDDED-DATA-FORMAT.v2+json
```

EMBEDDED-DATA-FORMAT 是 json 或 二进制。

例如，当使用嵌入的 JSON 数据格式获取订阅的消费者的记录时，包括这个 Accept 标头：

```
Accept: application/vnd.kafka.json.v2+json
```

6.1.6. CORS

跨 Origin 资源共享(CORS)允许您指定在 Kafka 网桥 HTTP 配置 中访问 Kafka 集群的允许方法和原始 URL。

Kafka Bridge 的 CORS 配置示例

```
# ...
cors:
  allowedOrigins: "https://strimzi.io"
  allowedMethods: "GET,POST,PUT,DELETE,OPTIONS,PATCH"
  # ...
```

CORS 允许在不同域中的原始源之间 简单 且 预先理解的请求。

简单请求适用于使用 GET、HEAD、POST 方法的标准请求。

预定义的请求会发送 HTTP OPTIONS 请求，作为检查实际请求是否安全发送的初始检查。确认后会发送实际请求。preflight 请求适用于需要更大保护的方法，如 PUT 和 DELETE，以及使用非标准标头。

所有请求都需要其标头中有一个 Origin 值，这是 HTTP 请求的来源。

6.1.6.1. 简单请求

例如：这个简单请求标头将原始文件指定为 https://strimzi.io。

```
Origin: https://strimzi.io
```

标头信息添加到请求中。

```
curl -v -X GET HTTP-ADDRESS/bridge-consumer/records \
-H 'Origin: https://strimzi.io'\
-H 'content-type: application/vnd.kafka.v2+json'
```

在 Kafka Bridge 的响应中，会返回 Access-Control-Allow-Origin 标头。

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: * (1)
```

(1) 返回星号(*)表示资源可以被任何域访问。

6.1.6.2. Preflighted 请求

使用 a OPTIONS 方法将初始 preflight 请求发送到 Kafka Bridge。HTTP OPTIONS 请求发送标头信息，以检查 Kafka Bridge 是否允许实际请求。

这里的 preflight 请求检查 POST 请求 是否从 https://strimzi.io 有效。

```
OPTIONS /my-group/instances/my-user/subscription HTTP/1.1
Origin: https://strimzi.io
Access-Control-Request-Method: POST (1)
Access-Control-Request-Headers: Content-Type (2)
```

(1) Kafka Bridge 已被警告，实际请求是 POST 请求。

2

实际的请求将与 Content-Type 标头一起发送。

OPTIONS 被添加到 preflight 请求的标题信息中。

```
curl -v -X OPTIONS -H 'Origin: https://strimzi.io' \
-H 'Access-Control-Request-Method: POST' \
-H 'content-type: application/vnd.kafka.v2+json'
```

Kafka Bridge 响应初始请求，以确认请求被接受。响应标头返回允许的来源、方法和标头。

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://strimzi.io
Access-Control-Allow-Methods: GET,POST,PUT,DELETE,OPTIONS,PATCH
Access-Control-Allow-Headers: content-type
```

如果原始或方法被拒绝，则返回错误消息。

实际请求不需要 Access-Control-Request-Method 标头，因为它已在 preflight 请求中确认，但它确实需要 origin 标头。

```
curl -v -X POST HTTP-ADDRESS/topics/bridge-topic \
-H 'Origin: https://strimzi.io' \
-H 'content-type: application/vnd.kafka.v2+json'
```

响应中显示允许的来源 URL。

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://strimzi.io
```

其它资源

获取 CORS 规格

6.1.7. Kafka Bridge API 资源

有关 REST API 端点和描述的完整列表，包括请求和响应示例，请查看 Kafka Bridge API 参考。

6.1.8. Kafka Bridge 部署

您可以使用 Cluster Operator 将 Kafka Bridge 部署到 OpenShift 集群中。

部署 Kafka Bridge 后，Cluster Operator 会在 OpenShift 集群中创建 Kafka Bridge 对象。对象包括 部署、服务和 pod，各自名称都以 Kafka Bridge 自定义资源中指定的名称命名。

其它资源

- 有关部署说明，请参阅 OpenShift 指南中的 Deploying AMQ Streams 中的将 Kafka Bridge 部署到 OpenShift 集群。
- 有关配置 Kafka 网桥的详情，请参考 第 2.5 节 “Kafka Bridge 集群配置”
- 有关为 KafkaBridge 资源配置主机和端口的详情请参考 第 2.5.1 节 “配置 Kafka 网桥”。
- 有关集成外部客户端的详情请参考 第 6.1.4 节 “访问 OpenShift 外部的 Kafka 网桥”。

6.2. KAFKA BRIDGE QUICKSTART

使用此快速入门尝试本地开发环境中的 AMQ Streams Kafka Bridge。您将学习如何：

- 将 Kafka Bridge 部署到 OpenShift 集群
- 使用端口转发向本地机器公开 Kafka Bridge 服务
- 生成到 Kafka 集群中主题和分区的信息
- 创建 Kafka 网桥消费者
- 执行基本的消费者操作，如将消费者订阅到主题并检索您生成的信息

在这个快速启动中，HTTP 请求格式化为 curl 命令，您可以将它们复制并粘贴到您的终端。需要访问 OpenShift 集群。

确保您具有先决条件，然后按照本章中提供的顺序按照任务进行操作。

关于数据格式

在此快速入门中，您将以 JSON 格式（而非二进制）生成和使用消息。有关示例请求中使用的数据格式和 HTTP 标头的更多信息，请参阅 第 6.1.5 节 “对 Kafka Bridge 的请求”。

快速启动的先决条件

- 集群管理员对本地或远程 OpenShift 集群的访问权限。
- 已安装 AMQ Streams。
- 一个正在运行的 Kafka 集群，由 Cluster Operator 在 OpenShift 命名空间中部署。
- Entity Operator 作为 Kafka 集群的一部分被部署并运行。

6.2.1. 将 Kafka Bridge 部署到 OpenShift 集群

AMQ Streams 包含一个 YAML 示例，用于指定 AMQ Streams Kafka Bridge 的配置。对此文件进行一些最小更改，然后将 Kafka Bridge 的实例部署到 OpenShift 集群。

流程

1. 编辑 example /bridge/kafka-bridge.yaml 文件。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: quickstart (1)
spec:
  replicas: 1
```

```
bootstrapServers: <cluster-name>-kafka-bootstrap:9092 (2)
http:
  port: 8080
```

(1) 部署 Kafka Bridge 后，-bridge 会被附加到部署的名称和其他相关资源中。在本例中，Kafka Bridge 部署名为 quickstart-bridge， 随附的 Kafka Bridge 服务名为 quickstart-bridge-service。

(2) 在 bootstrapServers 属性中，输入 Kafka 集群的名称作为 <cluster-name>。

2. 将 Kafka Bridge 部署到 OpenShift 集群：

```
oc apply -f examples/bridge/kafka-bridge.yaml
```

OpenShift 集群中会创建 Quickstart -bridge 部署、服务和其他相关资源。

3. 验证 Kafka 网桥是否已成功部署：

```
oc get deployments
```

```
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
quickstart-bridge         1/1     1            1           34m
my-cluster-connect        1/1     1            1           24h
my-cluster-entity-operator  1/1   1            1           24h
#...
```

接下来要做什么

将 Kafka Bridge 部署到 OpenShift 集群后，将 Kafka Bridge 服务公开给您的本地机器。

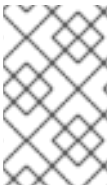
其它资源

- 有关配置 Kafka 网桥的详情请参考 第 2.5 节 “Kafka Bridge 集群配置”。

6.2.2. 将 Kafka Bridge 服务公开到您的本地机器

接下来，使用端口转发将 AMQ Streams Kafka Bridge 服务公开给 http://localhost:8080 上的本地机

器。



注意

端口转发仅适用于开发和测试目的。

流程

1. 列出 OpenShift 集群中 pod 的名称：

```
oc get pods -o name

pod/kafka-consumer
# ...
pod/quickstart-bridge-589d78784d-9jcnr
pod/strimzi-cluster-operator-76bcf9bc76-8dnfm
```

2. 连接到端口 8080 上的 quickstart-bridge pod:

```
oc port-forward pod/quickstart-bridge-589d78784d-9jcnr 8080:8080 &
```

注意

如果本地计算机上的端口 8080 已在使用，请使用其它 HTTP 端口，如 8008。

API 请求现在从本地机器的端口 8080 转发到 Kafka Bridge Pod 中的端口 8080。

6.2.3. 生成到主题和分区的消息

接下来，使用主题端点以 JSON 格式生成消息到 主题。您可以在请求正文中为消息指定目的地分区，如下所示。分区 端点 提供了一种备选方法，用于指定所有消息的单一目标分区，作为路径参数。

流程

1. 在文本编辑器中，为带有三个分区的 Kafka 主题创建 YAML 定义。

```
apiVersion: kafka.strimzi.io/v1beta2
```

```
kind: KafkaTopic
metadata:
  name: bridge-quickstart-topic
  labels:
    strimzi.io/cluster: <kafka-cluster-name> 1
spec:
  partitions: 3 2
  replicas: 1
  config:
    retention.ms: 7200000
    segment.bytes: 1073741824
```

1 部署 Kafka Bridge 的 Kafka 集群的名称。

2 主题的分区数量。

2. 将文件保存到 example /topic 目录中，作为 bridge-quickstart-topic.yaml。

3. 在 OpenShift 集群中创建主题：

```
oc apply -f examples/topic/bridge-quickstart-topic.yaml
```

4. 使用 Kafka Bridge，为您创建的主题生成三个信息：

```
curl -X POST \
  http://localhost:8080/topics/bridge-quickstart-topic \
  -H 'content-type: application/vnd.kafka.json.v2+json' \
  -d '{
    "records": [
        {
            "key": "my-key",
            "value": "sales-lead-0001"
        },
        {
            "value": "sales-lead-0002",
            "partition": 2
        },
        {
            "value": "sales-lead-0003"
        }
    ]
}'
```

- *sales-lead-0001 发送至基于密钥哈希的分区。*
- *sales-lead-0002 直接发送到分区 2。*
- *sales-lead-0003 通过循环方法发送到 bridge-quickstart-topic 主题中的分区。*

5. *如果请求成功，Kafka Bridge 将返回一个 偏移数组，以及 200 代码 和内容类型 标头 application/vnd.kafka.v2+json。对于每条消息，偏移 阵列描述：*

- *消息发送到的分区*
- *分区的当前消息偏移*

响应示例

```
#...
{
  "offsets":[
    {
      "partition":0,
      "offset":0
    },
    {
      "partition":2,
      "offset":0
    },
    {
      "partition":0,
      "offset":1
    }
  ]
}
```

接下来要做什么

在向主题和分区生成消息后，创建一个 Kafka 网桥使用者。

其它资源

- API 参考文档中的 POST /topics/{topicname}。
- API 参考文档中的 POST /topics/{topicname}/partitions/{partitionid}。

6.2.4. 创建 Kafka 网桥消费者

在 Kafka 集群中执行任何消费者操作前，您必须首先使用使用者端点创建 消费者。用户称为 Kafka 网桥消费者。

流程

1. 在名为 bridge-quickstart-consumer-group 的新使用者组中创建一个 Kafka 网桥使用者 ：

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group \
  -H 'content-type: application/vnd.kafka.v2+json' \
  -d '{
    "name": "bridge-quickstart-consumer",
    "auto.offset.reset": "earliest",
    "format": "json",
    "enable.auto.commit": false,
    "fetch.min.bytes": 512,
    "consumer.request.timeout.ms": 30000
  }'
```

   - 用户名为 bridge-quickstart-consumer，嵌入式数据格式则设为 json。
   - 定义了一些基本的配置设置：
   - 由于 enable.auto.commit 设置为 false，因此使用者不会自动向日志提交偏移。在此快速入门稍后您将手动提交偏移。

   如果请求成功，Kafka Bridge 会在响应正文返回使用者 ID(instance_id)和基本 URL(base_uri)，以及 200 代码。

   响应示例

```
#...
{
  "instance_id": "bridge-quickstart-consumer",
  "base_uri":"http://<bridge-name>-bridge-service:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer"
}
```

2. 复制基本 URL(base_uri)，以便在这个快速启动的其他消费者操作中使用。

接下来要做什么

现在，您已创建了 Kafka 网桥消费者，您可以为它订阅主题。

其它资源

- API 参考文档中的 POST/consumers/{groupid}。

6.2.5. 将 Kafka 网桥消费者订阅到主题

创建 Kafka 网桥消费者后，使用订阅端点将其 订阅 到一个或多个主题。订阅后，消费者开始收到生成到该主题的所有消息。

流程

- 在向主题和分区 生成信息时，将消费者订阅您之前创建的 bridge-quickstart-topic 主题：

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/subscription \
  -H 'content-type: application/vnd.kafka.v2+json' \
  -d '{
    "topics": [
        "bridge-quickstart-topic"
    ]
}'
```

主题 数组可以包含一个主题（如下所示）或多个主题。如果要将消费者订阅与正则表达式匹配的多个主题，您可以使用 topic_pattern 字符串而不是 主题 数组。

如果请求成功，Kafka Bridge 只会返回 204 (No Content)代码。

接下来要做什么

在将 Kafka 网桥使用者订阅到主题后，您可以从 消费者检索消息。

其它资源

- API 参考文档中的 POST /consumers/{groupid}/instances/{name}/subscription。

6.2.6. 从 Kafka Bridge 用户检索最新信息

接下来，通过从 记录 端点请求数据，从 Kafka 网桥使用者检索最新的消息。在生产环境中，HTTP 客户端可以重复调用此端点（在循环中）。

流程

1. 为 Kafka 网桥使用者生成其他信息，如 向主题和分区生成消息 中所述。

2. 将 GET 请求 提交到 记录 端点：

```
curl -X GET http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/records \
  -H 'accept: application/vnd.kafka.json.v2+json'
```

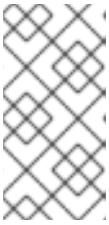
创建并订阅 Kafka 网桥消费者后，第一个 GET 请求将返回空响应，因为轮询操作会启动重新平衡过程来分配分区。

3. 重复步骤二，从 Kafka Bridge consumer 检索消息。

Kafka Bridge 返回一个消息的数组来代表响应正文中的主题名称、键、值、分区和偏移全部全部信息， 以及 200 个代码。消息默认从最新的偏移检索。

```
HTTP/1.1 200 OK
content-type: application/vnd.kafka.json.v2+json
#...
[
  {
```

```
  "topic":"bridge-quickstart-topic",
  "key":"my-key",
  "value":"sales-lead-0001",
  "partition":0,
  "offset":0
 },
 {
  "topic":"bridge-quickstart-topic",
  "key":null,
  "value":"sales-lead-0003",
  "partition":0,
  "offset":1
 },
#...
```



注意

如果返回空的响应，会按照 向主题和分区生成消息 中所述为消费者生成更多记录，然后尝试再次检索消息。

接下来要做什么

从 Kafka 网桥使用者检索消息后，尝试 向日志提交偏移。

其它资源

- API 参考文档中的 GET /consumers/{groupid}/instances/{name}/records。

6.2.7. 将偏移提交到日志

接下来，使用 偏移端点 将偏移手动提交到 Kafka Bridge使用者接收的所有消息的日志中。这是必要的，因为您在创建 Kafka Bridge 用户时，之前创建的 Kafka 网桥消费者 被配置为使用 enable.auto.commit 设置为 false。

流程

- 将偏移量提交到 bridge-quickstart-consumer 的日志：

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/offsets
```

由于没有提交请求正文，将为消费者收到的所有记录提交偏移。或者，请求正文可以包含一个数组(OffsetCommitSeekList)，用于指定您要提交偏移的主题和分区。

如果请求成功，Kafka Bridge 只会返回 204 代码。

接下来要做什么

向日志提交偏移后，尝试查找要 偏移的 端点。

其它资源

- API 参考文档中的 POST /consumers/{groupid}/instances/{name}/offsets。

6.2.8. 寻找分区的偏移

接下来，使用 位置 端点配置 Kafka 网桥使用者，以从特定偏移检索分区的消息，然后从最新的偏移中检索。这在 Apache Kafka 中称为搜索操作。

流程

1. 为 Quickstart -bridge-topic 主题的分区 0 寻找特定的偏移：

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/positions \
  -H 'content-type: application/vnd.kafka.v2+json' \
  -d '{
    "offsets": [
      {
        "topic": "bridge-quickstart-topic",
        "partition": 0,
        "offset": 2
      }
    ]
}'
```

如果请求成功，Kafka Bridge 只会返回 204 代码。

2. 将 GET 请求 提交到 记录 端点：

```
curl -X GET http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/records \
  -H 'accept: application/vnd.kafka.json.v2+json'
```

Kafka Bridge 从您要查找的偏移返回消息。

3. 通过查找同一分区的最后偏移来恢复默认消息检索行为。这一次使用 position /end 端点。

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-
group/instances/bridge-quickstart-consumer/positions/end \
  -H 'content-type: application/vnd.kafka.v2+json' \
  -d '{
    "partitions": [
      {
        "topic": "bridge-quickstart-topic",
        "partition": 0
      }
    ]
}'
```

如果请求成功，Kafka Bridge 会返回另一个 204 代码。

注意

您还可以使用 位置/探测端点 来查找一个或多个分区的第一个偏移。

接下来要做什么

在这个快速入门中，您已使用 AMQ Streams Kafka Bridge 在 Kafka 集群上执行几个常用操作。现在，您可以删除之前创建的 Kafka 网桥消费者。

其它资源

- API 参考文档中的 POST/consumers/{groupid}/instances/{name}/ 位置。
- POST /consumers/{groupid}/instances/{name}/locations/beginning in the API 参考文档中。
- 在 API 参考文档中 POST /consumers/{groupid}/instances/{name}/locations/end。

6.2.9. 删除 Kafka 网桥消费者

最后，删除整个快速入门中使用的 Kafa Bridge 消费者。

流程

- *通过向 实例 端点发送 DELETE 请求来删除 Kafka 网桥使用者。*

```
curl -X DELETE http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer
```

如果请求成功，Kafka Bridge 只会返回 204 代码。

其它资源

- *API 参考文档中的 DELETE /consumers/{groupid}/instances/{name}。*

第 7 章 使用带有 3SCALE 的 KAFKA 网桥

您可以将 Red Hat 3scale API 管理与 AMQ Streams Kafka Bridge 部署和集成。

7.1. 使用带有 3SCALE 的 KAFKA 网桥

使用 Kafka 网桥的纯部署，不会置备身份验证或授权，不支持与外部客户端的 TLS 加密连接。

3scale 可以通过 TLS 保护 Kafka 网桥，并提供身份验证和授权。与 3scale 集成还意味着还提供其他功能，如指标、速率限制和计费。

通过 3scale，您可以将不同类型的身份验证用于希望访问 AMQ Streams 的外部客户端的请求。3scale 支持以下类型的身份验证：

标准 API 密钥

单一随机字符串或哈希充当标识符和机密令牌。

应用程序标识符和密钥对

不可变标识符和可变 secret key 字符串。

OpenID Connect

用于委派的身份验证的协议.

使用现有的 3scale 部署？

如果您已将 3scale 部署到 OpenShift，并且希望将它与 Kafka 网桥搭配使用，请确保您有正确的设置。

设置在 第 7.2 节 “为 Kafka 网桥部署 3scale” 中描述。

7.1.1. Kafka Bridge 服务发现

3scale 是使用服务发现功能集成的，这需要 3scale 与 AMQ Streams 和 Kafka Bridge 部署到同一个 OpenShift 集群。

您的 AMQ Streams Cluster Operator 部署必须设置以下环境变量：

- STRIMZI_CUSTOM_KAFKA_BRIDGE_SERVICE_LABELS
- STRIMZI_CUSTOM_KAFKA_BRIDGE_SERVICE_ANNOTATIONS

部署 Kafka Bridge 时，公开 Kafka Bridge REST 接口的服务使用注解和标签来发现 3scale。

- 3 scale 使用 discovery.3scale.net=true 标签来查找服务。
- 注解提供有关服务的信息。

您可以通过导航到 Kafka Bridge 实例的 Services 来检查 OpenShift 控制台中的配置。您 会看到 Kafka 网桥的 OpenAPI 规格的端点。

7.1.2. 3scale APIcast 网关策略

3scale 与 3scale APIcast 一起使用，后者是部署有 3scale 的 API 网关，为 Kafka 网桥提供单一入口点。

APIcast 策略提供了一种自定义网关运行方式的机制。3scale 为网关配置提供一组标准策略。您还可以创建自己的策略。

如需有关 APIcast 策略的更多信息，请参阅 3scale 文档中的 管理 API 网关。

Kafka Bridge 的 APIcast 策略

policy _config.json 文件提供了 3scale 与 Kafka Bridge 集成的示例策略 配置，该文件定义了：

- 匿名访问

- 标头修改
- 路由
- URL 重写

通过此文件启用或禁用网关策略。

您可以使用此示例作为定义您自己的策略的起点。

匿名访问

匿名访问策略在没有身份验证的情况下公开服务，在 HTTP 客户端不提供它们时提供默认凭据（用于匿名访问）。策略不是强制的，如果始终需要身份验证，可以禁用或删除。

标头修改

标头修改策略允许修改现有的 HTTP 标头，或向通过网关的请求或响应添加新的标头。对于 3scale 集成，策略会为从 HTTP 客户端传递到 Kafka 网桥的每个请求添加标头。

当 Kafka Bridge 收到创建新消费者的请求时，它会返回包含 base_uri 字段的 JSON 有效负载，其中包含使用者必须用于所有后续请求的 URI。例如：

```
{
  "instance_id": "consumer-1",
  "base_uri":"http://my-bridge:8080/consumers/my-group/instances/consumer1"
}
```

使用 APIcast 时，客户端会将所有后续请求发送到网关，而不是直接发送到 Kafka 网桥。因此 URI 需要网关主机名，而不是网关后面的 Kafka 网桥地址。

使用标头修改策略，标头添加到来自 HTTP 客户端的请求中，以便 Kafka Bridge 使用网关主机名。

例如，通过应用 Forwarded: host=my-gateway:80;proto=http 标头，Kafka 网桥向使用者提供以下内容：

```
{
  "instance_id": "consumer-1",
  "base_uri":"http://my-gateway:80/consumers/my-group/instances/consumer1"
}
```

X-Forwarded-Path 标头承载从客户端到网关的请求中包含的原始路径。此标头与网关支持多个 Kafka Bridge 实例时应用的路由策略严格相关。

路由

当有多个 Kafka Bridge 实例时，会应用路由策略。请求必须发送到最初创建使用者的同一 Kafka Bridge 实例，因此请求必须指定网关将请求转发到适当的 Kafka Bridge 实例的路由。

每个网桥实例都有一个路由策略名称，路由则使用 名称来执行。部署 Kafka Bridge 时，您可以在 KafkaBridge 自定义资源中指定名称。

例如，从使用者到以下每个请求（使用 X-Forwarded-Path）

http://my-gateway:80/my-bridge-1/consumers/my-group/instances/consumer1

转发到：

http://my-bridge-1-bridge-service:8080/consumers/my-group/instances/consumer1

URL 重写策略会删除网桥名称，因为它在将请求从网关转发到 Kafka Bridge 时不会用到。

URL 重写

URL 重新连接策略确保从客户端到特定 Kafka Bridge 实例的请求在将网关的请求转发到 Kafka Bridge 时不包含网桥名称。

网桥名称不用于网桥公开的端点。

7.1.3. TLS 验证

您可以为 TLS 验证设置 APIcast，这需要使用模板自助管理 APIcast 部署。apicast 服务作为路由公

开。

您还可以将 TLS 策略应用到 Kafka Bridge API。

如需有关 TLS 配置的更多信息，请参阅 3scale 文档中的 管理 API 网关。

7.1.4. 3scale 文档

部署 3scale 以用于 Kafka Bridge 的步骤假定对 3scale 有一定的了解。

如需更多信息，请参阅 3scale 产品文档：

- Red Hat 3scale API 管理产品文档

7.2. 为 KAFKA 网桥部署 3SCALE

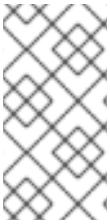
要将 3scale 与 Kafka 网桥搭配使用，首先对其进行部署，然后将其配置为发现 Kafka Bridge API。

您还将使用 3scale APIcast 和 3scale toolbox。

- APIcast 由 3scale 提供，作为基于 NGINX 的 API 网关，供 HTTP 客户端连接到 Kafka Bridge API 服务。

- 3scale toolbox 是一个配置工具，用于将 Kafka Bridge 服务的 OpenAPI 规格导入到 3scale。

在这种情况下，您将在同一 OpenShift 集群中运行 AMQ Streams、Kafka、Kafka Bridge 和 3scale/APIcast。



注意

如果您已在与 Kafka 网桥相同的集群中部署了 3scale，您可以跳过部署步骤并使用当前的部署。

先决条件

- AMQ Streams 和 Kafka 正在运行
- 部署 Kafka Bridge

对于 3scale 部署：

- 检查 Red Hat 3scale API Management 支持的配置。
- 安装时需要具有 cluster-admin 角色的用户，如 system:admin。
- 您需要访问以下 JSON 文件：
  - Kafka Bridge OpenAPI specification (openapiv2.json)
  - Kafka Bridge 的标头修改和路由策略(policies_config.json)

    在 GitHub 上查找 JSON 文件。

流程

1. 将 3scale API 管理部署到 OpenShift 集群。
   a. 创建新项目或使用现有的项目。

```
oc new-project my-project \
  --description="description" --display-name="display_name"
```

b.

部署 3scale.

使用 安装 3scale 指南中提供的信息，通过模板或操作器在 OpenShift 中部署 3scale。

无论您使用哪一种方法，请确保您将 WILDCARD_DOMAIN 参数设置为 OpenShift 集群的域。

记录用于访问 3scale 管理门户的 URLS 和凭据。

2.

为 3scale 授予发现 Kafka Bridge 服务的授权：

```
oc adm policy add-cluster-role-to-user view system:serviceaccount:my-project:amp
```

3.

通过 OpenShift 控制台或 CLI 验证 3scale 已成功部署到 Openshift 集群。

例如：

```
oc get deployment 3scale-operator
```

4.

设置 3scale toolbox。

a.

使用 Operating 3scale 指南中提供的信息来安装 3scale toolbox。

b.

设置环境变量以便与 3scale 交互：

```
export REMOTE_NAME=strimzi-kafka-bridge (1)
export SYSTEM_NAME=strimzi_http_bridge_for_apache_kafka (2)
export TENANT=strimzi-kafka-bridge-admin (3)
export PORTAL_ENDPOINT=$TENANT.3scale.net (4)
export TOKEN=3scale access token (5)
```

(1)

REMOTE_NAME 是分配给 3scale 管理门户的远程地址的名称。

2

SYSTEM_NAME 是 3scale 服务/API 的名称，通过 3scale toolbox 导入 OpenAPI 规格。

3

TENANT 是 3scale 管理门户的租户名称（即 https://$TENANT.3scale.net）。

4

PORTAL_ENDPOINT 是运行 3scale 管理门户的端点。

5

TOKEN 是 3scale 管理门户提供的访问令牌，用于通过 3scale toolbox 或 HTTP 请求进行交互。

c. *配置 3scale toolbox 的远程 Web 地址：*

```
3scale remote add $REMOTE_NAME https://$TOKEN@$PORTAL_ENDPOINT/
```

现在，当您每次运行 toolbox 时，不需要指定 3scale 管理门户的端点地址。

5. *检查您的 Cluster Operator 部署是否有 3scale 发现 Kafka Bridge 服务所需的标签和注解属性。*

```
#...
env:
- name: STRIMZI_CUSTOM_KAFKA_BRIDGE_SERVICE_LABELS
    value: |
    discovery.3scale.net=true
- name: STRIMZI_CUSTOM_KAFKA_BRIDGE_SERVICE_ANNOTATIONS
    value: |
    discovery.3scale.net/scheme=http
    discovery.3scale.net/port=8080
    discovery.3scale.net/path=/
    discovery.3scale.net/description-path=/openapi
#...
```

如果没有，请通过 OpenShift 控制台添加属性或尝试重新部署 Cluster Operator 和 Kafka Bridge。

6. 通过 3scale 发现 Kafka Bridge API 服务。

   a. 使用部署 3scale 时提供的凭据，登录 3scale 管理门户。

   b. 从 3scale 管理门户，导航到 OpenShift 中的 New API Import，其中您将看到 Kafka Bridge 服务。

   c. 点 Create Service。

   您可能需要刷新页面来查看 Kafka Bridge 服务。

   现在，您需要导入该服务的配置。从编辑器执行此操作，但请保持门户打开以检查导入是否成功。

7. 编辑 OpenAPI 规格（JSON 文件）中的 Host 字段，以使用 Kafka Bridge 服务的基本 URL：

   例如：

   ```
   "host": "my-bridge-bridge-service.my-project.svc.cluster.local:8080"
   ```

   检查 主机 URL 中包含正确的内容：

   - Kafka Bridge 名称(my-bridge)
   - 项目名称(my-project)
   - Kafka 网桥的端口(8080)

8. 使用 3scale toolbox 导入更新的 OpenAPI 规格：

```
3scale import openapi -k -d $REMOTE_NAME openapiv2.json -t myproject-my-bridge-bridge-service
```

9. 导入服务的标头修改和路由策略（JSON 文件）。

   a. 查找您在 3scale 中创建的服务 ID。

      在这里，我们使用 "jq"实用程序 ：

      ```
      export SERVICE_ID=$(curl -k -s -X GET "https://$PORTAL_ENDPOINT/admin/api/services.json?access_token=$TOKEN" | jq ".services[] | select(.service.system_name | contains(\"$SYSTEM_NAME\")) | .service.id")
      ```

      导入策略时需要 ID。

   b. 导入策略：

      ```
      curl -k -X PUT "https://$PORTAL_ENDPOINT/admin/api/services/$SERVICE_ID/proxy/policies.json" --data "access_token=$TOKEN" --data-urlencode policies_config@policies_config.json
      ```

10. 从 3scale Admin Portal，导航到 Integration → Configuration，以检查 Kafka Bridge 服务的端点和策略是否已加载。

11. 导航到 Applications → Create Application Plan 以创建应用计划。

12. 导航到 Audience → Developer → Applications → Create Application 以创建应用。

    需要应用才能获取用于身份验证的用户密钥。

13. （生产环境步骤）要让 API 可供生产网关使用，请提升配置：

    ```
    3scale proxy-config promote $REMOTE_NAME $SERVICE_ID
    ```

14.

使用 API 测试工具，验证您可以通过 APIcast 网关使用调用创建使用者以及为应用程序创建用户密钥来访问 Kafka 网桥。

例如：

```
https//my-project-my-bridge-bridge-service-3scale-apicast-staging.example.com:443/consumers/my-group?user_key=3dfc188650101010ecd7fdc56098ce95
```

如果从 Kafka Bridge 返回有效负载，则消费者创建成功。

```
{
  "instance_id": "consumer1",
  "base uri": "https//my-project-my-bridge-bridge-service-3scale-apicast-staging.example.com:443/consumers/my-group/instances/consumer1"
}
```

基础 URI 是客户端要在后续请求中使用的地址。

第 8 章 用于集群重新平衡的精简控制

您可以将 Cruise Control 部署到 AMQ Streams 集群，并使用它来 重新平衡 Kafka 集群。

cruise Control 是一个开源系统，用于自动执行 Kafka 操作，如监控集群工作负载、根据预定义的限制重新平衡集群，以及检测和修复异常情况。它包含四个主要组件 - Load Monitor、Analyzer、Anomaly Detector 和 Executor-，以及用于客户端交互的 REST API。AMQ Streams 利用 REST API 支持以下 Cruise Control 功能：

- 从多个 优化 目标生成优化。
- 根据优化建议重新平衡 Kafka 集群。

目前不支持其他 Cruise 控制功能，包括异常检测、通知、写入目标以及更改主题复制因素。

示例 /cruise-control/ 中提供了 Cruise 控制的 YAML 文件示例。

8.1. 为什么使用清理控制？

cruise Control 可减少运行高效、均衡 Kafka 集群所需的时间和工作量。

典型的群集可能会随着时间推移而变得异常加载。处理大量消息流量的分区可能会在可用的代理中不均匀分布。要重新平衡集群，管理员必须监控代理上的负载，并手动将繁忙的分区重新分配给具有备用容量的代理。

整合控制可自动执行集群重新平衡过程。它基于 CPU、磁盘和网络负载为 cluster- 构造一个资源利用率 工作负载模型，并为更均衡的分区分配生成优化调整（您可以批准或拒绝）。使用一组可配置的优化目标来计算这些假设。

当您批准一个优化建议时，Cruise Control 将其应用到 Kafka 集群。当集群重新平衡操作完成后，代理 Pod 会被更有效地使用，Kafka 集群也会更加均匀地平衡。

其它资源

- Scope Control Wiki

8.2. 优化目标概述

要重新平衡 Kafka 集群，Cruise Control 使用优化目标来生成 优化方法，您可以批准或拒绝这些优化。

优化目标是针对 Kafka 集群内工作负载重新分配和资源利用率的限制。AMQ Streams 支持在 Cruise Control 项目中开发的大多数优化目标。支持的目标（按默认降序排列）如下：

1. 机架感知
2. 每个代理针对一组主题的领导副本最少数量
3. 副本容量
4. 容量 ：磁盘容量、网络入站容量、网络出站容量、CPU 容量
5. 副本发布
6. 潜在的网络输出
7. 资源分布 ：磁盘利用率分布、网络入站利用率分布、网络出站利用率分布、CPU 利用率分布

注意

资源分配目标使用代理 资源的能力限制 来控制。

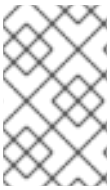
8. 领导字节速率分布

9. *主题副本发布*

10. *领导程序副本发布*

11. *首选领导选举机制*

有关每个优化目标的更多信息，请参阅 Cruise Control Wiki 中的目标。



注意

尚不支持broker 磁盘目标、"写入您自己的"目标和 Kafka 分配程序目标。

AMQ Streams 自定义资源中的目标配置

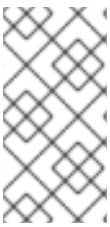
您可以在 Kafka 和 Kafka Rebalance 自定义资源中配置优化目标。整合控制具有必须满足 的硬 优化目标配置，以及主要、默认 和用户提供的 优化目标。资源分布（磁盘、网络入站、网络出站和 CPU）的优化目标受到代理资源 的容量限制。

以下小节更详细地描述了每个目标配置。

硬目标和软目标

硬目标就是在优化调整时 必须满足 的目标。未配置为硬目标的目标称为 软目标。您可以将软目标视为最佳工作 目标：它们在优化调整 时不 需要满足，而是包含在优化计算中。违反一个或多个软目标但满足所有硬目标的最佳建议是有效的。

粗体控件将计算出符合所有硬目标和尽可能多的软目标（按优先顺序）的优化条件。无法 满足所有硬目标的优化建议被拒绝，不会发送给用户进行审批。



注意

例如，您可能有一个软目标来在集群中平均分发主题的副本（主题副本分发目标）。如果这样做能够实现所有配置的硬目标，挑战控制将忽略此目标。

在 Cruise Control 中，以下主要优化目标是 预先设定为硬目标：

> RackAwareGoal; MinTopicLeadersPerBrokerGoal; ReplicaCapacityGoal; DiskCapacityGoal; NetworkInboundCapacityGoal; NetworkOutboundCapacityGoal; CpuCapacityGoal

您可以通过编辑 Kafka.spec.cruiseControl.config 中的 hard.goals 属性，在 Cruise Control 部署配置中配置硬目标。

- 要从 Cruise Control 继承预先设置的硬链接，请不要在 Kafka.spec.cruiseControl.config 中指定 hard. goals属性
- 要改变预先设定的硬目标，请使用其完全限定域名在 hard.goals 属性中指定预期的目标。

硬优化目标的 Kafka 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    topicOperator: {}
    userOperator: {}
  cruiseControl:
    brokerCapacity:
      inboundNetwork: 10000KB/s
      outboundNetwork: 10000KB/s
    config:
      hard.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundCapacityGoal
      # ...
```

增加配置的硬目标数量将降低 Cruise 控制产生有效优化判断的可能性。

如果在 KafkaRebalance 自定义资源中指定了 skipHardGoalCheck: true，Cruise Control 不会 检查用户提供的优化目标（在 KafkaRebalance.spec.goals 中）是否包含 所有 配置的硬目标(hard.goals)。

因此，如果一些（但不是全部）用户提供的优化目标在 hard.goals 列表中，即使指定了 skipHardGoalCheck: true，Cruise Control 仍会将它们视为硬目标。

主要优化目标

主要优化目标 对所有用户可用。未在主要优化目标中列出的目标不适用于 Cruise 控制操作。

除非更改了 Cruise Control 部署配置，A MQ Streams 将从 Cruise Control 中继承以下主要优化目标，其优先级降序排列：

> RackAwareGoal; ReplicaCapacityGoal; DiskCapacityGoal; NetworkInboundCapacityGoal; NetworkOutboundCapacityGoal; CpuCapacityGoal; ReplicaDistributionGoal; PotentialNwOutGoal; DiskUsageDistributionGoal; NetworkInboundUsageDistributionGoal; NetworkOutboundUsageDistributionGoal; CpuUsageDistributionGoal; TopicReplicaDistributionGoal; LeaderReplicaDistributionGoal; LeaderBytesInDistributionGoal; PreferredLeaderElectionGoal

其中六个目标被预先设定为 困难的目标。

为降低复杂性，建议您使用继承的主要优化目标，除非您需要 完全排除 在 KafkaRebalance 资源中使用的一个或多个目标。如果需要，可在配置 默认优化目标时修改主要优化目标 的优先级顺序。

您可以在 Cruise Control 部署配置中配置主要优化目标： Kafka.spec.cruiseControl.config.goals

- 要接受继承的主要优化目标，请不要在 Kafka.spec.cruiseControl.config 中指定 目标 属性。
- 如果您需要修改继承的主要优化目标，请在 目标 配置选项中以降序排列为目标列表。

注意

如果更改了继承的主要优化目标，您必须确保在 Kafka.spec.cruiseControl .config 中的 hard. goals 属性中配置的硬目标是您配置的主要优化目标的子集。否则，生成优化时会出现错误。

默认优化目标

精简控制 使用默认优化目标 来生成 缓存优化建议。有关缓存优化建议的详情请参考 第 8.3 节 “优化调整概述”。

您可以通过在 KafkaRebalance 自定义资源中设置 用户提供的优化目标 来覆盖默认优化目标。

除非在 Cruise Control 部署配置 中指定了 default.goals，否则主要优化目标将用作默认的优化目标。在这种情况下，缓存的优化建议是使用主要优化目标生成的。

- 要将主要优化目标用作默认目标，请不要在 Kafka.spec.cruiseControl.config 中指定 default. goals 属性。
- 要修改默认优化目标，请编辑 Kafka.spec.cruiseControl.config 中的 default. goals 属性。您必须使用主要优化目标的子集。

默认优化目标的 Kafka 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    topicOperator: {}
    userOperator: {}
  cruiseControl:
    brokerCapacity:
      inboundNetwork: 10000KB/s
      outboundNetwork: 10000KB/s
    config:
      default.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal
      # ...
```

如果没有指定默认优化目标，则使用主要优化目标生成缓存建议。

用户提供的优化目标

用户提供的优化目标缩小了针对特定优化建议配置的默认目标。您可以根据需要在 KafkaRebalance 自定义资源中的 spec.goals 中设置它们：

```
KafkaRebalance.spec.goals
```

用户提供的优化目标可针对不同的情景生成优化。例如，您可能想要在 Kafka 集群间优化领导副本分布，而不考虑磁盘容量或磁盘使用率。因此，您可以创建一个 KafkaRebalance 自定义资源，其中包含领导副本分发的用户提供的目标。

用户提供的优化目标必须：

- 包括所有配置的硬目标或发生错误
- 成为主要优化目标的子集

要在生成优化建议时忽略配置的硬目标，请将 skipHardGoalCheck: true 属性添加到 KafkaRebalance 自定义资源。请参阅 第 8.7 节 “生成优化分析”。

其它资源

- 第 8.5 节 “精简控制配置”
- Cruise Control Wiki 中的 配置.

8.3. 优化调整概述

优化建议是建议更改的总结，这些更改将产生更为均衡的 Kafka 集群，在代理间更均匀地分配分区工作负载。每个优化建议都基于用于生成目标的一组优化目标，受代理资源配置容量限制的影响。

KafkaRebalance 自定义资源的 Status.Optimization Result 属性中包含了一个优化建议。提供的信息是完整优化建议摘要。使用摘要决定是否：

- 批准优化建议。这指示 Cruise Control 将提议应用到 Kafka 集群并启动集群重新平衡操作。

- 驳回优化建议。您可以更改优化目标，然后产生另一个建议。

所有优化调整都是 空运行 ：您无法在不首先生成优化建议的情况下批准集群重新平衡。对可生成的优化说明数量没有限制。

缓存优化建议

整合控制根据配置的默认 优化目标维护缓存优化建议。从工作负载模型生成的缓存优化建议每 15 分钟更新一次，以反映 Kafka 集群的当前状态。如果您使用默认优化目标生成一个优化建议，Cruise Control 将返回最新的缓存建议。

要更改缓存的优化建议刷新间隔，请编辑 Cruise Control 部署配置中的 proposed.expiration.ms 设置。为快速更改集群考虑一个较短的间隔，尽管这会增加 Cruise Control 服务器上的负载。

优化调整的内容

优化建议由两个主要部分组成：

- 摘要 存储在 KafkaRebalance 资源的状态中。
- 代理负载 存储在 ConfigMap 中，该 ConfigMap 以 JSON 字符串形式包含数据。

摘要概述了建议的群集重新平衡，并指示相关更改的规模。代理负载在建议的重新平衡值前后显示，因此您可以看到对集群中的每个代理的影响。

概述

下表解释了优化建议摘要部分中的属性：

表 8.1. 优化建议中包含的属性

JSON 属性	描述
numIntraBrokerReplicaMovements	在集群的代理磁盘之间传输的分区副本总数。 **重新平衡操作期间的性能影响** ：相对较高，但低于 **numReplicaMovements**。
excludedBrokersForLeadership	尚不支持.返回一个空列表。

JSON 属性	描述
numReplicaMovements	在独立代理之间移动的分区副本数。 **重新平衡操作期间的性能影响** ：相对高.
onDemandBalancednessScoreBefore, onDemandBalancednessScoreAfter	对生成优化建议之前和之后的 Kafka 集群的整体 *平衡性* 进行测量。 分数是通过从 100 减去每个违反软目标的 **BalancednessScore** 的总和来计算的。海军控件根据以下几个因素为每个优化目标分配 a **BalancednessScore**，包括优先级 - 在 **default.goals** 或用户提供的目标列表中的位置。 **Before** 分数基于 Kafka 集群的当前配置。**After** 分数基于生成的优化建议。
intraBrokerDataToMoveMB	在同一代理的磁盘之间移动的每个分区副本的大小总和（请参阅 **numIntraBrokerReplicaMovements**）。 **重新平衡操作期间的性能影响** ：变量.数量越大，集群重新平衡所需的时间也越长。在相同代理的磁盘之间移动大量数据比不同代理之间的影响小（请参阅 **dataToMoveMB**）。
recentWindows	优化建议所基于的指标窗口数量。
dataToMoveMB	移动到独立代理的每个分区副本的大小总和（请参阅 同时也是 **numReplicaMovements**）。 **重新平衡操作期间的性能影响** ：变量.数量越大，集群重新平衡所需的时间也越长。
monitoredPartitionsPercentage	优化建议涵盖 Kafka 集群中分区的百分比。受 **排除主题数量的影响.**
excludedTopics	如果您在 **KafkaRebalance** 资源中的 **spec.excludedTopicsRegex** 属性中指定了正则表达式，则与该表达式匹配的所有主题名称都会在此列出。这些主题不包括在优化建议中的分区副本/领导移动计算中。
numLeaderMovements	领导者将切换到不同副本的分区数量。这涉及对 ZooKeeper 配置的更改。 **重新平衡操作期间的性能影响** ：相对较低的.
excludedBrokersForReplicaMove	尚不支持.返回一个空列表。

代理负载

代理负载与 JSON 格式字符串存储在 ConfigMap 中（其名称与 KafkaRebalance 自定义资源相同）。此 JSON 字符串由 JSON 对象组成，其中包含每个代理 ID 的密钥，用于链接到每个代理的多个指

标。每个指标包含三个值：第一个是应用优化建议前的指标值，第二个是应用 建议后指标的预期值，第三个是前两个值（早于减后）。

要从 ConfigMap 中提取 JSON 字符串，您可以使用以下命令，该命令使用 jq 命令行 JSON 解析器工具：

```
oc get configmap MY-REBALANCE -o json | jq '.["data"]["brokerLoad.json"]|fromjson|.'
```

下表解释了优化建议的代理负载 ConfigMap 中包含的属性：

JSON 属性	描述
leaders	此代理中作为分区领导的副本数。
replicas	此代理中的副本数。
cpuPercentage	CPU 利用率为定义容量的百分比。
diskUsedPercentage	磁盘利用率为定义容量的百分比。
diskUsedMB	以 MB 为单位的绝对磁盘使用量。
networkOutRate	代理的网络输出率总数。
leaderNetworkInRate	此代理上所有分区领导副本的网络输入率。
followerNetworkInRate	此代理上所有后续副本的网络输入率。
potentialMaxNetworkOutRate	如果这个代理成为当前主机的所有副本的领导，可以实现假设的最大网络输出率。

其它资源

- *第 8.2 节 “优化目标概述”*
- *第 8.7 节 “生成优化分析”*
- *第 8.8 节 “批准优化建议”*

8.4. 重新平衡性能调优概述

您可以为集群重新平衡调整几个性能调整选项。这些选项控制如何执行重新平衡中的副本和领导移动，以及分配给重新平衡操作的带宽。

分区重新分配命令

优化调整 由单独的分区重新分配命令组成。当您 批准 一个建议时，Cruise Control 服务器将这些命令应用到 Kafka 集群。

分区重新分配命令由以下任一操作类型组成：

- 分区移动：将分区副本及其数据传输到新位置。分区移动可采用以下两种形式之一：
  - Broker 移动：分区副本移到不同代理上的日志目录中。
  - Broker 内部移动：分区副本移到同一代理上的不同日志目录中。
- 领导移动：这涉及切换分区副本的领导机。

批量将控制问题的分区重新分配给 Kafka 集群的命令。集群在重新平衡期间的性能会受到每个批处理中包含的每种移动类型的数量的影响。

副本移动策略

集群重新平衡性能还受到应用于分区重新分配命令 的副本移动策略 的影响。默认情况下，Cruise Control 使用 BaseReplicaMovementStrategy，它只按照生成命令的顺序应用它们。但是，如果在建议早期存在一些非常大的分区重新分配，此策略可能会减慢其他重新分配的应用速度。

海军控件提供了四种替代副本移动策略，可用于优化调整：

- PrioritizeSmallReplicaMovementStrategy: Order reassignments按升序大小排序。
- PrioritizeLargeReplicaMovementStrategy: Order reassignments按降序排列大小。

- *PostponeUrpReplicaMovementStrategy ：优先分配没有同步副本的分区副本。*

- *PrioritizeMinIsrWithOfflineReplicasStrategy ：使用(At/Under)MinISR 分区优先级重新分配带有离线副本。只有 Kafka 自定义资源的 spec 中将 cruiseControl.config.concurrency.adjuster.min.isr.check.enabled 设置为 true 时，此策略才会有效。*

这些策略可以配置为序列。第一种策略尝试使用其内部逻辑比较两个分区重新分配。如果重新分配是等效的，那么它会将它们传递给序列中的下一个策略，以确定顺序，以此类推。

重新平衡调优选项

cruise Control 提供多个配置选项来调整上面讨论的重新平衡参数。您可以在 Cruise Control 服务器或 优化建议 级别设置这些调整选项：

- *Cruise Control 服务器设置可以在 Kafka .spec.cruiseControl.config 下的 Kafka 自定义资源中设置。*

- *单独的重新平衡性能配置可以在 KafkaRebalance.spec 下设置。*

相关配置总结如下：

服务器和 KafkaRebalance 配置	描述	默认值
num.concurrent.partition.movements.per.broker	每个分区重新分配批次中broker 分区移动的最大数量	5
concurrentPartitionMovementsPerBroker		
num.concurrent.intra.broker.partition.movements	每个分区重新分配批次中broker 分区移动的最大数量	2
concurrentIntraBrokerPartitionMovements		
num.concurrent.leader.movements	每个分区重新分配批中分区领导更改的最大数量	1000

服务器和 KafkaRebalance 配置	描述	默认值
concurrentLeaderMovements		
default.replication.throttle **replicationThrottle**	要分配给分区重新分配的带宽（以字节/秒为单位）	无限制
default.replica.movement.strategies **replicaMovementStrategies**	用于确定执行分区重新分配命令的顺序的策略列表（按优先级顺序排列）。 对于 server 设置，使用逗号分隔的字符串以及策略类的完全限定名称（在每个类名称 **的开头添加 com.linkedin.kafka.cruisecontrol.executor.strategy** ）。对于 **KafkaRebalance** 资源设置，请使用策略类名称的 YAML 数组。	**BaseReplicaMovementStrategy**

更改默认设置会影响重新平衡完成所需的时间，以及重新平衡过程中在 Kafka 集群上放置的负载。使用较低值可以减少负载，但会增加所需时间，反之亦然。

其它资源

- ***第 13.2.50 节 “CruiseControlSpec 模式参考”.***
- ***第 13.2.131 节 “KafkaRebalanceSpec 模式参考”.***

8.5. 精简控制配置

Kafka.spec.cruiseControl 中的 config 属性包含作为键的配置选项，其值为以下 JSON 类型之一：

- ***字符串***
- ***数字***
- ***布尔值***

除了由 AMQ Streams 直接管理的选项外，您还可以指定并配置 Cruise Control 文档 的"配置"一节中列出的所有选项。特别是，您无法 修改与 此处 提及的其中一个键相等或开头的键的配置选项。

如果指定了 restricted 选项，则忽略它们，并在 Cluster Operator 日志文件中输出警告信息。所有支持的选项都传递给 Cruise Control。

Cruise Control 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    # ...
    config:
      default.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal
      cpu.balance.threshold: 1.1
      metadata.max.age.ms: 300000
      send.buffer.bytes: 131072
    # ...
```

跨操作系统资源共享配置

通过跨操作资源共享(CORS)，您可以指定访问 REST API 的允许方法和原始 URL。

默认情况下，Cruise Control REST API 禁用 CORS。启用后，只允许对 Kafka 集群状态进行只读访问的 GET 请求。这意味着外部应用以与 AMQ Streams 组件不同的来源运行，无法向 Cruise Control API 发出 POST 请求。但是，这些应用程序可能会发出 GET 请求 来访问 Kafka 集群的只读信息，如当前集群负载或最新的优化建议。

为 Cruise Control 启用 CORS

您可以在 Kafka.spec.cruiseControl.config 中启用并配置 CORS。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
```

```
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    # ...
    config:
      webserver.http.cors.enabled: true
      webserver.http.cors.origin: "*"
      webserver.http.cors.exposeheaders: "User-Task-ID,Content-Type"
    # ...
```

如需更多信息，请参阅 Cruise Control Wiki 中的 REST API。

容量配置

整合控制使用 容量限制 来确定资源分布的优化目标是否受到破坏。这种类型的四个目标：

- DiskUsageDistributionGoal - 磁盘使用率分布
- CpuUsageDistributionGoal - CPU 使用率分布
- NetworkInboundUsageDistributionGoal - 网络入站利用率分布
- NetworkOutboundUsageDistributionGoal - 网络出站利用率分布

您可以在 Kafka. spec.cruiseControl 中的 brokerCapacity 属性中为 Kafka 代理资源指定容量限值。它们默认是启用的，您可以更改其默认值。可以使用标准 OpenShift 字节单元（K、M、G 和 T）或其双字节（双倍的幂等）等效项（Ki、Mi、Gi 和 Ti）来为以下代理资源设置容量限制：

- 磁盘 - 每个代理的磁盘存储（默认值：100000Mi）
- cpuUtilization - CPU 使用率作为百分比（默认值：100）
- Inbound Network - 入站网络吞吐量每秒字节数（默认值：10000KiB/s）

- 出站网络 - 出站网络吞吐量每秒字节数（默认值：10000KiB/s）

因为 AMQ Streams Kafka 代理是同构的，所以 Cruise Control 会为其监控的每个代理应用相同的容量限制。

使用 Bbyte 单元的 Cruise Control 代理功能配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    # ...
    brokerCapacity:
      disk: 100Gi
      cpuUtilization: 100
      inboundNetwork: 10000KiB/s
      outboundNetwork: 10000KiB/s
    # ...
```

其它资源

如需更多信息，请参阅 第 13.2.52 节 “BrokerCapacity schema 参考”。

日志记录配置

精简控制具有自己的可配置日志记录器：

- rootLogger.level

整合控制使用 Apache log4j 2 日志记录器实施。

使用 logging 属性来配置日志记录器和日志记录器级别。

您可以通过直接（内线）指定日志记录器和级别来设置日志级别，或使用自定义（外部）ConfigMap。如果使用 ConfigMap，则将 logging.valueFrom.configMapKeyRef.name 属性设置为包含外部日志记录配置的 ConfigMap 的名称。在 ConfigMap 中，日志配置使用 log4j.properties 进行 描述。logging.valueFrom.configMapKeyRef.name 和 logging.valueFrom.configMapKeyRef.key 属性均是必需的。在运行 Cluster Operator 时，会使用自定义资源创建使用指定准确日志配置的 ConfigMap，然后在每次协调后重新创建。如果没有指定自定义 ConfigMap，则会使用默认日志设置。如果没有设置特定的日志记录器值，则会继承该日志记录器的上一级日志记录器设置。此处我们会看到 内联 和外部 记录示例。

内联日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
# ...
spec:
  cruiseControl:
    # ...
    logging:
      type: inline
      loggers:
        rootLogger.level: "INFO"
    # ...
```

外部日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
# ...
spec:
  cruiseControl:
    # ...
    logging:
      type: external
      valueFrom:
        configMapKeyRef:
          name: customConfigMap
          key: cruise-control-log4j.properties
    # ...
```

8.6. 部署清理控制

要将 Cruise Control 部署到 AMQ Streams 集群，请使用 Kafka 资源中的 cruise Control 属性定义配置，然后创建或更新资源。

每个 Kafka 集群部署一个 Cruise Control 实例。

先决条件

- OpenShift 集群
- 一个正在运行的 Cluster Operator

流程

1. 编辑 Kafka 资源并添加 cruise Control 属性。

   您可以配置的属性显示在此示例配置中：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    brokerCapacity: 1
      inboundNetwork: 10000KB/s
      outboundNetwork: 10000KB/s
      # ...
    config: 2
      default.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal
        # ...
      cpu.balance.threshold: 1.1
      metadata.max.age.ms: 300000
      send.buffer.bytes: 131072
      # ...
    resources: 3
      requests:
        cpu: 1
        memory: 512Mi
      limits:
        cpu: 2
        memory: 2Gi
```

```
    logging: 4
      type: inline
      loggers:
        rootLogger.level: "INFO"
    template: 5
      pod:
        metadata:
          labels:
            label1: value1
        securityContext:
          runAsUser: 1000001
          fsGroup: 0
        terminationGracePeriodSeconds: 120
    readinessProbe: 6
      initialDelaySeconds: 15
      timeoutSeconds: 5
    livenessProbe: 7
      initialDelaySeconds: 15
      timeoutSeconds: 5
  # ...
```

1

指定代理资源的容量限制。如需更多信息，请参阅 容量配置。

2

定义 Cruise 控制配置，包括默认优化目标（默认为默认值），以及对主要优化目标（目标）或硬目标（硬目标）的自定义。除了由 AMQ Streams 直接管理的以外，您还可以提供任何 标准 Cruise Control 配置选项。有关配置优化目标的详情请参考 第 8.2 节 “优化目标概述”。

3

为 Cruise Control 保留的 CPU 和内存资源.如需更多信息，请参阅 第 13.1.5 节 “资源”。

4

定义的日志记录器和日志级别通过 ConfigMap 直接（内线）或间接（外部）添加。自定义 ConfigMap 必须放在 log4j.properties 键下。登陆控制具有一个名为 rootLogger.level 的单个日志记录器。您可以将日志级别设置为 INFO、ERROR、WARN、TRACE、DEBUG、FATAL 或 OFF。如需更多信息，请参阅 日志配置。

5

自定义部署模板和容器集.

6

健康检查就绪度探测.

7 *健康检查存活度探测*.

2. *创建或更新资源：*

```
oc apply -f kafka.yaml
```

3. *验证 Cruise Control 是否已成功部署：*

```
oc get deployments -l app.kubernetes.io/name=cruise-control
```

自动创建的主题

下表显示了部署 Cruise Control 时自动创建的三个主题。这些主题是必需的，清理控制才能正常工作，且不得删除或更改。

表 8.2. 自动创建的主题

自动创建的主题	创建者	功能
strimzi.cruisecontrol.metrics	AMQ Streams Metrics Reporter	将 Metrics Reporter 中的原始指标存储在每个 Kafka 代理中。
strimzi.cruisecontrol.partitionmetricsamples	Sything Control	存储每个分区派生的指标。它们由 Metric Sample Aggregator 创建。
strimzi.cruisecontrol.modeltrainingsamples	Sything Control	存储用于创建集群 工作负载模型 的指标示例。

为防止删除 Cruise Control 所需的记录，在自动创建的主题中禁用了日志压缩。

接下来要做什么

配置和部署 Cruise Control 后，您可以 生成优化建议。

其它资源

第 13.2.51 节 “CruiseControlTemplate 模式参考”.

8.7. 生成优化分析

当您创建或更新 KafkaRebalance 资源时，Cruise Control 会根据配置的 优化目标为 Kafka 集群生成优化建议。

分析优化建议中的信息，并决定是否批准它。

先决条件

- 您已将 Cruise Control 部署到 AMQ Streams 集群中。
- 您已配置了 优化目标，以及可选 的代理资源容量限制。

流程

1. 创建 KafkaRebalance 资源：

   a. 要使用 Kafka 资源中定义的 默认优化目标，请将 spec 属性留空：

   ```
   apiVersion: kafka.strimzi.io/v1beta2
   kind: KafkaRebalance
   metadata:
     name: my-rebalance
     labels:
       strimzi.io/cluster: my-cluster
   spec: {}
   ```

   b. 要配置 用户提供的优化目标 而不是使用默认目标，请添加 target 属性并输入一个或多个目标。

   在以下示例中，机架意识和副本容量被配置为用户提供的优化目标：

   ```
   apiVersion: kafka.strimzi.io/v1beta2
   kind: KafkaRebalance
   metadata:
     name: my-rebalance
   ```

```
  labels:
    strimzi.io/cluster: my-cluster
spec:
  goals:
    - RackAwareGoal
    - ReplicaCapacityGoal
```

c. 要忽略配置的硬目标，请添加 skipHardGoalCheck: true 属性：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
  name: my-rebalance
  labels:
    strimzi.io/cluster: my-cluster
spec:
  goals:
    - RackAwareGoal
    - ReplicaCapacityGoal
  skipHardGoalCheck: true
```

2. 创建或更新资源：

```
oc apply -f your-file
```

Cluster Operator 从 Cruise Control 请求优化建议。这可能需要一些时间，具体取决于 Kafka 集群的大小。

3. 检查 KafkaRebalance 资源的状态：

```
oc describe kafkarebalance rebalance-cr-name
```

cruise Control 返回两个状态之一：

- PendingProposal ：重新平衡操作器正在轮询 Cruise Control API，以检查优化建议是否已就绪。
- ProposalReady ：优化建议已准备好审核，并在需要时予以批准。优化建议包含在 KafkaRebalance 资源的 Status.Optimization Result 属性中。

4. 查看优化建议。

```
oc describe kafkarebalance rebalance-cr-name
```

下面是一个示例：

```
Status:
  Conditions:
    Last Transition Time:  2020-05-19T13:50:12.533Z
    Status:                ProposalReady
    Type:                  State
  Observed Generation:     1
  Optimization Result:
    Data To Move MB:  0
    Excluded Brokers For Leadership:
    Excluded Brokers For Replica Move:
    Excluded Topics:
    Intra Broker Data To Move MB:         0
    Monitored Partitions Percentage:      100
    Num Intra Broker Replica Movements:   0
    Num Leader Movements:                 0
    Num Replica Movements:                26
    On Demand Balancedness Score After:   81.8666802863978
    On Demand Balancedness Score Before:  78.01176356230222
    Recent Windows:                       1
  Session Id:                             05539377-ca7b-45ef-b359-e13564f1458c
```

Optimization Result 部分中的属性描述待处理的集群重新平衡操作。有关每个属性的描述，请参阅 优化方法的内容。

接下来要做什么

第 8.8 节 “批准优化建议”

其它资源

- 第 8.3 节 “优化调整概述”

8.8. 批准优化建议

如果 Cruise Control 的状态是 Proposal Ready，则批准由 Cruise Control 生成的 优化建议 。然后，Bootation Control 会将优化建议应用到 Kafka 集群，将分区重新分配给代理并更改分区领导。

小心

这不是空运行。在批准优化建议前，您必须：

- 刷新该提议，以防其过时。
- 仔细检查 提议的内容。

先决条件

- 您已 通过 Cruise 控制生成了一个优化建议。
- KafkaRebalance 自定义资源状态是 Proposal Ready。

流程

执行这些步骤进行您要批准的优化建议：

1. 除非优化建议是新生成的，否则请检查它是否基于当前 Kafka 集群状态的信息。要做到这一点，刷新优化建议以确保它使用最新的集群指标：

   a. 使用 刷新给 OpenShift 中的 KafkaRebalance 资源标注 ：

   ```
   oc annotate kafkarebalance rebalance-cr-name strimzi.io/rebalance=refresh
   ```

   b. 检查 KafkaRebalance 资源的状态：

   ```
   oc describe kafkarebalance rebalance-cr-name
   ```

   c. 等待状态更改为 Proposal Ready。

2. 批准您希望应用 Cruise Control 的优化建议。

给 OpenShift 中的 KafkaRebalance 资源标注：

```
oc annotate kafkarebalance rebalance-cr-name strimzi.io/rebalance=approve
```

3. Cluster Operator 会检测到注解的资源，并指示 Cruise Control 重新平衡 Kafka 集群。

4. 检查 KafkaRebalance 资源的状态：

```
oc describe kafkarebalance rebalance-cr-name
```

5. cruise Control 返回三个状态之一：

- 重新平衡：集群重新平衡操作正在进行中。
- Ready：集群重新平衡操作成功完成。要使用同一 KafkaRebalance 自定义资源生成另一个优化建议，请将 刷新 注解应用到自定义资源。这会将自定义资源移到 PendingProposal 或 Proposal Ready 状态。然后，您可以审查优化建议并根据需要批准该提议。
- NotReady：出错信息 - 请查看 第 8.10 节 “修复 KafkaRebalance 资源的问题”。

其它资源

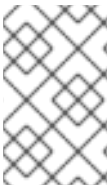
- 第 8.3 节 “优化调整概述”
- 第 8.9 节 “停止集群重新平衡”

8.9. 停止集群重新平衡

启动后，集群重新平衡操作可能需要一段时间才能完成，并影响 Kafka 集群的整体性能。

如果要停止正在进行中的集群重新平衡操作，请将 stop 注解应用到 KafkaRebalance 自定义资源。这指示 Cruise Control 完成当前的一组分区重新分配，然后停止重新平衡。重新平衡停止后，已完成的分

区重新分配已被应用；因此，与在重新平衡操作开始前相比，Kafka 集群的状态会有所不同。如果需要进一步重新平衡，您应该生成一个新的优化建议。



注意

Kafka 集群处于中间（停止）状态的性能可能比初始状态差。

先决条件

- 您已通过注解 KafkaRebalance 自定义资源 批准了优化建议 。
- KafkaRebalance 自定义资源的状态是 重新平衡。

流程

1. 给 OpenShift 中的 KafkaRebalance 资源标注：

```
oc annotate kafkarebalance rebalance-cr-name strimzi.io/rebalance=stop
```

2. 检查 KafkaRebalance 资源的状态：

```
oc describe kafkarebalance rebalance-cr-name
```

3. 等待状态更改为 Stopped。

其它资源

- 第 8.3 节 “优化调整概述”

8.10. 修复 KAFKAREBALANCE 资源的问题

如果在创建 KafkaRebalance 资源或与 Cruise Control 交互时出现问题，则会在资源状态中报告错误，以及如何修复它。资源也进入 NotReady 状态。

要继续进行集群重新平衡操作，您必须修复 KafkaRebalance 资源本身或整个 Cruise Control 部署中的问题。问题可能包括以下几项：

- KafkaRebalance 资源中配置错误的参数。
- 在 Kafka Rebalance 资源中缺少用于指定 Kafka 集群的 The strimzi.io/cluster 标签。
- Cruise Control 服务器没有被部署，因为 Kafka 资源中缺少 cruise Control 属性。
- 无法访问 Cruise Control 服务器。

修复此问题后，您需要在 KafkaRebalance 资源中添加 刷新 注解。在"刷新"期间，要求来自 Cruise Control 服务器的新优化建议。

先决条件

- 您 已批准了一个优化建议。
- 重新平衡操作的 KafkaRebalance 自定义资源的状态是 NotReady。

流程

1. 从 KafkaRebalance 状态获取错误信息：

```
oc describe kafkarebalance rebalance-cr-name
```

2. 尝试解决 KafkaRebalance 资源中的问题。
3. 给 OpenShift 中的 KafkaRebalance 资源标注：

```
oc annotate kafkarebalance rebalance-cr-name strimzi.io/rebalance=refresh
```

4. 检查 KafkaRebalance 资源的状态：

```
oc describe kafkarebalance rebalance-cr-name
```

5. 等待状态更改为 PendingProposal，或直接变为 Proposal Ready。

其它资源

- 第 8.3 节 “优化调整概述”

第 9 章 使用 SERVICE REGISTRY 验证模式

您可以将 Red Hat Service Registry 与 AMQ Streams 一起使用。

服务注册表是一种数据存储，用于跨 API 和事件驱动的架构共享标准事件模式和 API 设计。您可以使用 Service Registry 将数据结构与客户端应用程序分离，并使用 REST 接口在运行时共享和管理数据类型和 API 描述。

Service Registry 存储用于序列化和取消序列化消息的架构，然后可以从客户端应用引用这些信息，以确保它们发送和接收的消息与这些模式兼容。Service Registry 为 Kafka 生产者和消费者应用程序提供 Kafka 客户端串行器/deserializers。Kafka 制作者应用程序使用串行程序对符合特定事件架构的消息进行编码。Kafka 使用者应用程序使用 deserializers，它会根据特定的架构 ID 验证消息是否使用正确的 schema 进行序列化。

您可以使应用程序使用 registry 中的 schema。这可确保使用一致的模式，并帮助防止运行时出现数据错误。

其它资源

- Service Registry 文档
- Service Registry 构建于 Apicurio Registry 开源社区项目基础上： Apicurio/apicurio-registry

第 10 章 分布式追踪

分布式追踪允许您跟踪分布式系统中应用程序之间的事务处理进度。在微服务架构中，跟踪服务之间的事务处理进度。跟踪数据可用于监控应用性能以及调查目标系统和最终用户应用的问题。

在 AMQ Streams 中，追踪有助于端到端跟踪信息：从源系统到 Kafka，然后从 Kafka 跟踪到目标系统和应用程序。它补充了在 Grafana 仪表板 以及组件日志记录器中查看的指标。

AMQ 流如何支持追踪

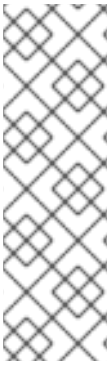
对追踪的支持内置在以下组件中：

- Kafka Connect（包括 Kafka Connect with Source2Image 支持）
- MirrorMaker
- MirrorMaker 2.0
- AMQ Streams Kafka Bridge

您可以使用自定义资源中的模板配置属性为这些组件启用和配置追踪。

要在 Kafka 制作者、使用者和 Kafka Streams API 应用程序中启用追踪，您可以使用 OpenTracing Apache Kafka Client Instrumentation 库（包括在 AMQ Streams 中）检测 应用程序 代码。当使用工具时，客户端会生成追踪数据；例如，当生成消息或写入偏移到日志中时。

trace 会根据抽样策略进行抽样，然后在 Jaeger 用户界面中视觉化。



注意

Kafka 代理不支持追踪。

为 AMQ Streams 以外的应用和系统设置追踪不在本章的讨论范围之内。要了解有关此主题的更多信息，请在 OpenTracing 文档中的 搜索"注入和提取"。

流程概述

要设置 AMQ Streams 的追踪，请按照以下步骤操作：

- 为客户端设置追踪：
  - 为 Kafka 客户端初始化 Jaeger tracer
- 使用 tracers 检测客户端：
  - 检测生产者和消费者的追踪
  - 检测 Kafka Streams 应用程序进行追踪
- 为 MirrorMaker、Kafka Connect 和 Kafka Bridge 设置追踪

先决条件

- Jaeger 后端组件部署到 OpenShift 集群中。有关部署说明，请参阅 Jaeger 部署文档。

10.1. OPENTRACING 和 JAEGER 概述

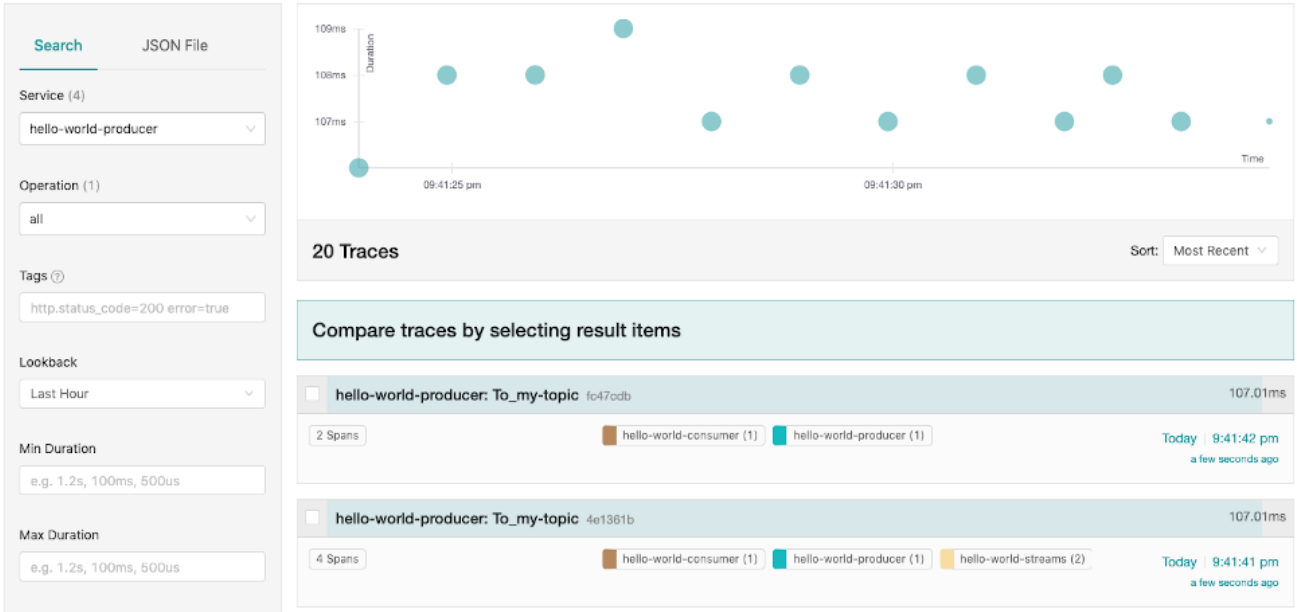
AMQ Streams 使用 OpenTracing 和 Jaeger 项目。

OpenTracing 是一种独立于追踪或监控系统的 API 规范。

- *OpenTracing API 用于检测应用程序代码*
- *工具的应用程序为分布式系统的个别交易生成追踪*
- *trace 由定义了特定工作单元的 span 组成*

Jaeger 是用于基于微服务的分布式系统的追踪系统。

- *Jaeger 实现 OpenTracing API，并为检测提供客户端库*
- *Jaeger 用户界面允许您查询、过滤和分析 trace 数据*



其它资源

- *OpenTracing*
- *Jaeger*

10.2. 为 KAFKA 客户端设置追踪

初始化 Jaeger tracer，以检测客户端应用程序进行分布式追踪。

10.2.1. 为 Kafka 客户端初始化 Jaeger tracer

使用一组 追踪环境变量 配置并初始化 Jaeger 追踪器。

流程

在每个客户端应用程序中：

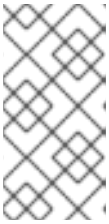
1. 将 Jaeger 的 Maven 依赖项添加到客户端应用程序的 pom.xml 文件中：

```
<dependency>
  <groupId>io.jaegertracing</groupId>
  <artifactId>jaeger-client</artifactId>
  <version>1.1.0.redhat-00002</version>
</dependency>
```

2. 使用 追踪环境变量 定义 Jaeger 追踪器的配置。

3. 从在第 2 步中定义的环境变量创建 Jaeger tracer：

```
Tracer tracer = Configuration.fromEnv().getTracer();
```



注意

有关初始化 Jaeger tracer 的其他方法，请参阅 Java OpenTracing 库 文档。

4. 将 Jaeger tracer 注册为全局追踪器：

```
GlobalTracer.register(tracer);
```

现在，Jaeger tracer 被初始化，供客户端应用程序使用。

10.2.2. 用于追踪的环境变量

在为 Kafka 客户端配置 Jaeger tracer 时，请使用这些环境变量。

注意

追踪环境变量是 Jaeger 项目的一部分，可能会有所变化。如需最新的环境变量，请参阅 Jaeger 文档。

属性	必需	描述
JAEGER_SERVICE_NAME	是	Jaeger tracer 服务的名称。
JAEGER_AGENT_HOST	否	用于通过用户数据报协议(UDP)与 **jaeger-agent** 通信的主机名。
JAEGER_AGENT_PORT	否	用于通过 UDP 与 **jaeger-agent** 通信的端口。
JAEGER_ENDPOINT	否	**trace** 端点。只有客户端应用将绕过 jaeger- **agent 并直接连接到 jaeger- collector** 时，才定义此变量。
JAEGER_AUTH_TOKEN	否	以 bearer 令牌形式发送到端点的身份验证令牌。
JAEGER_USER	否	如果使用基本身份验证，则发送到端点的用户名。
JAEGER_PASSWORD	否	如果使用基本身份验证，则发送到端点的密码。
JAEGER_PROPAGATION	否	用于传播 trace 上下文的格式的逗号分隔列表。默认为标准 Jaeger 格式。有效值为 **jaeger**、**b3** 和 **w3c**。
JAEGER_REPORTER_LOG_SPANS	否	指明报告者是否还应记录范围。
JAEGER_REPORTER_MAX_QUEUE_SIZE	否	报告器的最大队列大小。

属性	必需	描述
JAEGER_REPORTER_FLUSH_INTERVAL	否	报告器的清空间隔，以 ms 为单位。定义 Jaeger reporter flushes span 批处理的频率。
JAEGER_SAMPLER_TYPE	否	用于客户端 trace 的抽样策略： • 恒定 • Probabilistic • 速率限制 • remote（默认） 要对所有 trace 进行抽样，使用 Constant 抽样策略并将参数设为 1。 如需更多信息，请参阅 Jaeger 文档。
JAEGER_SAMPLER_PARAM	否	sampler 参数（数字）。
JAEGER_SAMPLER_MANAGER_HOST_PORT	否	如果选择了远程抽样策略，要使用的主机名和端口。
JAEGER_TAGS	否	以逗号分隔的 tracer 级别标签列表，这些标签添加到所有报告范围中。 该值也可以引用采用格式 **${envVarName:default}** 的环境变量。**:default** 是可选的，并且在无法找到环境变量时标识要使用的值。

其它资源

- *第 10.2.1 节 “为 Kafka 客户端初始化 Jaeger tracer”*

10.3. 使用 TRACERS 强制 KAFKA 客户端

检测 Kafka 制作者和消费者客户端，以及 Kafka Streams API 应用程序以实现分布式追踪。

10.3.1. 强制生产者和消费者进行追踪

使用 Decorator 模式或 Interceptors 来检测 Java 制作者和使用者应用代码进行追踪。

流程

在每个生产者和消费者应用程序的应用程序代码中：

1. 将 OpenTracing 的 Maven 依赖项添加到制作者或消费者的 pom.xml 文件中。

```
<dependency>
   <groupId>io.opentracing.contrib</groupId>
   <artifactId>opentracing-kafka-client</artifactId>
   <version>0.1.15.redhat-00001</version>
</dependency>
```

2. 使用 Decorator 模式或 Interceptors 检测您的客户端应用程序代码。

- 使用 Decorator 模式：

```
// Create an instance of the KafkaProducer:
KafkaProducer<Integer, String> producer = new KafkaProducer<>(senderProps);

// Create an instance of the TracingKafkaProducer:
TracingKafkaProducer<Integer, String> tracingProducer = new
TracingKafkaProducer<>(producer,
      tracer);

// Send:
tracingProducer.send(...);

// Create an instance of the KafkaConsumer:
KafkaConsumer<Integer, String> consumer = new KafkaConsumer<>
(consumerProps);

// Create an instance of the TracingKafkaConsumer:
TracingKafkaConsumer<Integer, String> tracingConsumer = new
TracingKafkaConsumer<>(consumer,
      tracer);

// Subscribe:
tracingConsumer.subscribe(Collections.singletonList("messages"));

// Get messages:
ConsumerRecords<Integer, String> records = tracingConsumer.poll(1000);

// Retrieve SpanContext from polled record (consumer side):
```

```
ConsumerRecord<Integer, String> record = ...
SpanContext spanContext =
TracingKafkaUtils.extractSpanContext(record.headers(), tracer);
```

- 使用 Interceptors:

```
// Register the tracer with GlobalTracer:
GlobalTracer.register(tracer);

// Add the TracingProducerInterceptor to the sender properties:
senderProps.put(ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,
        TracingProducerInterceptor.class.getName());

// Create an instance of the KafkaProducer:
KafkaProducer<Integer, String> producer = new KafkaProducer<>(senderProps);

// Send:
producer.send(...);

// Add the TracingConsumerInterceptor to the consumer properties:
consumerProps.put(ConsumerConfig.INTERCEPTOR_CLASSES_CONFIG,
        TracingConsumerInterceptor.class.getName());

// Create an instance of the KafkaConsumer:
KafkaConsumer<Integer, String> consumer = new KafkaConsumer<>
(consumerProps);

// Subscribe:
consumer.subscribe(Collections.singletonList("messages"));

// Get messages:
ConsumerRecords<Integer, String> records = consumer.poll(1000);

// Retrieve the SpanContext from a polled message (consumer side):
ConsumerRecord<Integer, String> record = ...
SpanContext spanContext =
TracingKafkaUtils.extractSpanContext(record.headers(), tracer);
```

10.3.1.1. Decorator 模式中的自定义范围名称

span 是 Jaeger 中的逻辑工作单元，具有操作名称、开始时间和持续时间。

要使用 Decorator 模式来检测您的制作者和消费者应用程序，请在创建 TracingKafkaProducer 和 TracingKafka Consumer 对象时传递 BiFunction 对象作为额外参数来定义自定义 span 名称。OpenTracing Apache Kafka Client Instrumentation 库包含多个内置范围名称。

示例：使用自定义 span 名称在 Decorator 模式中检测客户端应用程序代码

```
// Create a BiFunction for the KafkaProducer that operates on (String operationName,
ProducerRecord consumerRecord) and returns a String to be used as the name:

BiFunction<String, ProducerRecord, String> producerSpanNameProvider =
    (operationName, producerRecord) -> "CUSTOM_PRODUCER_NAME";

// Create an instance of the KafkaProducer:
KafkaProducer<Integer, String> producer = new KafkaProducer<>(senderProps);

// Create an instance of the TracingKafkaProducer
TracingKafkaProducer<Integer, String> tracingProducer = new TracingKafkaProducer<>
(producer,
        tracer,
        producerSpanNameProvider);

// Spans created by the tracingProducer will now have "CUSTOM_PRODUCER_NAME" as the
span name.

// Create a BiFunction for the KafkaConsumer that operates on (String operationName,
ConsumerRecord consumerRecord) and returns a String to be used as the name:

BiFunction<String, ConsumerRecord, String> consumerSpanNameProvider =
    (operationName, consumerRecord) -> operationName.toUpperCase();

// Create an instance of the KafkaConsumer:
KafkaConsumer<Integer, String> consumer = new KafkaConsumer<>(consumerProps);

// Create an instance of the TracingKafkaConsumer, passing in the
consumerSpanNameProvider BiFunction:

TracingKafkaConsumer<Integer, String> tracingConsumer = new TracingKafkaConsumer<>
(consumer,
        tracer,
        consumerSpanNameProvider);

// Spans created by the tracingConsumer will have the operation name as the span name, in
upper-case.
// "receive" -> "RECEIVE"
```

10.3.1.2. 内置范围名称

在定义自定义 span 名称时，您可以在 ClientSpanNameProvider 类中使用以下 BiFunctions。如果没有指定 spanNameProvider，则使用 CONSUMER_OPERATION_NAME 和 PRODUCER_OPERATION_NAME。

BiFunction	描述
CONSUMER_OPERATION_NAME, PRODUCER_OPERATION_NAME	将 **operationName** 返回为 span name: "receive" for 使用者，并为生产者返回"send"。
CONSUMER_PREFIXED_OPERATION_NAME（String 前缀），PRODUCER_PREFIXED_OPERATION_NAME（String 前缀）	返回 **prefix** 和 **operationName 的 String 连接**。
CONSUMER_TOPIC, PRODUCER_TOPIC	返回消息发送到或检索到的主题的名称，格式为 **（record.topic（））**。
PREFIXED_CONSUMER_TOPIC（String 前缀）, PREFIXED_PRODUCER_TOPIC（String 前缀）	返回 **前缀** 的字符串串联和主题名称，格式为 **（record.topic（））**。
CONSUMER_OPERATION_NAME_TOPIC, PRODUCER_OPERATION_NAME_TOPIC	返回操作名称和主题名称 **："operationName - record.topic（）"。**
CONSUMER_PREFIXED_OPERATION_NAME_TOPIC（String 前缀）、PRODUCER_PREFIXED_OPERATION_NAME_TOPIC（String 前缀）	返回 **前缀** 的字符串串联和 **"operationName - record.topic（）"。**

10.3.2. 用于追踪的 Kafka Streams 应用程序

本节论述了如何检测 Kafka Streams API 应用程序进行分布式追踪。

流程

在每个 Kafka Streams API 应用程序中：

1. *将 opentracing-kafka-streams 依赖项添加到 Kafka Streams API 应用程序的 pom.xml 文件中：*

```
<dependency>
  <groupId>io.opentracing.contrib</groupId>
  <artifactId>opentracing-kafka-streams</artifactId>
  <version>0.1.15.redhat-00001</version>
</dependency>
```

2. *创建 TracingKafkaClientSupplier 供应商 界面的实例：*

```
KafkaClientSupplier supplier = new TracingKafkaClientSupplier(tracer);
```

3. 为 KafkaStreams 提供供应商接口：

```
KafkaStreams streams = new KafkaStreams(builder.build(), new
StreamsConfig(config), supplier);
streams.start();
```

10.4. 为 MIRRORMAKER、KAFKA CONNECT 和 KAFKA BRIDGE 设置追踪

MirrorMaker、MirrorMaker 2.0、Kafka Connect（包括使用 Source2Image 支持的 Kafka Connect）和 AMQ Streams Kafka Bridge 支持分布式追踪。

MirrorMaker 和 MirrorMaker 2.0 中的追踪

对于 MirrorMaker 和 MirrorMaker 2.0，信息会从源集群追踪到目标集群。追踪数据记录进入和离开 MirrorMaker 或 MirrorMaker 2.0 组件的信息。

Kafka Connect 中的追踪

只有 Kafka Connect 本身生成并使用的消息才会被跟踪。要追踪在 Kafka Connect 和外部系统间发送的信息，您必须在连接器中为这些系统配置追踪。如需更多信息，请参阅 第 2.2.1 节 “配置 Kafka 连接”。

在 Kafka Bridge 中追踪

Kafka Bridge 生成并使用的消息会被跟踪。还会追踪从客户端应用程序发送和接收消息的传入 HTTP 请求。要进行端到端追踪，您必须在 HTTP 客户端中配置追踪。

10.4.1. 在 MirrorMaker、Kafka Connect 和 Kafka Bridge 资源中启用追踪

更新 KafkaMirrorMaker、KafkaMirrorMaker2、KafkaConnect S2I 和 KafkaBridge 自定义资源的配置，以为每个资源指定和配置 Jaeger tracer 服务。更新 OpenShift 集群中启用追踪的资源会触发两个事件：

- 在 MirrorMaker、MirrorMaker 2.0、Kafka Connect 或 AMQ Streams Kafka Bridge 中的集成消费者和生产者中更新拦截器类。
- 对于 MirrorMaker、MirrorMaker 2.0 和 Kafka Connect，追踪代理会根据资源中定义的追踪配置初始化 Jaeger tracer。

- *对于 Kafka Bridge，基于资源中定义的追踪配置的 Jaeger tracer 由 Kafka Bridge 本身初始化。*

流程

为每个 KafkaMirrorMaker、KafkaMirrorMaker2、KafkaConnect、KafkaConnect S2I 和 KafkaBridge 资源执行这些步骤。

1. 在 spec.template 属性中配置 Jaeger tracer 服务。例如：

Kafka Connect 的 Jaeger tracer 配置

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  template:
    connectContainer: 1
      env:
        - name: JAEGER_SERVICE_NAME
          value: my-jaeger-service
        - name: JAEGER_AGENT_HOST
          value: jaeger-agent-name
        - name: JAEGER_AGENT_PORT
          value: "6831"
  tracing: 2
    type: jaeger
  #...
```

MirrorMaker 的 Jaeger tracer 配置

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker
metadata:
  name: my-mirror-maker
spec:
```

```
  #...
  template:
    mirrorMakerContainer:
      env:
        - name: JAEGER_SERVICE_NAME
          value: my-jaeger-service
        - name: JAEGER_AGENT_HOST
          value: jaeger-agent-name
        - name: JAEGER_AGENT_PORT
          value: "6831"
  tracing:
    type: jaeger
#...
```

MirrorMaker 2.0 的 Jaeger tracer 配置

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mm2-cluster
spec:
  #...
  template:
    connectContainer:
      env:
        - name: JAEGER_SERVICE_NAME
          value: my-jaeger-service
        - name: JAEGER_AGENT_HOST
          value: jaeger-agent-name
        - name: JAEGER_AGENT_PORT
          value: "6831"
  tracing:
    type: jaeger
#...
```

Kafka Bridge 的 Jaeger tracer 配置

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
```

```
#...
template:
  bridgeContainer:
    env:
      - name: JAEGER_SERVICE_NAME
        value: my-jaeger-service
      - name: JAEGER_AGENT_HOST
        value: jaeger-agent-name
      - name: JAEGER_AGENT_PORT
        value: "6831"
tracing:
  type: jaeger
#...
```

1 使用 追踪环境变量 作为模板配置属性。

2 将 spec.tracing.type 属性设置为 jaeger。

2. 创建或更新资源：

```
oc apply -f your-file
```

其它资源

- 第 13.2.40 节 “ContainerTemplate 架构参考”
- 第 2.6 节 “自定义 OpenShift 资源”

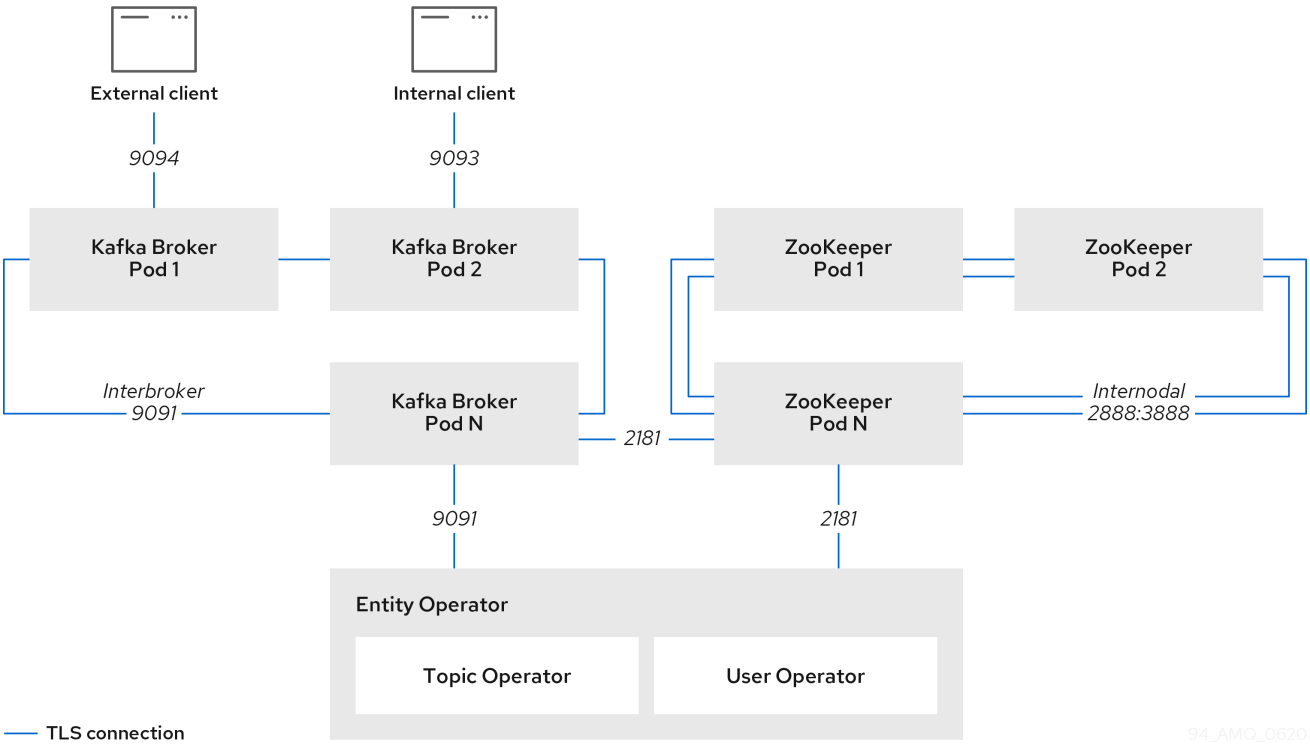
第 11 章 管理 TLS 证书

AMQ Streams 支持使用 TLS 协议在 Kafka 和 AMQ Streams 组件之间进行加密通信。Kafka 代理（interbroker 通信）、ZooKeeper 节点之间的通信（交互通信）以及这些组件和 AMQ Streams 操作器之间的通信始终会被加密。Kafka 客户端和 Kafka 代理之间的通信会根据集群的配置方式加密。对于 Kafka 和 AMQ Streams 组件，也使用 TLS 证书进行身份验证。

Cluster Operator 自动设置并更新 TLS 证书，以在集群中启用加密和身份验证。如果要在 Kafka 代理和客户端间启用加密或 TLS 身份验证，它也会设置其他 TLS 证书。用户提供的证书不会被续订。

您可以为启用了 TLS 加密的 TLS 监听程序或者外部监听程序提供自己的服务器证书，称为 Kafka 侦听程序证书。如需更多信息，请参阅 第 11.7 节 “Kafka 侦听程序证书”。

图 11.1. TLS 安全通信的示例架构



11.1. 证书颁发机构

为了支持加密，每个 AMQ Streams 组件都需要自己的私钥和公钥证书。所有组件证书都由名为 集群 CA 的内部证书颁发机构(CA)签名。

同样，使用 TLS 客户端身份验证连接到 AMQ Streams 的每个 Kafka 客户端应用程序都需要提供私钥和证书。第二个内部 CA（名为 clients CA ）用于为 Kafka 客户端签署证书。

11.1.1. CA 证书

集群 CA 和客户端 CA 都具有自签名证书。

Kafka 代理配置为信任由集群 CA 或客户端 CA 签名的证书。客户端不需要连接的组件，如 ZooKeeper，只信任集群 CA 签名的证书。除非禁用外部监听器的 TLS 加密，否则客户端应用必须信任集群 CA 签名的证书。对于执行 mutual TLS 身份验证 的客户端应用，也是如此。

默认情况下，AMQ Streams 会自动生成和更新集群 CA 或客户端 CA 发布的 CA 证书。您可以在 Kafka.spec. clusterCa 和 Kafka.spec.clientsCa 对象中配置这些 CA 证书的管理。用户提供的证书不会被续订。

您可以为集群 CA 或客户端 CA 提供自己的 CA 证书。如需更多信息，请参阅 第 11.1.2 节 “安装您自己的 CA 证书”。如果您提供自己的证书，则必须在需要时手动更新它们。

11.1.2. 安装您自己的 CA 证书

此流程描述了如何安装您自己的 CA 证书和密钥，而不是使用 Cluster Operator 生成的 CA 证书和私钥。

您可以使用此流程安装自己的集群或客户端 CA 证书。

流程描述了 PEM 格式的 CA 证书续订。您还可以使用 PKCS #12 格式的证书。

先决条件

- Cluster Operator 正在运行。
- Kafka 集群尚未部署。
- 您自己的 X.509 证书和密钥，采用 PEM 格式用于集群 CA 或客户端 CA。
  - 如果要使用不是 Root CA 的集群或客户端 CA，您必须在证书文件中包含整个链。链应该按照以下顺序排列：

1. *集群或客户端 CA*

2. *一个或多个中间 CA*

3. *根 CA*

- *链中的所有 CA 都应配置为 X509v3 基本约束中的 CA。*

流程

1. *将您的 CA 证书放在对应的 Secret 中。*

a. *删除现有的 secret：*

```
oc delete secret CA-CERTIFICATE-SECRET
```

ca-CERTIFICATE-SECRET 是 Secret 的名称，它是集群 CA 证书的CLUSTER-NAME-cluster-ca-cert，用于客户端 CA 证书的 CLUSTER-NAME-clients-ca-cert。

忽略任何"Not Exists"错误。

b. *创建并标记新 secret*

```
oc create secret generic CA-CERTIFICATE-SECRET --from-file=ca.crt=CA-CERTIFICATE-FILENAME
```

2. *将您的 CA 密钥放在对应的 Secret 中。*

a. *删除现有的 secret：*

```
oc delete secret CA-KEY-SECRET
```

ca-KEY-SECRET 是 CA 密钥的名称，这是 用于集群 CA 密钥的 CLUSTER-NAME -

cluster-ca，用于客户端 CA 密钥。

b. *创建新 secret：*

```
oc create secret generic CA-KEY-SECRET --from-file=ca.key=CA-KEY-SECRET-FILENAME
```

3. *使用 labels strimzi.io/kind=Kafka 和 strimzi.io/cluster=CLUSTER-NAME 标记 secret：*

```
oc label secret CA-CERTIFICATE-SECRET strimzi.io/kind=Kafka strimzi.io/cluster=CLUSTER-NAME
oc label secret CA-KEY-SECRET strimzi.io/kind=Kafka strimzi.io/cluster=CLUSTER-NAME
```

4. *为集群创建 Kafka 资源，将 Kafka.spec.clusterCa 或 Kafka.spec.clientsCa 对象配置为不使用生成的 CA：*

将集群 CA 配置为使用您自己提供的证书的片段 Kafka 资源示例

```
kind: Kafka
version: kafka.strimzi.io/v1beta2
spec:
  # ...
  clusterCa:
    generateCertificateAuthority: false
```

其它资源

- *要续订您之前安装的 CA 证书，请参阅 第 11.3.5 节 “续订您自己的 CA 证书”。*
- *第 11.7.1 节 “提供您自己的 Kafka 侦听程序证书”.*

11.2. SECRETS

AMQ Streams 使用 Secret 来存储 Kafka 集群组件和客户端的私钥和证书。secret 用于在 Kafka 代理

之间以及代理与客户端之间建立 TLS 加密连接。它们也用于 mutual TLS 身份验证。

- Cluster Secret 包含用于为 Kafka 代理证书签名的集群 CA 证书，供连接的客户端用于与 Kafka 集群建立 TLS 加密连接以验证代理身份。

- Client Secret 包含用户用来签署自己的客户端证书的客户端 CA 证书，以允许对 Kafka 集群进行相互身份验证。代理通过客户端 CA 证书本身验证客户端身份。

- 用户 Secret 包含私钥和证书，在创建新用户时由客户端 CA 证书生成和签名。密钥和证书用于访问集群时的身份验证和授权。

secret 以 PEM 和 PKCS #12 格式提供私钥和证书。使用 PEM 格式的私钥和证书意味着用户必须从 Secret 中获取，并生成对应的信任存储（或密钥存储），以便在其 Java 应用程序中使用。PKCS #12 存储提供了一个可直接使用的信任存储（或密钥存储）。

所有键的大小都是 2048 位。

11.2.1. PKCS #12 存储

PKCS #12 定义了一个归档文件格式(.p12)，用于将加密对象存储在单个文件中并设有密码保护。您可以使用 PKCS #12 在一个位置管理证书和密钥。

每个 Secret 都包含特定于 PKCS #12 的字段。

- .p12 字段包含证书和密钥。

- .password 字段是保护存档的密码。

11.2.2. 集群 CA Secret

下表描述了由 Cluster Operator 在 Kafka 集群中管理的 Cluster Secret。

客户端只需要使用 <cluster> -cluster-ca-cert Secret。所有其他 Secret 仅需要由 AMQ Streams 组件访问。如果需要，您可以使用基于 OpenShift 角色的访问控制来强制实施此操作。

表 11.1. <cluster>-cluster-ca Secret 中的 字段

项	描述
ca.key	集群 CA 的当前私钥。

表 11.2. <cluster>-cluster-ca-cert Secret 中的 字段

项	描述
ca.p12	PKCS #12 存档文件用于存储证书和密钥。
ca.password	用于保护 PKCS #12 归档文件的密码。
ca.crt	集群 CA 的当前证书。

注意

<cluster>-cluster-ca-cert 中的 CA 证书必须被 Kafka 客户端应用程序信任，以便在通过 TLS 连接到 Kafka 代理时验证 Kafka 代理证书。

表 11.3. <cluster>-kafka-brokers Secret 中的 字段

项	描述
***<cluster>*-kafka-*<num>*.p12**	PKCS #12 存档文件用于存储证书和密钥。
***<cluster>*-kafka-*<num>*.password**	用于保护 PKCS #12 归档文件的密码。
***<cluster>*-kafka-*<num>*.crt**	Kafka 代理 pod *<num>* 的证书。由 ***<cluster> -cluster-ca 中的*** 当前或以前的集群 CA 私钥签名。
***<cluster>*-kafka-*<num>*.key**	Kafka 代理 pod ***<num>*** 的私钥。

表 11.4. <cluster>-zookeeper-nodes Secret 中的 字段

项	描述
***<cluster>*-zookeeper-*<num>*.p12**	PKCS #12 存档文件用于存储证书和密钥。
***<cluster>*-zookeeper-*<num>*.password**	用于保护 PKCS #12 归档文件的密码。

项	描述
<cluster>-zookeeper-<num>.crt	ZooKeeper 节点 <num> 的证书。由 **<cluster> -cluster-ca 中的** 当前或以前的集群 CA 私钥签名。
<cluster>-zookeeper-<num>.key	ZooKeeper pod **<num>** 的私钥。

表 11.5. <cluster>-entity-operator-certs Secret 中的 字段

项	描述
entity-operator_.p12	PKCS #12 存档文件用于存储证书和密钥。
entity-operator_.password	用于保护 PKCS #12 归档文件的密码。
entity-operator_.crt	Entity Operator 和 Kafka 或 ZooKeeper 之间的 TLS 通信证书。由 **<cluster> -cluster-ca 中的** 当前或以前的集群 CA 私钥签名。
entity-operator.key	Entity Operator 和 Kafka 或 ZooKeeper 之间的 TLS 通信的私钥。

11.2.3. 客户端 CA Secret

表 11.6. 在 <cluster>中由 Cluster Operator 管理的客户端 CA Secret

Secret 名称	Secret 中的字段	描述
<cluster>-clients-ca	**ca.key**	客户端 CA 的当前私钥。
<cluster>-clients-ca-cert	**ca.p12**	PKCS #12 存档文件用于存储证书和密钥。
	ca.password	用于保护 PKCS #12 归档文件的密码。
	ca.crt	客户端 CA 的当前证书。

<cluster>-clients-ca-cert 中的证书是 Kafka 代理信任的证书。

注意

<cluster>-clients-ca 用于签署客户端应用程序的证书。如果您打算在不使用 User Operator 的情况下发布应用程序证书，则需要对 AMQ Streams 组件和管理员访问权限进行访问。如果需要，您可以使用基于 OpenShift 角色的访问控制来强制实施此操作。

11.2.4. 在 Secret 中添加标签和注解

通过在 Kafka 自定义资源中配置 clusterCaCert template 属性，您可以在 Cluster Operator 创建的 Cluster CA Secret 中添加自定义标签和注解。标签和注解可用于识别对象和添加上下文信息。您可以在 AMQ Streams 自定义资源中配置模板属性。

将标签和注解添加到 Secret 的模板自定义示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    template:
      clusterCaCert:
        metadata:
          labels:
            label1: value1
            label2: value2
          annotations:
            annotation1: value1
            annotation2: value2
    # ...
```

有关配置模板属性的详情请参考 第 2.6 节 “自定义 OpenShift 资源”。

11.2.5. 在 CA Secret 中禁用 ownerReference

默认情况下，Cluster 和 Client CA Secret 是使用设置为 Kafka 自定义资源的 ownerReference 属性创建的。这意味着，当删除 Kafka 自定义资源时，OpenShift 也会删除（收集垃圾回收） CA secret。

如果要为新集群重复使用 CA，您可以通过在 Kafka 配置中将 Cluster 和 Client CA Secrets 的 generateSecretOwnerReference 属性设置为 false 来禁用 owner Reference。当禁用 ownerReference 时，当对应的 Kafka 自定义资源被删除时，OpenShift 不会删除 CA Secret。

为集群和客户端 CA 禁用 ownerReference 的 Kafka 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
# ...
spec:
# ...
  clusterCa:
    generateSecretOwnerReference: false
  clientsCa:
    generateSecretOwnerReference: false
# ...
```

其它资源

- *certificateAuthority schema 参考*

11.2.6. 用户 Secret

表 11.7. 由 User Operator 管理的 secret

Secret 名称	Secret 中的字段	描述
<user>	**user.p12**	PKCS #12 存档文件用于存储证书和密钥。
	user.password	用于保护 PKCS #12 归档文件的密码。
	user.crt	用户的证书，由客户端 CA 签名
	user.key	用户的私钥

11.3. 证书续订和有效期周期

集群 CA 和客户端 CA 证书仅在有限时间内有效，称为有效期。这通常定义为生成证书起的天数。

对于 Cluster Operator 自动创建的 CA 证书，您可以配置以下有效期：

- *Kafka.spec.clusterCa.validityDays中的集群 CA 证书*
- *Kafka.spec.clientsCa.validityDays中的客户端 CA 证书*

两个证书的默认有效期为 365 天。手动安装的 CA 证书应该定义自己的有效期。

当 CA 证书过期时，仍信任该证书的组件和客户端将不接受来自证书由 CA 私钥签名的对等点的 TLS 连接。组件和客户端需要信任 新的 CA 证书。

为了能在不丢失服务的情况下续订 CA 证书，Cluster Operator 将在旧 CA 证书过期前启动证书续订。

您可以配置 Cluster Operator 创建的证书的续订周期：

- Kafka.spec.clusterCa.renewalDays中的集群 CA 证书
- Kafka.spec.clientsCa.renewalDays中的客户端 CA 证书

两个证书的默认续订期限为 30 天。

从当前证书的到期日期起，续订期向后测量。

针对续订期的有效周期

```
Not Before                                Not After
   |                                          |
   |<---------------- validityDays ---------------->|
                          <--- renewalDays --->|
```

要在创建 Kafka 集群后更改有效期和续订周期，您需要配置并应用 Kafka 自定义资源，并 手动更新 CA 证书。如果您不手动续订证书，在下次自动更新证书时将使用新的句点。

证书的有效性和续订周期的 Kafka 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
# ...
spec:
# ...
  clusterCa:
    renewalDays: 30
    validityDays: 365
    generateCertificateAuthority: true
  clientsCa:
    renewalDays: 30
    validityDays: 365
    generateCertificateAuthority: true
# ...
```

在续订期间，Cluster Operator 的行为取决于证书生成属性的设置、generate Authority 和 generateCertificateAuthority。

true

如果属性设为 true，则 CA 证书由 Cluster Operator 自动生成，并在续订期间自动更新。

false

如果属性设为 false，则 Cluster Operator 不会生成 CA 证书。如果要 安装自己的证书，请使用这个选项。

11.3.1. 使用自动生成的 CA 证书的续订过程

Cluster Operator 执行以下流程来更新 CA 证书：

1. 生成新的 CA 证书，但保留现有密钥。新证书将旧证书替换为对应 Secret 中的名称 ca.crt。

2. 生成新的客户端证书（用于 ZooKeeper 节点、Kafka 代理和 Entity Operator）。这并非绝对必要，因为签名密钥尚未更改，但它保持客户端证书的有效性期限与 CA 证书同步。

3. 重启 ZooKeeper 节点，以便它们信任新的 CA 证书并使用新的客户端证书。

4. 重启 Kafka 代理，以便它们信任新的 CA 证书并使用新的客户端证书。

5. 重启主题和用户 Operator，以便它们信任新的 CA 证书并使用新的客户端证书。

11.3.2. 客户端证书续订

Cluster Operator 不知道使用 Kafka 集群的客户端应用程序。

当连接到集群并确保它们正常工作时，客户端应用程序必须：

- 信任 <cluster> -cluster-ca-cert Secret 中发布的集群 CA 证书。
- 使用其 <user-name> Secret 中发布的凭证来连接到集群。

  用户 Secret 以 PEM 和 PKCS #12 格式提供凭证，或者在使用 SCRAM-SHA 身份验证时可以提供密码。User Operator 在创建用户时创建用户凭据。

您必须确保在证书续订后客户继续工作。续订过程取决于如何配置客户端。

如果要手动置备客户端证书和密钥，您必须生成新的客户端证书，并确保客户端在续订期内使用这些新证书。如果续订周期结束后未能这样做，则可能导致客户应用程序无法连接到集群。

注意

对于在同一 OpenShift 集群和命名空间中运行的工作负载，可以将机密挂载为卷，以便客户端 Pod 从 Secret 的当前状态构建其密钥存储和信任存储。有关此流程的详情，请参阅配置内部客户端以信任集群 CA。

11.3.3. 手动续订 Cluster Operator 生成的 CA 证书

Cluster Operator 生成的集群和客户端 CA 证书会在各自的证书续订期开始时自动更新。但是，您可以在证书续订周期开始前，使用 strimzi.io/force-renew 注解手动续订这些证书的一个或多个证书。出于安全原因 或更改了证书的续订期或有效期，您可以这样做。

更新的证书使用与旧证书相同的私钥。

注意

如果您使用自己的 CA 证书，则无法使用 force-renew 注解。相反，请按照 更新您自己的 CA 证书 的步骤进行操作。

先决条件

- Cluster Operator 正在运行。
- 安装 CA 证书和私钥的 Kafka 集群。

流程

1. 将 strimzi.io/force-renew 注解应用到包含您要续订的 CA 证书的 Secret。

表 11.8. 强制续订证书的 Secret 注解

证书	Secret	annotate 命令
集群 CA	*KAFKA-CLUSTER-NAME*-cluster-ca-cert	**oc annotate secret *KAFKA-CLUSTER-NAME*-cluster-ca-cert strimzi.io/force-renew=true**
客户端 CA	*KAFKA-CLUSTER-NAME*-clients-ca-cert	**oc annotate secret *KAFKA-CLUSTER-NAME*-clients-ca-cert strimzi.io/force-renew=true**

在下一次协调中，Cluster Operator 会为您注解的 Secret 生成一个新的 CA 证书。如果配置了维护时间窗，Cluster Operator 将在下一个维护时间窗内第一次协调时生成新的 CA 证书。

客户端应用程序必须重新载入由 Cluster Operator 更新的集群和客户端 CA 证书。

2. *检查 CA 证书有效的周期：*

   例如，使用 openssl 命令：

   ```
   oc get secret CA-CERTIFICATE-SECRET -o 'jsonpath={.data.CA-CERTIFICATE}' |
   base64 -d | openssl x509 -subject -issuer -startdate -enddate -noout
   ```

   ca-CERTIFICATE-SECRET 是 Secret 的名称，这是用于集群 CA 证书的 KAFKA-CLUSTER-NAME-cluster-ca-cert 和用于客户端 CA 证书的 KAFKA-CLUSTER-NAME-clients-ca-cert。

   ca-CERTIFICATE 是 CA 证书的名称，如 jsonpath={.data.ca\.crt}。

   该命令会返回 notBefore 和 notAfter 日期，这是 CA 证书的有效周期。

   例如，对于集群 CA 证书：

   ```
   subject=O = io.strimzi, CN = cluster-ca v0
   issuer=O = io.strimzi, CN = cluster-ca v0
   notBefore=Jun 30 09:43:54 2020 GMT
   notAfter=Jun 30 09:43:54 2021 GMT
   ```

3. *从 Secret 删除旧证书。*

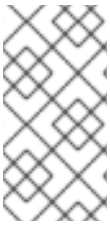
   当组件使用新证书时，旧证书可能仍处于活跃状态。删除旧证书以消除任何潜在的安全风险。

其它资源

- *第 11.2 节 “Secrets”*
- *第 2.1.5 节 “用于滚动更新的维护时间窗”*
- *第 13.2.49 节 “certificateAuthority schema 参考”*

11.3.4. 替换 Cluster Operator 生成的 CA 证书使用的私钥

您可以替换由集群 CA 使用的私钥和 Cluster Operator 生成的客户端 CA 证书。替换私钥时，Cluster Operator 会为新私钥生成一个新的 CA 证书。



注意

如果您使用自己的 CA 证书，则无法使用 force-replace 注解。相反，请按照 *更新您自己的 CA 证书* 的步骤进行操作。

先决条件

- Cluster Operator 正在运行。
- 安装 CA 证书和私钥的 Kafka 集群。

流程

- 将 strimzi.io/force-replace 注解应用到包含您要续订的私钥的 Secret。

表 11.9. 替换私钥的命令

的私钥	Secret	annotate 命令
集群 CA	*CLUSTER-NAME*-cluster-ca	**oc annotate secret *CLUSTER-NAME*-cluster-ca strimzi.io/force-replace=true**
客户端 CA	*CLUSTER-NAME*-clients-ca	**oc annotate secret *CLUSTER-NAME*-clients-ca strimzi.io/force-replace=true**

在下一次协调中，Cluster Operator 将：

- 为注解的 Secret 生成一个新的私钥

- 生成新的 CA 证书

如果配置了维护时间窗，Cluster Operator 将在下一个维护时间窗内第一次协调时生成新的私钥和 CA 证书。

客户端应用程序必须重新载入由 Cluster Operator 更新的集群和客户端 CA 证书。

其它资源

- 第 11.2 节 “Secrets”
- 第 2.1.5 节 “用于滚动更新的维护时间窗”

11.3.5. 续订您自己的 CA 证书

此流程描述了如何续订您自己安装的 CA 证书和密钥，而不是使用 Cluster Operator 生成的证书。

如果您使用自己的证书，Cluster Operator 不会自动续订它们。因此，您必须在证书的续订期内遵循此步骤，以替换即将过期的 CA 证书。

该程序描述了 PEM 格式的 CA 证书的续订。您还可以使用 PKCS #12 格式的证书。

先决条件

- Cluster Operator 正在运行。
- 已安装您自己的 CA 证书和私钥。
- 您有新的集群和客户端 X.509 证书和密钥，格式为 PEM。

它们可使用 openssl 命令生成，例如：

```
openssl req -x509 -new -days NUMBER-OF-DAYS-VALID --nodes -out ca.crt -keyout ca.key
```

流程

1. 在 Secret 中检查当前 CA 证书的详情：

```
oc describe secret CA-CERTIFICATE-SECRET
```

ca-CERTIFICATE-SECRET 是 Secret 的名称，这是用于集群 CA 证书的 KAFKA-CLUSTER-NAME-cluster-ca-cert 和用于客户端 CA 证书的 KAFKA-CLUSTER-NAME-clients-ca-cert。

2. 创建一个目录，使其包含 secret 中的现有 CA 证书。

```
mkdir new-ca-cert-secret
cd new-ca-cert-secret
```

3. 获取您要续订的每个 CA 证书的 secret：

```
oc get secret CA-CERTIFICATE-SECRET -o 'jsonpath={.data.CA-CERTIFICATE}' |
base64 -d > CA-CERTIFICATE
```

将 CA-CERTIFICATE 替换为每个 CA 证书的名称。

4. 将旧的 ca.crt 文件重命名为 ca-DATE. crt，其中 DATE 是证书到期日期，格式为 YEAR -MONTH-DAYTHOUR-MINUTE-SECONDZ。

例如 ca-2018-09-27T17-32-00Z.crt。

```
mv ca.crt ca-$(date -u -d$(openssl x509 -enddate -noout -in ca.crt | sed 's/.*=//') +'%Y-
%m-%dT%H-%M-%SZ').crt
```

5. 将您的新 CA 证书复制到该目录，并将其命名为 ca.crt ：

```
cp PATH-TO-NEW-CERTIFICATE ca.crt
```

6. 将您的 CA 证书放在对应的 Secret 中。

   a. 删除现有的 secret：

   ```
   oc delete secret CA-CERTIFICATE-SECRET
   ```

   CA-CERTIFICATE-SECRET 是 Secret 的名称，在第一步中返回。

   忽略任何"Not Exists"错误。

   b. 重新创建 secret：

   ```
   oc create secret generic CA-CERTIFICATE-SECRET --from-file=.
   ```

7. 删除您创建的目录：

   ```
   cd ..
   rm -r new-ca-cert-secret
   ```

8. 将您的 CA 密钥放在对应的 Secret 中。

   a. 删除现有的 secret：

   ```
   oc delete secret CA-KEY-SECRET
   ```

   ca-KEY-SECRET 是 CA 密钥的名称，这是用于集群 CA 密钥的 KAFKA-CLUSTER-NAME-cluster-ca，用于客户端 CA 密钥的 KAFKA-CLUSTER-NAME-clients-ca。

   b. 使用新 CA 密钥重新创建 secret：

   ```
   oc create secret generic CA-KEY-SECRET --from-file=ca.key=CA-KEY-SECRET-FILENAME
   ```

9. 使用 labels strimzi.io/kind=Kafka 和 strimzi.io/cluster=KAFKA-CLUSTER-NAME 标记

secret：

```
oc label secret CA-CERTIFICATE-SECRET strimzi.io/kind=Kafka
strimzi.io/cluster=KAFKA-CLUSTER-NAME
oc label secret CA-KEY-SECRET strimzi.io/kind=Kafka strimzi.io/cluster=KAFKA-
CLUSTER-NAME
```

11.4. TLS 连接

11.4.1. zookeeper 通讯

所有端口上的 ZooKeeper 节点以及客户端和 ZooKeeper 之间的通信使用 TLS 加密。

Kafka 代理和 ZooKeeper 节点之间的通信也被加密。

11.4.2. Kafka 间通信

Kafka 代理之间的通信始终使用 TLS 加密。

除非启用了 ControlPlaneListener 功能门，否则所有代理间通信都通过端口 9091 上的内部监听程序进行。如果启用功能门，来自 control plane 的流量会通过端口 9090 上的内部 control plane 侦听程序。数据平面的流量继续使用端口 9091 上的现有内部监听程序。

Kafka 客户端无法使用这些内部监听程序。

11.4.3. 主题和用户 Operator

所有 Operator 使用加密来与 Kafka 和 ZooKeeper 进行通信。在主题和用户 Operator 中，与 ZooKeeper 通信时使用 TLS sidecar。

11.4.4. Sything Control

整合控制使用加密来与 Kafka 和 ZooKeeper 进行通信。与 ZooKeeper 通信时使用 TLS sidecar。

11.4.5. Kafka 客户端连接

Kafka 代理和客户端之间的加密或未加密通信是使用 spec.kafka.listeners 的 tls 属性配置的。

11.5. 配置内部客户端以信任集群 CA

此流程描述了如何配置驻留在 OpenShift 集群中 - 连接到 TLS 侦听器 - 信任集群 CA 证书的 Kafka 客户端。

对于内部客户端而言，最简单的方法是使用卷挂载来访问包含所需证书和密钥的 Secret。

按照步骤配置集群 CA 为基于 Java 的 Kafka Producer、Consumer 和 Streams API 签名的信任证书。

根据集群 CA 的证书格式选择要执行的步骤：PKCS #12(.p12)或 PEM(.crt)。

步骤描述了如何将 Cluster Secret 挂载来验证 Kafka 集群的身份到客户端 pod。

先决条件

- Cluster Operator 必须正在运行。
- OpenShift 集群中需要有 Kafka 资源。
- 您需要在 OpenShift 集群内有一个 Kafka 客户端应用，它将使用 TLS 进行连接，需要信任集群 CA 证书。
- 客户端应用程序必须与 Kafka 资源在同一命名空间中运行。

使用 PKCS #12 格式(.p12)

1. 在定义客户端 Pod 时，将集群 Secret 挂载为卷。

   例如：

   -

```
kind: Pod
apiVersion: v1
metadata:
  name: client-pod
spec:
  containers:
  - name: client-name
    image: client-name
    volumeMounts:
    - name: secret-volume
      mountPath: /data/p12
    env:
    - name: SECRET_PASSWORD
      valueFrom:
        secretKeyRef:
          name: my-secret
          key: my-password
  volumes:
  - name: secret-volume
    secret:
      secretName: my-cluster-cluster-ca-cert
```

在这里，我们要挂载：

- PKCS #12 文件到一个精确的路径，该路径可以被配置
- 密码到环境变量中，可在其中用于 Java 配置

2. 使用以下属性配置 Kafka 客户端：

- 安全协议选项：
  - security.protocol：使用 TLS 加密时的 SSL （是否带有 TLS 身份验证）。
  - security.protocol：通过 TLS 使用 SCRAM-SHA 身份验证时 SASL_SSL.
- 导入证书的信任存储位置的 SSL.truststore.location。
- SSL.truststore.password，密码为用于访问信任存储的密码。

- *SSL.truststore.type=PKCS12 标识信任存储类型。*

使用 PEM 格式(.crt)

1. *在定义客户端 Pod 时，将集群 Secret 挂载为卷。*

   例如：

```
kind: Pod
apiVersion: v1
metadata:
  name: client-pod
spec:
  containers:
  - name: client-name
    image: client-name
    volumeMounts:
    - name: secret-volume
      mountPath: /data/crt
  volumes:
  - name: secret-volume
    secret:
      secretName: my-cluster-cluster-ca-cert
```

2. *将证书与使用 X.509 格式的客户端一起使用。*

11.6. 配置外部客户端以信任集群 CA

此流程描述了如何配置驻留在 OpenShift 集群外的 Kafka 客户端 - 连接到 外部 监听器 - 以信任集群 CA 证书。设置客户端时，并在续订期间替换旧客户端 CA 证书时，按照以下步骤操作。

按照步骤配置集群 CA 为基于 Java 的 Kafka Producer、Consumer 和 Streams API 签名的信任证书。

根据集群 CA 的证书格式选择要执行的步骤：PKCS #12(.p12)或 PEM(.crt)。

步骤描述了如何从 Cluster Secret 获取验证 Kafka 集群身份的证书。

重要

<cluster-name>-cluster-ca-cert Secret 在 CA 证书续订期间将包含多个 CA 证书。客户端必须将 所有这些 添加到其信任存储中。

先决条件

- Cluster Operator 必须正在运行。
- OpenShift 集群中需要有 Kafka 资源。
- 您需要在 OpenShift 集群外有一个 Kafka 客户端应用，它将使用 TLS 进行连接，需要信任集群 CA 证书。

使用 PKCS #12 格式(.p12)

1. 从生成的 <cluster-name> -cluster-ca-cert Secret 中提取集群 CA 证书和密码。

```
oc get secret <cluster-name>-cluster-ca-cert -o jsonpath='{.data.ca\.p12}' | base64 -d > ca.p12
```

```
oc get secret <cluster-name>-cluster-ca-cert -o jsonpath='{.data.ca\.password}' | base64 -d > ca.password
```

2. 使用以下属性配置 Kafka 客户端：
   - 安全协议选项：
     - security.protocol：使用 TLS 加密时的 SSL （是否带有 TLS 身份验证）。
     - security.protocol：通过 TLS 使用 SCRAM-SHA 身份验证时 SASL_SSL.
   - 导入证书的信任存储位置的 SSL.truststore.location。
   - SSL.truststore.password，密码为用于访问信任存储的密码。如果信任存储不需要此

属性，则可以省略它。

- SSL.truststore.type=PKCS12 标识信任存储类型。

使用 PEM 格式(.crt)

1. 从生成的 <cluster-name> -cluster-ca-cert Secret 中提取集群 CA 证书。

```
oc get secret <cluster-name>-cluster-ca-cert -o jsonpath='{.data.ca\.crt}' | base64 -d > ca.crt
```

2. 将证书与使用 X.509 格式的客户端一起使用。

11.7. KAFKA 侦听程序证书

您可以为以下类型的监听程序提供自己的服务器证书和私钥：

- 用于 OpenShift 集群内部通信的内部 TLS 侦听器
- 外部监听程序（路由、负载均衡器、 入口 和 节点端口 类型），它们启用了 TLS 加密，用于 Kafka 客户端和 Kafka 代理之间的通信

这些用户提供的证书称为 Kafka 侦听程序证书。

为外部监听程序提供 Kafka 侦听程序证书可让您利用现有安全基础架构，如组织的私有 CA 或公共 CA。Kafka 客户端将使用 Kafka 侦听程序证书而不是由集群 CA 或客户端 CA 签名的证书连接到 Kafka 代理。

在需要时，您必须手动更新 Kafka 侦听程序证书。

11.7.1. 提供您自己的 Kafka 侦听程序证书

此流程演示了如何配置监听程序以使用您自己的私钥和服务器证书，称为 Kafka 侦听程序证书。

您的客户端应用程序应使用 CA 公钥作为可信证书，以验证 Kafka 代理的身份。

先决条件

- OpenShift 集群。
- Cluster Operator 正在运行。
- 对于每个侦听器，由外部 CA 签名的兼容服务器证书。
  - 提供 PEM 格式的 X.509 证书。
  - 为每个监听器指定正确的主题备用名称(SAN)。如需更多信息，请参阅 第 11.7.2 节 “Kafka 监听程序服务器证书中的其他主题”。
  - 您可以在证书文件中提供包含整个 CA 链的证书。

流程

1. 创建包含私钥和服务器证书的 Secret ：

```
oc create secret generic my-secret --from-file=my-listener-key.key --from-file=my-listener-certificate.crt
```

2. 编辑集群的 Kafka 资源。配置侦听器，以在 configuration.brokerCertCertChainAndKey 属性中使用您的 Secret、证书文件和私钥文件。

启用 TLS 加密的负载均衡器 外部监听程序配置示例

```
# ...
listeners:
  - name: plain
    port: 9092
    type: internal
    tls: false
```

```
  - name: external
    port: 9094
    type: loadbalancer
    tls: true
    authentication:
      type: tls
    configuration:
      brokerCertChainAndKey:
        secretName: my-secret
        certificate: my-listener-certificate.crt
        key: my-listener-key.key
# ...
```

TLS 侦听器的配置示例

```
# ...
listeners:
  - name: plain
    port: 9092
    type: internal
    tls: false
  - name: tls
    port: 9093
    type: internal
    tls: true
    authentication:
      type: tls
    configuration:
      brokerCertChainAndKey:
        secretName: my-secret
        certificate: my-listener-certificate.crt
        key: my-listener-key.key
# ...
```

3. 应用新配置以创建或更新资源：

```
oc apply -f kafka.yaml
```

Cluster Operator 会启动 Kafka 集群的滚动更新，该更新会更新监听器的配置。

注意

如果您在已由 TLS 或外部监听器使用的 Secret 中更新 Kafka 侦听程序证书，还会启动滚动更新。

其它资源

- Kafka 监听程序服务器证书中的其他主题
- GenericKafkaListener 模式参考
- Kafka 侦听程序证书

11.7.2. Kafka 监听程序服务器证书中的其他主题

要将 TLS 主机名验证与您自己的 Kafka 侦听程序证书 一起使用，您必须为每个监听器使用正确的 Subject 备用名称(SAN)。证书 SAN 必须指定主机名：

- 集群中的所有 Kafka 代理
- Kafka 集群 bootstrap 服务

如果您的 CA 支持通配符证书，您可以使用通配符证书。

11.7.2.1. TLS 侦听器 SAN 示例

使用以下示例来帮助您在证书中为 TLS 侦听器指定 SAN 的主机名。

通配符示例

```
//Kafka brokers
*.<cluster-name>-kafka-brokers
*.<cluster-name>-kafka-brokers.<namespace>.svc
```

```
// Bootstrap service
<cluster-name>-kafka-bootstrap
<cluster-name>-kafka-bootstrap.<namespace>.svc
```

非通配符示例

```
// Kafka brokers
<cluster-name>-kafka-0.<cluster-name>-kafka-brokers
<cluster-name>-kafka-0.<cluster-name>-kafka-brokers.<namespace>.svc
<cluster-name>-kafka-1.<cluster-name>-kafka-brokers
<cluster-name>-kafka-1.<cluster-name>-kafka-brokers.<namespace>.svc
# ...

// Bootstrap service
<cluster-name>-kafka-bootstrap
<cluster-name>-kafka-bootstrap.<namespace>.svc
```

11.7.2.2. 外部监听程序 SAN 示例

对于启用了 TLS 加密的外部监听程序，您需要在证书中指定的主机名取决于外部监听程序类型。

表 11.10. 每种类型的外部监听器的 sans

外部监听程序类型	在 SANs 中指定...
Route	所有 Kafka 代理 **路由** 的地址以及 bootstrap **路由** 的地址。 您可以使用匹配的通配符名称。
LoadBalancer	所有 Kafka 代理 **负载均衡器和 bootstrap 负载均衡器地址的地址**。 您可以使用匹配的通配符名称。
NodePort	Kafka 代理 Pod 可能调度到的所有 OpenShift 工作程序节点的地址。 您可以使用匹配的通配符名称。

其它资源

- *第 11.7.1 节 “提供您自己的 Kafka 侦听程序证书”*

第 12 章 管理 AMQ 流

本章论述了维护 AMQ 流部署的任务。

12.1. 使用自定义资源

您可以使用 oc 命令来检索信息，并对 AMQ Streams 自定义资源执行其他操作。

通过将 oc 与自定义资源的 status 子资源一起使用，您可以获取有关资源的信息。

12.1.1. 对自定义资源执行 oc 操作

使用 get、describe、edit 或 delete 等 oc 命令对资源类型执行操作。例如，oc get kafkatopics 检索所有 Kafka 主题的列表，oc get kafkas 检索 所有部署的 Kafka 集群。

引用资源类型时，您可以同时使用单数和复数名称：oc get kafkas 的结果与 oc get kafka 相同。

您还可以使用资源的 短名称。在管理 AMQ Streams 时，学习短名称可节省您时间。Kafka 的短名称为 k，因此您也可以运行 oc get k 以 列出所有 Kafka 集群。

```
oc get k

NAME         DESIRED KAFKA REPLICAS   DESIRED ZK REPLICAS
my-cluster   3                        3
```

表 12.1. 每个 AMQ Streams 资源的长和短名称

AMQ Streams 资源	长名称	短名称
kafka	kafka	k
kafka 主题	kafkatopic	kt
Kafka 用户	kafkauser	ku
Kafka Connect	kafkaconnect	kc
Kafka Connect S2I	kafkaconnects2i	kcs2i

AMQ Streams 资源	长名称	短名称
Kafka Connector	kafkaconnector	kctr
Kafka Mirror Maker	kafkamirrormaker	kmm
Kafka Mirror Maker 2	kafkamirrormaker2	kmm2
Kafka Bridge	kafkabridge	kb
kafka 重新平衡	kafkarebalance	kR

12.1.1.1. 资源类型

oc 命令还可以使用自定义资源的类别。

所有 AMQ Streams 自定义资源都属于 category strimzi，因此您可以使用 strimzi 来通过一个命令获取所有 AMQ Streams 资源。

例如，运行 oc get strimzi 会列出给定命名空间中的所有 AMQ Streams 自定义资源。

```
oc get strimzi

NAME                               DESIRED KAFKA REPLICAS DESIRED ZK REPLICAS
kafka.kafka.strimzi.io/my-cluster      3                      3

NAME                               PARTITIONS REPLICATION FACTOR
kafkatopic.kafka.strimzi.io/kafka-apps 3          3

NAME                               AUTHENTICATION AUTHORIZATION
kafkauser.kafka.strimzi.io/my-user     tls            simple
```

oc get strimzi -o name 命令返回所有资源类型和资源名称。o name 选项以 type/name 格式获取输出

```
oc get strimzi -o name

kafka.kafka.strimzi.io/my-cluster
kafkatopic.kafka.strimzi.io/kafka-apps
kafkauser.kafka.strimzi.io/my-user
```

您可以将这一 strimzi 命令与其他命令相结合。例如，您可以将其传递到 oc delete 命令以删除单个命令中的所有资源。

```
oc delete $(oc get strimzi -o name)

kafka.kafka.strimzi.io "my-cluster" deleted
kafkatopic.kafka.strimzi.io "kafka-apps" deleted
kafkauser.kafka.strimzi.io "my-user" deleted
```

在单个操作中删除所有资源可能很有用，例如测试新 AMQ Streams 功能时。

12.1.1.2. 查询子资源的状态

您也可以传递给 -o 选项的其他值。例如，通过使用 -o yaml，您可以获取 YAML 格式的输出。usng -o json 将以 JSON 形式返回。

您可以查看 oc get --help 中的所有选项。

其中一个最有用的选项是 JSONPath 支持，它允许您传递 JSONPath 表达式来查询 Kubernetes API。JSONPath 表达式可以提取或导航任何资源的特定部分。

例如，您可以使用 JSONPath 表达式 {.status.listeners[?(@.type=="tls")].bootstrapServers} 从 Kafka 自定义资源的状态获取 bootstrap 地址，并在 Kafka 客户端中使用它。

此处，命令找到 tls 侦听器的 bootstrapServers 值。

```
oc get kafka my-cluster -o=jsonpath='{.status.listeners[?(@.type=="tls")].bootstrapServers}
{"\n"}'

my-cluster-kafka-bootstrap.myproject.svc:9093
```

通过将类型条件更改为 @.type=="external" 或 @.type=="plain"，您也可以获取其他 Kafka 侦听器的地址。

```
oc get kafka my-cluster -o=jsonpath='{.status.listeners[?
(@.type=="external")].bootstrapServers}{"\n"}'

192.168.1.247:9094
```

您可以使用 jsonpath 从任何自定义资源中提取任何其他属性或属性组。

12.1.2. AMQ Streams 自定义资源状态信息

几个资源具有 status 属性，如下表中所述。

表 12.2. 自定义资源状态属性

AMQ Streams 资源	架构参考	在... 上发布状态信息
kafka	第 13.2.55 节 “**KafkaStatus** 架构参考”	Kafka 集群。
KafkaConnect	第 13.2.82 节 “**KafkaConnectStatus** 模式参考”	Kafka Connect 集群（如果已部署）。
KafkaConnectS2I	第 13.2.86 节 “**KafkaConnectS2IStatus** 模式参考”	带有 Source-to-Image 支持的 Kafka Connect 集群（如果已部署）。
KafkaConnector	第 13.2.122 节 “**KafkaConnectorStatus** 模式参考”	KafkaConnector 资源（如果已部署）。
KafkaMirrorMaker	第 13.2.109 节 “**KafkaMirrorMakerStatus** 模式参考”	Kafka MirrorMaker 工具（如果已部署）。
KafkaTopic	第 13.2.89 节 “**KafkaTopicStatus** schema reference”	Kafka 集群中的主题。
KafkaUser	第 13.2.102 节 “**KafkaUserStatus** 模式参考”	Kafka 集群中的用户。
KafkaBridge	第 13.2.119 节 “**KafkaBridgeStatus** 模式参考”	AMQ Streams Kafka Bridge（如果已部署）。

资源的 status 属性提供资源的信息：

- 当前状态，在 status.conditions 属性中
- 最后观察到的生成，在 status.observedGeneration 属性中

status 属性也提供特定于资源的信息。例如：

- KafkaStatus 提供有关监听地址和 Kafka 集群 ID 的信息。
- KafkaConnectStatus 为 Kafka Connect 连接器提供 REST API 端点。
- KafkaUserStatus 提供 Kafka 用户的用户名以及存储其凭证的 Secret。
- KafkaBridgeStatus 提供 HTTP 地址，外部客户端应用程序可以访问 Bridge 服务。

资源的当前状态可用于跟踪与达到所需状态的资源相关的进度，如 spec 属性所定义。状态条件提供了资源更改的时间和原因，以及防止或延迟 Operator 实现资源所需状态的事件详情。

最后观察到的生成是 Cluster Operator 最后协调的资源的生成。如果 observedGeneration 的值与 metadata.generation 的值不同，Operator 还没有对资源的最新版本进行处理。如果这些值相同，状态信息反映了对资源的最新更改。

AMQ Streams 创建和维护自定义资源的状态，定期评估自定义资源的当前状态并相应地更新其状态。当使用 oc edit 对自定义资源执行更新时，其状态不可编辑。此外，更改状态不会影响 Kafka 集群的配置。

此处，我们会看到为 Kafka 自定义资源指定的 status 属性。

带有状态的 Kafka 自定义资源

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
spec:
  # ...
status:
  conditions: (1)
  - lastTransitionTime: 2021-07-23T23:46:57+0000
    status: "True"
    type: Ready (2)
  observedGeneration: 4 (3)
```

```
  listeners: 4
  - addresses:
    - host: my-cluster-kafka-bootstrap.myproject.svc
      port: 9092
    type: plain
  - addresses:
    - host: my-cluster-kafka-bootstrap.myproject.svc
      port: 9093
    certificates:
    - |
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
    type: tls
  - addresses:
    - host: 172.29.49.180
      port: 9094
    certificates:
    - |
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
    type: external
  clusterId: CLUSTER-ID 5
# ...
```

1

状态 条件 描述与无法从现有资源信息推断或特定于资源实例相关的条件。

2

Ready 条件指示 Cluster Operator 当前是否认为 Kafka 集群可以处理流量。

3

observedGeneration 表示最近由 Cluster Operator 协调的 Kafka 自定义资源的生成。

4

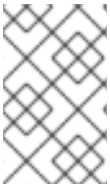
侦听器 根据类型描述当前的 Kafka bootstrap 地址。

5

Kafka 集群 ID。

重要

目前不支持带有 nodeport 类型的外部监听程序自定义资源状态中的地址。



注意

状态中列出的 Kafka bootstrap 地址不表示这些端点或 Kafka 集群处于 ready 状态。

访问状态信息

您可以从命令行访问资源的状态信息。如需更多信息，请参阅 第 12.1.3 节 “查找自定义资源的状态”。

12.1.3. 查找自定义资源的状态

此流程描述了如何查找自定义资源的状态。

先决条件

- OpenShift 集群。
- Cluster Operator 正在运行。

流程

- 指定自定义资源，并使用 -o jsonpath 选项应用标准 JSONPath 表达式来选择 status 属性：

```
oc get kafka <kafka_resource_name> -o jsonpath='{.status}'
```

此表达式返回指定自定义资源的所有状态信息。您可以使用点表示法（如 status.listeners 或 status.observedGeneration ）来微调您希望看到的状态信息。

其它资源

- 第 12.1.2 节 “AMQ Streams 自定义资源状态信息”

- 有关使用 JSONPath 的更多信息，请参阅 JSONPath 支持。

12.2. 暂停自定义资源协调

有时，暂停由 AMQ Streams Operator 管理的自定义资源的协调会很有用，以便您可以执行修复或更新。如果暂停协调，Operator 会忽略对自定义资源所做的任何更改，直到暂停结束。

如果要暂停自定义资源的协调，在其配置中将 strimzi.io/pause-reconciliation 注解设置为 true。这会指示适当的 Operator 暂停自定义资源的协调。例如，您可以将注解应用到 KafkaConnect 资源，以便暂停 Cluster Operator 的协调。

您还可以在启用了 pause 注解的情况下创建自定义资源。自定义资源会被创建，但会被忽略。

先决条件

- 管理自定义资源的 AMQ Streams Operator 正在运行。

流程

1. 给 OpenShift 中的自定义资源标注，将 pause-reconciliation 设置为 true ：

```
oc annotate KIND-OF-CUSTOM-RESOURCE NAME-OF-CUSTOM-RESOURCE strimzi.io/pause-reconciliation="true"
```

例如，对于 KafkaConnect 自定义资源：

```
oc annotate KafkaConnect my-connect strimzi.io/pause-reconciliation="true"
```

2. 检查自定义资源的状态条件是否显示对 ReconciliationPaused 的更改：

```
oc describe KIND-OF-CUSTOM-RESOURCE NAME-OF-CUSTOM-RESOURCE
```

类型 条件更改为 lastTransitionTime 使用的 ReconciliationPa。

带有暂停协调条件类型的自定义资源示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  annotations:
    strimzi.io/pause-reconciliation: "true"
    strimzi.io/use-connector-resources: "true"
  creationTimestamp: 2021-03-12T10:47:11Z
  #...
spec:
  # ...
status:
  conditions:
  - lastTransitionTime: 2021-03-12T10:47:41.689249Z
    status: "True"
    type: ReconciliationPaused
```

恢复暂停

- 要恢复协调，您可以将注解设置为 false，或删除注解。

其它资源

- 自定义 OpenShift 资源
- 查找自定义资源的状态

12.3. 手动启动 KAFKA 和 ZOOKEEPER 集群的滚动更新

AMQ Streams 支持在 StatefulSet 和 Pod 资源上使用注解来通过 Cluster Operator 手动触发 Kafka 和 ZooKeeper 集群的滚动更新。滚动更新使用新 pod 重启资源的 pod。

通常仅在特殊情况下需要手动对来自同一 StatefulSet 的特定 pod 或一组 pod 执行滚动更新。但是，如果您通过 Cluster Operator 执行滚动更新，请确保：

- 手动删除 pod 不会与同时 Cluster Operator 操作冲突，如并行删除其他 pod。
- Cluster Operator 逻辑处理 Kafka 配置规格，如同步副本的数量。

12.3.1. 先决条件

要执行手动滚动更新，您需要一个正在运行的 Cluster Operator 和 Kafka 集群。

有关运行 以下的说明，请参阅 OpenShift 指南中的部署和升级 AMQ Streams ：

- Cluster Operator
- Kafka 集群

12.3.2. 使用 StatefulSet 注解执行滚动更新

此流程描述了如何使用 OpenShift StatefulSet 注解手动触发现有 Kafka 集群或 ZooKeeper 集群的滚动更新。

流程

1. 查找控制您要手动更新的 Kafka 或 ZooKeeper pod 的 StatefulSet 的名称。

   例如，如果您的 Kafka 集群名为 my-cluster，对应的 StatefulSet 名称为 my-cluster-kafka 和 my-cluster-zookeeper。

2. 给 OpenShift 中的 StatefulSet 资源标注。

   使用 oc annotate：

```
oc annotate statefulset cluster-name-kafka strimzi.io/manual-rolling-update=true

oc annotate statefulset cluster-name-zookeeper strimzi.io/manual-rolling-update=true
```

3.

等待下一次协调发生（默认为每隔两分钟）。只要协调过程检测到注解的 StatefulSet，则会触发注解的滚动更新。当所有 pod 的滚动更新完成后，注解会从 StatefulSet 中删除。

12.3.3. 使用 Pod 注解执行滚动更新

此流程描述了如何使用 OpenShift Pod 注解手动触发现有 Kafka 集群或 ZooKeeper 集群的滚动更新。当同一 StatefulSet 的多个 pod 被注解时，会在同一个协调运行中连续滚动更新。

流程

1.

查找您要手动更新的 Kafka 或 ZooKeeper Pod 的名称。

例如，如果您的 Kafka 集群名为 my-cluster，对应的 Pod 名称为 my-cluster-kafka-index 和 my-cluster-zookeeper-index。索引 从零开始，最后以副本总数结束。

2.

给 OpenShift 中的 容器集资源标注：

使用 oc annotate:

```
oc annotate pod cluster-name-kafka-index strimzi.io/manual-rolling-update=true

oc annotate pod cluster-name-zookeeper-index strimzi.io/manual-rolling-update=true
```

3.

等待下一次协调发生（默认为每隔两分钟）。只要协调过程检测到注解，则 会触发注解的滚动更新。pod 的滚动更新完成后，该注解会从 Pod 中删除。

12.4. 使用标签和注解发现服务

服务发现使得在与 AMQ Streams 相同的 OpenShift 集群中运行的客户端应用可以更轻松地与 Kafka 集群交互。

为用于访问 Kafka 集群的服务生成 服务发现 标签和注解：

- 内部 Kafka bootstrap 服务

- HTTP Bridge 服务

该标签有助于使服务可被发现，注释提供了客户端应用程序可用于进行连接的连接详情。

Service 资源的服务发现标签, strimzi.io/discovery 设置为 true。服务发现注释具有相同的密钥，以 JSON 格式提供每个服务的连接详细信息。

内部 Kafka bootstrap 服务示例

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    strimzi.io/discovery: |-
      [ {
        "port" : 9092,
        "tls" : false,
        "protocol" : "kafka",
        "auth" : "scram-sha-512"
      }, {
        "port" : 9093,
        "tls" : true,
        "protocol" : "kafka",
        "auth" : "tls"
      } ]
  labels:
    strimzi.io/cluster: my-cluster
    strimzi.io/discovery: "true"
    strimzi.io/kind: Kafka
    strimzi.io/name: my-cluster-kafka-bootstrap
  name: my-cluster-kafka-bootstrap
spec:
  #...
```

HTTP Bridge 服务示例

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    strimzi.io/discovery: |-
      [ {
        "port" : 8080,
        "tls" : false,
        "auth" : "none",
        "protocol" : "http"
      } ]
  labels:
    strimzi.io/cluster: my-bridge
```

```
strimzi.io/discovery: "true"
strimzi.io/kind: KafkaBridge
strimzi.io/name: my-bridge-bridge-service
```

12.4.1. 返回服务的连接详情

您可以从命令行或对应的 API 调用时指定发现标签来查找服务。

```
oc get service -l strimzi.io/discovery=true
```

在检索服务发现标签时，将返回连接详情。

12.5. 从持久性卷中恢复集群

如果持久性卷(PV)仍然存在，您可以从它们恢复 Kafka 集群。

您可能想要执行此操作，例如：

- 命名空间被意外删除
- 整个 OpenShift 集群会丢失，但 PV 会保留在基础架构中

12.5.1. 从命名空间删除中恢复

由于持久性卷和命名空间之间的关系，可以从删除命名空间中进行恢复。A PersistentVolume (PV)是位于命名空间外的存储资源。PV 使用 PersistentVolumeClaim (PVC)挂载到 Kafka pod 中，该 PVC 驻留在一个命名空间内。

PV 的重新声明(reclaim)策略指定了在删除命名空间时集群如何操作。如果重新声明策略被设置为：

- 删除 （默认）当 PVC 在命名空间中被删除时会删除 PV
- 保留，在删除命名空间时不会删除 PV

为确保意外删除命名空间时，您可以从 PV 中恢复该策略，必须在 PV 规格中使用 persistentVolumeReclaimPolicy 属性重置该 策略 ：

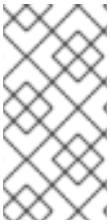
```
apiVersion: v1
kind: PersistentVolume
# ...
spec:
  # ...
  persistentVolumeReclaimPolicy: Retain
```

另外， PV 可以继承关联的存储类的重新声明策略。存储类用于动态卷分配。

通过为存储类配置 reclaimPolicy 属性，使用存储类的 PV 会使用适当的重新声明策略创建。使用 storageClassName 属性为 PV 配置存储类。

```
apiVersion: v1
kind: StorageClass
metadata:
  name: gp2-retain
parameters:
  # ...
# ...
reclaimPolicy: Retain
```

```
apiVersion: v1
kind: PersistentVolume
# ...
spec:
  # ...
  storageClassName: gp2-retain
```



注意

如果您使用 Retain 作为重新声明策略，但要删除整个集群，则需要手动删除 PV。否则它们将不会被删除，并可能导致不必要的资源开支。

12.5.2. 在 OpenShift 集群丢失后进行恢复

当集群丢失时，如果您在基础架构中保留了集群，则可以使用磁盘/卷中的数据来恢复集群。恢复过程与删除命名空间的过程相同，假设 PV 可以恢复并手动创建。

12.5.3. 从持久性卷中恢复已删除的集群

这个步骤描述了如何从持久性卷(PV)中恢复删除的集群。

在这种情况下，Topic Operator 会识别 Kafka 中存在的主题，但 KafkaTopic 资源不存在。

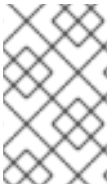
当您进入重新创建集群的步骤时，有两个选项：

1. 当可以恢复所有 KafkaTopic 资源时，请使用 Option 1。

   因此，必须在集群启动前恢复 KafkaTopic 资源，以便 Topic Operator 不会删除对应的主题。

2. 当无法恢复所有 KafkaTopic 资源时，请使用 Option 2。

   在这种情况下，您可以在没有 Topic Operator 的情况下部署集群，删除 Topic Operator 主题存储元数据，然后使用 Topic Operator 重新部署 Kafka 集群，以便它能够从对应主题重新创建 KafkaTopic 资源。



注意

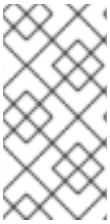
如果没有部署 Topic Operator，您只需要恢复 PersistentVolumeClaim (PVC)资源。

开始前

在这一流程中，必须将 PV 挂载到正确的 PVC 中以避免数据崩溃。为 PVC 指定了一个 volumeName，且必须与 PV 的名称匹配。

如需更多信息，请参阅：

- 持久性卷声明命名
- JBOD 和持久性卷声明



注意

该流程不包括 KafkaUser 资源的恢复，必须手动重新创建。如果需要保留密码和证书，则必须在创建 KafkaUser 资源前重新创建 secret。

流程

1. 检查集群中的 PV 信息：

```
oc get pv
```

为 PV 提供数据信息。

显示对这个流程很重要的列的输出示例：

```
NAME                                       RECLAIMPOLICY CLAIM
pvc-5e9c5c7f-3317-11ea-a650-06e1eadd9a4c ... Retain ...   myproject/data-my-cluster-zookeeper-1
pvc-5e9cc72d-3317-11ea-97b0-0aef8816c7ea ... Retain ...   myproject/data-my-cluster-zookeeper-0
pvc-5ead43d1-3317-11ea-97b0-0aef8816c7ea ... Retain ...   myproject/data-my-cluster-zookeeper-2
pvc-7e1f67f9-3317-11ea-a650-06e1eadd9a4c ... Retain ...   myproject/data-0-my-cluster-kafka-0
pvc-7e21042e-3317-11ea-9786-02deaf9aa87e ... Retain ...   myproject/data-0-my-cluster-kafka-1
pvc-7e226978-3317-11ea-97b0-0aef8816c7ea ... Retain ...   myproject/data-0-my-cluster-kafka-2
```

- NAME 显示每个 PV 的名称。
- RECLAIM POLICY 显示 PV 会被 保留。
- CLAIM 显示到原始 PVC 的链接。

2. 重新创建原始命名空间：

```
oc create namespace myproject
```

3. *重新创建原始 PVC 资源规格，将 PVC 链接到适当的 PV：*

例如：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-0-my-cluster-kafka-0
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
  storageClassName: gp2-retain
  volumeMode: Filesystem
  volumeName: pvc-7e1f67f9-3317-11ea-a650-06e1eadd9a4c
```

4. *编辑 PV 规格，以删除绑定原始 PVC 的 claimRef 属性。*

例如：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    kubernetes.io/createdby: aws-ebs-dynamic-provisioner
    pv.kubernetes.io/bound-by-controller: "yes"
    pv.kubernetes.io/provisioned-by: kubernetes.io/aws-ebs
  creationTimestamp: "<date>"
  finalizers:
  - kubernetes.io/pv-protection
  labels:
    failure-domain.beta.kubernetes.io/region: eu-west-1
    failure-domain.beta.kubernetes.io/zone: eu-west-1c
  name: pvc-7e226978-3317-11ea-97b0-0aef8816c7ea
  resourceVersion: "39431"
  selfLink: /api/v1/persistentvolumes/pvc-7e226978-3317-11ea-97b0-0aef8816c7ea
  uid: 7efe6b0d-3317-11ea-a650-06e1eadd9a4c
spec:
  accessModes:
  - ReadWriteOnce
  awsElasticBlockStore:
    fsType: xfs
    volumeID: aws://eu-west-1c/vol-09db3141656d1c258
  capacity:
    storage: 100Gi
  claimRef:
    apiVersion: v1
```

```
  kind: PersistentVolumeClaim
  name: data-0-my-cluster-kafka-2
  namespace: myproject
  resourceVersion: "39113"
  uid: 54be1c60-3319-11ea-97b0-0aef8816c7ea
 nodeAffinity:
  required:
   nodeSelectorTerms:
   - matchExpressions:
     - key: failure-domain.beta.kubernetes.io/zone
       operator: In
       values:
       - eu-west-1c
     - key: failure-domain.beta.kubernetes.io/region
       operator: In
       values:
       - eu-west-1
 persistentVolumeReclaimPolicy: Retain
 storageClassName: gp2-retain
 volumeMode: Filesystem
```

在这个示例中，删除了以下属性：

```
claimRef:
  apiVersion: v1
  kind: PersistentVolumeClaim
  name: data-0-my-cluster-kafka-2
  namespace: myproject
  resourceVersion: "39113"
  uid: 54be1c60-3319-11ea-97b0-0aef8816c7ea
```

5. 部署 Cluster Operator。

```
oc create -f install/cluster-operator -n my-project
```

6. 重新创建集群。

根据您是否拥有重新创建集群所需的所有 KafkaTopic 资源，按照以下步骤执行操作。

选项 1 ：如果您在丢失集群前已存在 所有 KafkaTopic 资源，包括 __consumer_offsets 提供的内部主题：

1. 重新创建所有 KafkaTopic 资源。

您必须在部署集群前重新创建资源，否则 Topic Operator 将删除主题。

2. *部署 Kafka 集群。*

   例如：

```
oc apply -f kafka.yaml
```

选项 2 ：如果您没有丢失集群前存在的所有 KafkaTopic 资源：

1. *部署 Kafka 集群，如第一个选项一样，但不通过在部署前从 Kafka 资源中删除 topicOperator 属性来不使用 Topic Operator。*

   如果您在部署中包含主题 Operator，主题 Operator 将删除所有主题。

2. *从 Kafka 集群中删除内部主题存储主题：*

```
oc run kafka-admin -ti --image=registry.redhat.io/amq7/amq-streams-kafka-28-rhel7:1.8.0 --rm=true --restart=Never -- ./bin/kafka-topics.sh --bootstrap-server localhost:9092 --topic __strimzi-topic-operator-kstreams-topic-store-changelog --delete && ./bin/kafka-topics.sh --bootstrap-server localhost:9092 --topic __strimzi_store_topic --delete
```

   命令必须与用于访问 Kafka 集群的监听程序和身份验证对应。

3. *通过使用 topicOperator 属性重新部署 Kafka 集群来启用 Topic Operator，以重新创建 KafkaTopic 资源。*

   例如：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  #...
```

```
entityOperator:
  topicOperator: {} 1
  #...
```

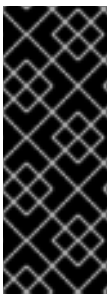
1

在此，我们会显示默认配置，它没有附加属性。您可以使用 第 13.2.45 节 “EntityTopicOperatorSpec schema reference” 中描述的属性指定所需的配置。

7. 通过列出 KafkaTopic 资源来验证恢复：

```
oc get KafkaTopic
```

12.6. 使用 KAFKA STATIC QUOTA 插件设置代理的限制



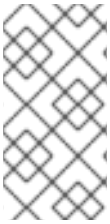
重要

Kafka 静态配额插件只是一个技术预览。技术预览功能不被红帽产品服务级别协议 (SLA)支持，且可能无法完成功能。红帽不推荐在生产环境中实施任何技术预览功能。此技术预览功能为您提供对即将推出的产品创新的早期访问，允许您在开发过程中测试并提供反馈。有关红帽技术预览功能支持范围的更多信息，请参阅 技术预览功能支持范围。

使用 Kafka Static Quota 插件，为 Kafka 集群中的代理设置吞吐量和存储限制。您可以通过配置 Kafka 资源启用插件并设置限值。您可以设置字节阈值和存储配额，对与代理交互的客户端设定限制。

您可以为生产者和消费者带宽设置字节速率阈值。总限值分布在所有访问代理的客户端中。例如，您可以为生产者设置 40 MBps 的字节阈值。如果两个制作者正在运行，则每个生产者的吞吐量限制为 20 MBps。

存储配额在软限制和硬限制之间节流 Kafka 磁盘存储限制。这些限制适用于所有可用磁盘空间。在软限制和硬限制之间逐渐减慢生产者的速度。这些限制可防止磁盘过快占用并超出其容量。完整磁盘可能会导致问题难以解决。硬限制是最大存储限制。



注意

对于 JBOD 存储，限制适用于所有磁盘。如果代理使用两个 1 TB 磁盘，且配额为 1.1 TB，则一个磁盘可能会填满，另一个磁盘将几乎为空。

先决条件

- *管理 Kafka 集群的 Cluster Operator 正在运行。*

流程

1. *将插件属性添加到 Kafka 资源的配置中。*

   示例配置中显示了插件属性。

   Kafka 静态配额插件配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    config:
      client.quota.callback.class: io.strimzi.kafka.quotas.StaticQuotaCallback (1)
      client.quota.callback.static.produce: 1000000 (2)
      client.quota.callback.static.fetch: 1000000 (3)
      client.quota.callback.static.storage.soft: 400000000000 (4)
      client.quota.callback.static.storage.hard: 500000000000 (5)
      client.quota.callback.static.storage.check-interval: 5 (6)
```

(1) *加载 Kafka 静态配额插件。*

(2) *设置制作者字节率阈值。本例中为 1 Mbps。*

(3) *设置使用者字节率阈值。本例中为 1 Mbps。*

(4)

为存储设置下限。本例中为 400 GB。

5

为存储设置较高的硬限制。本例中为 500 GB。

6

设置存储检查间隔（以秒为单位）。在这个示例中为 5 秒。您可以将此项设置为 0 以禁用检查。

2. 更新资源。

```
oc apply -f KAFKA-CONFIG-FILE
```

其它资源

- Kafka 代理配置调整
- 设置用户配额

12.7. 调整 KAFKA 配置

使用配置属性优化 Kafka 代理、生产者和使用者的性能。

需要最小的配置属性集合，但您可以添加或调整属性以更改生产者和使用者如何与 Kafka 代理交互。例如，您可以调整消息的延迟和吞吐量，以便客户端能够实时响应数据。

首先，您可以分析指标来测量进行初始配置的位置，然后逐步更改并进一步比较指标，直到您拥有所需的配置。

其它资源

- Apache Kafka 文档

12.7.1. Kafka 代理配置调整

使用配置属性优化 Kafka 代理的性能。您可以使用标准 Kafka 代理配置选项，但由 AMQ Streams 管理的属性除外。

12.7.1.1. 基本代理配置

某些代理配置选项由 Kafka 自定义资源规格直接由 AMQ Streams 管理：

- broker.id 是 Kafka 代理的 ID
- log.dirs 是日志数据的目录
- zookeeper.connect 是用于将 Kafka 与 ZooKeeper 连接的配置
- 侦听器 向客户端公开 Kafka 集群
- 授权 机制允许或拒绝用户执行的操作
- 验证 机制证明需要访问 Kafka 的用户的身份

代理 ID 从 0（零）开始，对应于代理副本数。日志目录根据 Kafka 自定义资源中的 spec.kafka.storage 配置挂载到 /var/lib/kafka/data/kafka -logIDX 中。IDX 是 Kafka 代理 pod 索引。

因此，您无法通过 Kafka 自定义资源的 config 属性配置 这些选项。有关排除列表，请参阅 KafkaClusterSpec 模式引用。

但是，典型的代理配置将包括与主题、线程和日志相关的属性设置。

基本代理配置属性

```
# ...
num.partitions=1
```

```
default.replication.factor=3
offsets.topic.replication.factor=3
transaction.state.log.replication.factor=3
transaction.state.log.min.isr=2
log.retention.hours=168
log.segment.bytes=1073741824
log.retention.check.interval.ms=300000
num.network.threads=3
num.io.threads=8
num.recovery.threads.per.data.dir=1
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
group.initial.rebalance.delay.ms=0
zookeeper.connection.timeout.ms=6000
# ...
```

12.7.1.2. 复制高可用性主题

基本主题属性为主题设置默认分区和复制因子，它们将应用到没有明确设置这些属性的主题，包括在自动创建主题时。

```
# ...
num.partitions=1
auto.create.topics.enable=false
default.replication.factor=3
min.insync.replicas=2
replica.fetch.max.bytes=1048576
# ...
```

auto.create.topics.enable 属性默认为启用，因此生产者和使用者需要时会自动创建不存在的主题。如果您使用自动创建主题，可以使用 num.partitions 为主题设置默认分区数。但通常禁用此属性，以便通过创建显式主题来更好地控制主题，您可以使用 AMQ Streams KafkaTopic 资源或应用程序来创建主题。

对于高可用性环境，建议将主题的复制因素增加到至少 3，并将所需同步副本的最小数量设置为比复制因素少 1 个。对于使用 KafkaTopic 资源创建的主题，复制因素使用 spec.replicas 设置。

对于 数据持久性，您还应在 主题 配置中设置 min.insync.replicas，并在 制作者 配置中使用 acks=all 设置消息发送确认。

使用 replica.fetch.max.bytes 设置复制领导分区的每个后续程序获取的消息的最大大小（以字节为单

位）。根据平均消息大小和吞吐量更改此值。在考虑读/写缓冲所需的总内存分配时，可用内存还必须能够适应所有跟随者乘以时的最大复制消息大小。

delete.topic.enable 属性默认为启用，以允许删除主题。在生产环境中，您应该禁用此属性以避免意外删除主题，从而导致数据丢失。但是，您可以临时启用它并删除主题，然后再次禁用它。如果启用了 delete.topic.enable，您可以使用 KafkaTopic 资源删除主题。

```
# ...
auto.create.topics.enable=false
delete.topic.enable=true
# ...
```

12.7.1.3. 事务和提交的内部主题设置

如果您使用事务来启用从生产者到分区的原子写入，事务状态将存储在内部 __transaction_state 主题中。默认情况下，代理被配置为有 3 个复制因素，这个主题最少需要 2 个 in-sync 副本，这意味着 Kafka 集群中至少需要三个代理。

```
# ...
transaction.state.log.replication.factor=3
transaction.state.log.min.isr=2
# ...
```

同样，存储消费者状态的内部 _ consumer_offsets 主题对分区数量和复制因素具有默认设置。

```
# ...
offsets.topic.num.partitions=50
offsets.topic.replication.factor=3
# ...
```

不要在生产环境中减少这些设置。您可以 在生产环境中 增加设置。作为例外，您可能要减少单broker 测试 环境中的设置。

12.7.1.4. 通过增加 I/O 线程改进请求处理吞吐量

网络线程处理对 Kafka 集群的请求，如从客户端应用程序生成和获取请求。将生成请求放入请求队列中。将响应放入响应队列中。

网络线程数量应当反映复制因素以及与 Kafka 集群交互的客户端生产者和消费者的活动级别。如果您要有大量请求，您可以增加线程数量，使用时间线程闲置来确定何时添加更多线程。

要减少拥塞并规范请求流量，您可以在阻止网络线程前限制请求队列中允许的请求数。

I/O 线程从请求队列获取请求来处理它们。添加更多线程可提高吞吐量，但 CPU 内核数和磁盘带宽会施加一个实际的上限。至少，I/O 线程数量应当等于存储卷的数量。

```
# ...
num.network.threads=3 1
queued.max.requests=500 2
num.io.threads=8 3
num.recovery.threads.per.data.dir=1 4
# ...
```

1

Kafka 集群的网络线程数量。

2

请求队列中允许的请求数。

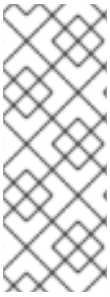
3

Kafka 代理的 I/O 线程数量。

4

用于在启动时加载日志的线程数，在关机时用于清除日志。

所有代理的线程池的配置更新可能会在集群级别动态进行。这些更新限制为当前大小的一半和当前大小的两倍。



注意

Kafka 代理指标可帮助确定所需的线程数量。例如，平均时间网络线程闲置的指标 (kafka.network:type=SocketServer,name=NetworkProcessorAvgIdlePercent)表示已使用的资源的百分比。如果有 0% 的空闲时间，则所有资源都在使用，这意味着添加更多线程可能很有用。

如果线程因为磁盘数量慢或受限，您可以尝试增加网络请求的缓冲大小以提高吞吐量：

```
# ...
replica.socket.receive.buffer.bytes=65536
# ...
```

另外，增加 bytes Kafka 可接收的最大字节数：

```
# ...
socket.request.max.bytes=104857600
# ...
```

12.7.1.5. 为高延迟连接增加带宽

Kafka 批量数据，以便在从 Kafka 到客户端的高延迟连接中实现合理的吞吐量，如数据中心之间的连接。但是，如果存在高延迟问题，您可以增加用于发送和接收消息的缓冲区的大小。

```
# ...
socket.send.buffer.bytes=1048576
socket.receive.buffer.bytes=1048576
# ...
```

您可以使用带宽延迟（以字节/秒为单位）估算缓冲区的最大带宽，以估算保持最大吞吐量所需的缓冲区的大小（以字节/为单位）。

12.7.1.6. 使用数据保留策略管理日志

Kafka 使用日志来存储消息数据。日志是与各种索引关联的一系列片段。新消息写入活动段，随后从不修改。服务片段从消费者获取请求时读取。活动片段定期被滚动为只读，并创建一个新的活跃网段来替换它。次只有一个部分处于活动状态。旧片段会保留，直到它们符合删除条件。

代理级别的配置设置日志片段的最大大小，以及推出活跃片段前以毫秒为单位的最大时间：

```
# ...
log.segment.bytes=1073741824
log.roll.ms=604800000
# ...
```

您可以使用 segment. bytes 和 segment.ms，在主题级别上覆盖这些设置。您是否需要降低还是提升这些值取决于片段删除策略。较大的大小意味着活跃片段包含更多信息，且不频繁地推出。段也不再频繁地符合删除条件。

您可以设置基于时间或基于大小的日志保留和清理策略，以便保持日志可被管理。根据您的要求，您

可以使用日志保留配置来删除旧片段。如果使用日志保留策略，在达到保留限制时会删除非活跃日志片段。删除旧片段会绑定日志所需的存储空间，因此您不会超过磁盘容量。

对于基于时间的日志保留，您需要根据小时、分钟和毫秒设置保留期限。保留周期基于附加至网段的时间信息。

毫秒配置具有超过分钟的优先级，其优先级高于小时。默认情况下，分钟和毫秒配置为空，但这三个选项提供了对您要保留的数据的大量控制。应优先使用毫秒配置，因为它是唯一可动态更新的三个属性之一。

```
# ...
log.retention.ms=1680000
# ...
```

如果 log.retention.ms 设为 -1，则不会对日志保留应用任何时间限制，因此所有日志都会保留。磁盘使用量应始终受到监控，但通常不建议 -1 设置，因为它可能会导致完整磁盘出现问题，而这可能难以修复。

对于基于大小的日志保留，您需要以字节为单位设置最大日志大小（日志中所有片段）：

```
# ...
log.retention.bytes=1073741824
# ...
```

换句话说，当日志达到稳定状态后，通常大约有 log.retention.bytes/log.segment.bytes 片段。达到最大日志大小时，会删除旧的网段。

使用最大日志大小时潜在的问题在于它没有考虑消息附加到片段中的时间。您可以将基于时间和大小的日志保留用于清理策略，以获得所需的平衡。达到阈值时首先触发清理。

如果要在从系统中删除片段文件前添加时间延迟，您可以为主题配置中的特定主题使用 log.segment.delete.delay.ms 为代理级别或 file.delay.ms 添加延迟。

```
# ...
log.segment.delete.delay.ms=60000
# ...
```

12.7.1.7. 使用清理策略删除日志数据

删除较旧日志数据的方法由 日志清理 配置决定。

默认情况下，为代理启用日志清理功能：

```
# ...
log.cleaner.enable=true
# ...
```

您可以在主题或代理级别设置清理策略。代理级配置是尚未设置策略的主题的默认设置。

您可以将策略设置为删除日志、紧凑日志，或同时执行这两者：

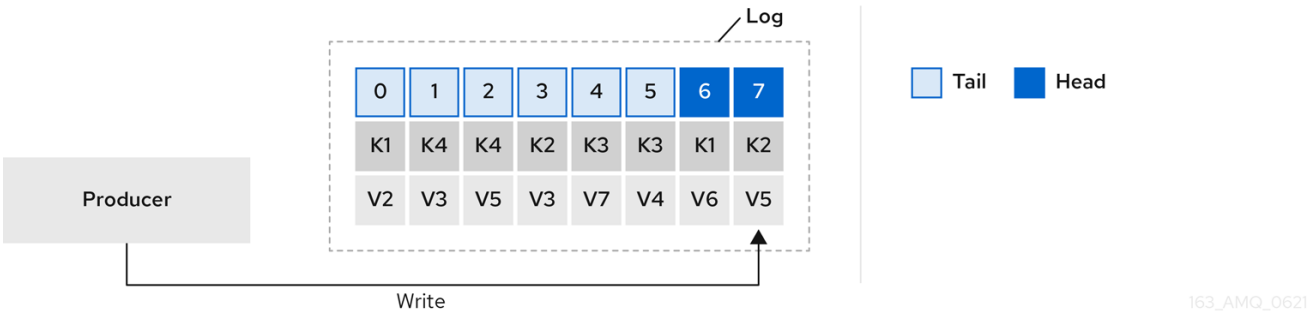
```
# ...
log.cleanup.policy=compact,delete
# ...
```

delete 策略与使用数据保留策略管理日志对应。当不需要永久保留数据时，它非常适合。紧凑 策略保证为每个消息键保留最新的消息。日志紧凑适合更改消息值，您想要保留最新的更新。

如果将清理策略设置为删除日志，则会根据日志保留限制删除旧的片段。否则，如果没有启用日志清理功能，且没有日志保留限制，日志将继续增加。

如果为日志紧凑设置了清理策略，日志 的头 将作为一个标准 Kafka 日志运行，按顺序为新消息写入操作。在紧凑日志的 尾部 （日志清理操作），如果日志中稍后出现具有相同密钥的另一条记录，则将删除记录。带有 null 值的消息也会被删除。如果不使用键，就无法使用紧凑，因为需要键来识别相关消息。虽然 Kafka 保证每个密钥的最新消息将被保留，但它不能保证整个紧凑的日志不会包含重复项。

图 12.1. 在压缩前以偏移位置显示键值写入的日志

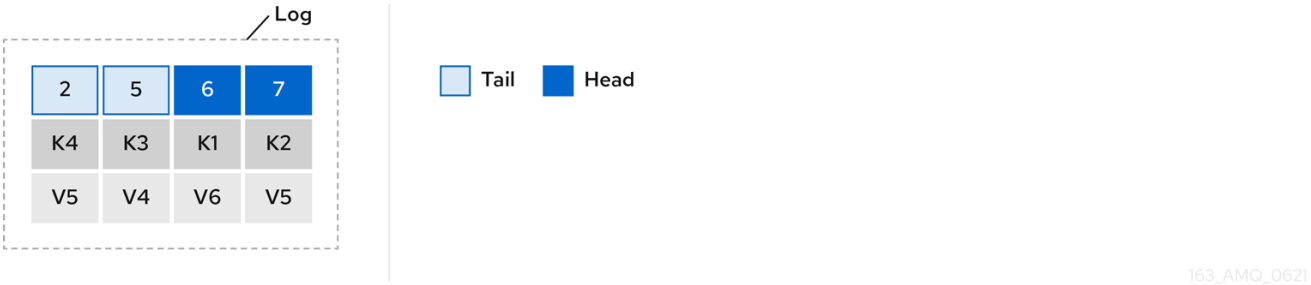


使用密钥来识别信息，Kafka 紧凑会保留特定消息键的最新消息（带有最高偏移值），最终丢弃之前

具有相同密钥的消息。换句话说，其最新状态的消息始终可用，当日志安全运行时，该特定消息的任何过期记录最终都会被删除。您可以将消息恢复到以前的状态。

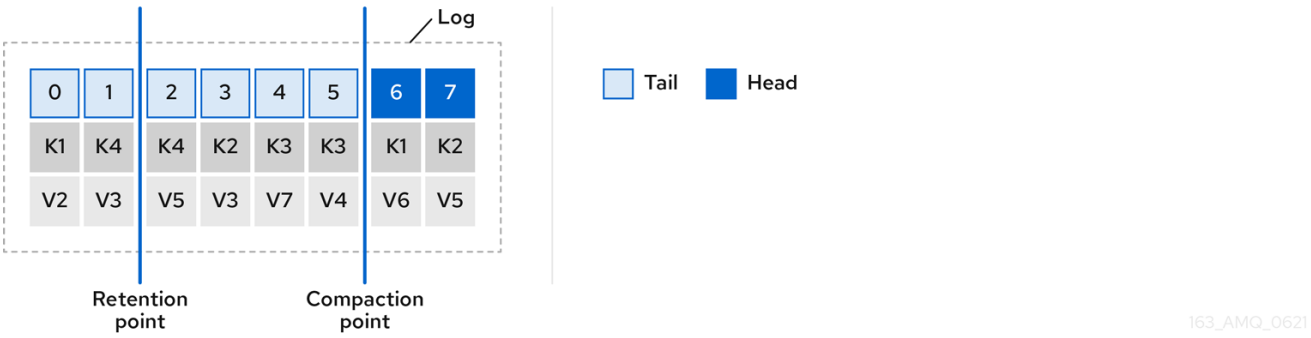
即使记录被删除，记录也会保留原始偏移。因此，尾部可能有非连续偏移。当消耗在尾部中不再可用的偏移时，会找到具有下一个高偏移量的记录。

图 12.2. 紧凑后日志



如果您只选择紧凑的策略，您的日志仍然可以变得任意大。在这种情况下，您可以将策略设置为压缩和删除 日志。如果您选择压缩和删除，则首先压缩日志数据，从而删除日志头带有键的记录。之后，在删除日志保留阈值之前发布的数据。

图 12.3. 日志保留点和压缩点



您需要设置以毫秒为单位检查清理的频率：

```
# ...
log.retention.check.interval.ms=300000
# ...
```

调整与日志保留设置相关的日志保留检查间隔。较小的保留大小可能需要更频繁地检查。

清理的频率应该足以管理磁盘空间，但经常影响某个主题的性能。

您还可以以毫秒为单位设置一个时间，以便在没有要清理的日志时将清理干净到备用中：

```
# ...
log.cleaner.backoff.ms=15000
# ...
```

如果您选择删除旧的日志数据，您可以在清除数据前设置一个毫秒来保留删除的数据：

```
# ...
log.cleaner.delete.retention.ms=86400000
# ...
```

删除的数据保留期限给予了时间，可以注意到数据在不可删除之前已丢失。

若要删除与特定密钥相关的所有消息，制作者可以发送 tomb stone 消息。am bstone 具有 null 值，充当标记来告知消费者删除其值。紧凑后，只保留了 tombstone，它必须持续足够长的时间，使消费者能够知道该消息被删除。删除旧消息后，没有值，也会从分区中删除 tombstone 密钥。

12.7.1.8. 管理磁盘使用率

还有许多其他与日志清理相关的配置设置，但特别重要的是内存分配。

deduplication 属性指定在所有日志清理线程中清理的总内存。您可以设置通过缓冲区负载因数使用的内存百分比的上限。

```
# ...
log.cleaner.dedupe.buffer.size=134217728
log.cleaner.io.buffer.load.factor=0.9
# ...
```

每个日志条目都使用正好 24 字节，因此您可以计算一次缓冲区可以处理的日志条目数量，并相应地调整设置。

如果您希望减少日志清理时间，请考虑增加日志清理线程数量：

```
# ...
log.cleaner.threads=8
# ...
```

如果您遇到 100% 磁盘带宽使用情况的问题，您可以限制日志清理 I/O，根据执行操作的磁盘功能，读写操作的总和小于指定的双值：

```
# ...
log.cleaner.io.max.bytes.per.second=1.7976931348623157E308
# ...
```

12.7.1.9. 处理大消息大小

消息的默认批处理大小为 1MB，这是大多数用例中最大吞吐量的最佳选择。如果有足够的磁盘容量，Kafka 能够以更低的吞吐量来容纳较大的批处理。

处理较大的消息的大小有四种方式：

1. 生产者侧消息压缩 将压缩的消息写入日志中。
2. 基于参考的消息传递仅发送对消息值中某个其他系统中存储数据的引用。
3. 内联消息传递将消息分割为使用相同键的块，然后通过 Kafka Streams 等流处理器将其组合在输出上。
4. 为处理更大消息大小而构建的代理和制作者/消费者客户端应用配置.

建议使用基于参考的消息和消息压缩选项，并覆盖大多数情况。对于这些选项，必须小心谨慎以避免引入性能问题。

生产者侧压缩

对于制作者配置，您可以指定一个 compression.type （如 Gzip），然后应用到生产者生成的批量数据。使用代理配置 compression.type=producer 时，代理会保留制作者使用的任何压缩。每当生产者和主题压缩不匹配时，代理必须在将批处理附加到日志中，这会影响代理性能。

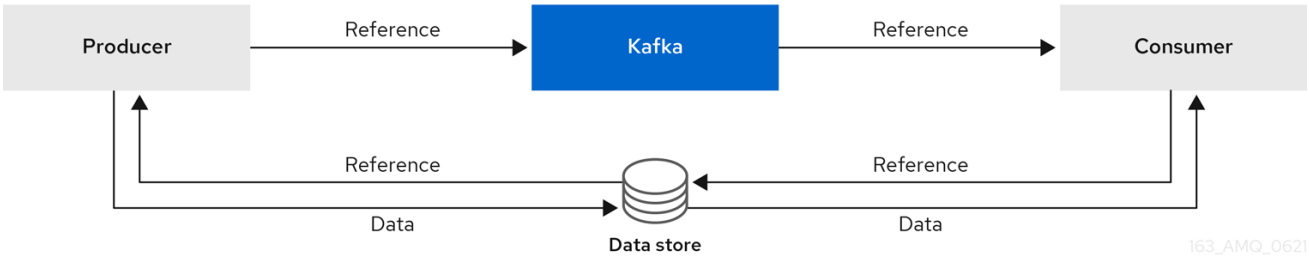
压缩还会增加制作者的额外处理开销和消费者的解压缩开销，但在批量中包含更多数据，因此在消息数据压缩不错时，通常对吞吐量很有用。

将制作者侧压缩与批处理大小的微调相结合，促进最佳吞吐量。使用指标有助于测量所需的平均批处理大小。

基于参考的消息传递

当您不知道消息的大程度时，基于参考的消息传递对于数据复制非常有用。外部数据存储必须快速、持久且高度可用，此配置才能发挥作用。将数据写入数据存储，并返回对数据的引用。制作者发送一条包含对 Kafka 引用的消息。使用者从消息中获取引用，并使用它来从数据存储中获取数据。

图 12.4. 基于参考的消息传递流



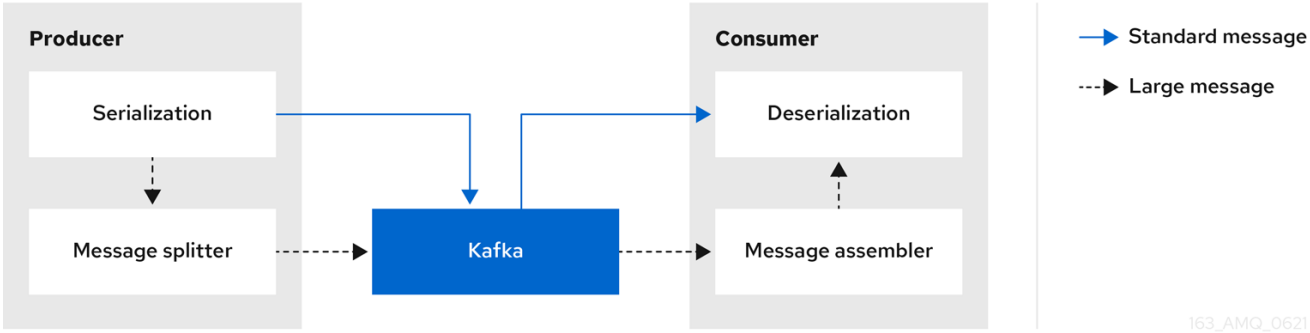
由于消息传递需要更多行程，端到端延迟会增加。这种方法的另一个显著缺点是，在清理 Kafka 消息时，没有自动清理外部系统中的数据。混合方法是仅向数据存储直接发送大型消息，并直接处理标准大小的消息。

内联消息传递

内联消息传递很复杂，但开销不会依赖于基于参考的消息传递等外部系统。

如果消息太大，产生客户端应用必须按顺序进行序列化，然后对数据进行块。然后，制作者使用 Kafka ByteArraySerializer 或类似的方法在发送前再次对每个块进行序列化。使用者跟踪消息和缓冲块，直到消息完整消息为止。消耗的客户端应用程序接收区块，这些区块在反序列化之前组装。根据每组区块消息的第一块或最后一个块的偏移，将完整消息传送到正在使用的应用的其余部分。针对偏移元数据检查成功发送完整消息，以避免在重新平衡期间重复。

图 12.5. 内联消息传递流



由于需要缓冲，内联消息传递在消费者一侧具有性能开销，特别是在并行处理一系列大型消息时。大型消息的块可能会相互交集，因此，如果缓冲区中另一条大消息的块不完整，则始终无法在消息的所有块被使用时提交。因此，缓冲区通常由持久消息区块或实施提交逻辑来支持。

配置以处理较大的信息

如果无法避免较大的消息，并且为了避免在消息流的任意点上阻止，您可以增加消息限值。为此，可在主题级别上配置 message.max.bytes，为各个主题设置最大记录批处理大小。如果您在代理级别设置了 message.max.bytes，则允许所有主题显示较大的信息。

代理将拒绝任何大于使用 message. max.bytes 设置的限制值的消息。生产者(max.request.size)和使用者(message.max.bytes)的缓冲区大小必须能够容纳较大的消息。

12.7.1.10. 控制消息数据的日志清除

日志刷新属性控制缓存的消息数据的定期写入磁盘。调度程序以毫秒为单位指定日志缓存中的检查频率：

```
# ...
log.flush.scheduler.interval.ms=2000
# ...
```

您可以根据信息保存内存的最大时间以及日志中的信息在写入磁盘前的最大信息数来控制刷新的频率：

```
# ...
log.flush.interval.ms=50000
log.flush.interval.messages=100000
# ...
```

清空之间的等待包括检查的时间和执行清除前的指定间隔。增加清空的频率可能会影响吞吐量。

通常，建议不要设置明确的清空阈值，并让操作系统使用其默认设置执行后台刷新。分区复制比写入单个磁盘的数据持久性要高，因为故障代理可以从其同步副本中恢复。

如果您使用应用程序清空管理，如果使用更快的磁盘，则可能最好设置较低的清空阈值。

12.7.1.11. 对可用性进行分区重新平衡

分区可以在代理之间复制，以实现容错。对于给定的分区，一个代理被选为领导，并处理所有生成的请求（写入到日志）。在出现领导故障时，其他代理的分区跟踪者复制分区领导者的分区数据，以实现数据可靠性。

跟随者通常不为客户端服务，虽然 机架 配置 允许使用者在 Kafka 集群跨越多个数据中心时消耗来自最接近的副本的消息。跟随者只从分区领导机复制消息，并在领导机失败时允许恢复。恢复需要一个同步跟踪器。跟随者通过向领导发送获取请求来保持同步，后者按顺序向跟随者返回消息。如果跟踪者发现了领导上最近提交的邮件，则被视为同步。领导机通过查看跟随者请求的最后偏移来检查这一点。除非允许未清理的领导选举，否则无法同步跟进者作为当前的领导机 故障。

您可以调整跟随者被视为不同步前的滞后时间：

```
# ...
replica.lag.time.max.ms=30000
# ...
```

滞后时间为将消息复制到所有同步副本的时间施加了上限，以及生产者必须等待确认的时间。如果跟踪器无法发出获取请求并在指定滞后时间内跟上最新的消息，则会从同步副本中删除它。您可以缩短检测失败副本的滞后时间，但这样做可能会增加无用同步的后续者的数量。正确的滞后时间值取决于网络延迟和代理磁盘带宽。

当领导分区不再可用时，会选择同步内的副本之一作为新领导。分区副本列表中的第一个代理称为 首选 的领导。默认情况下，Kafka 会根据对领导发行的定期检查来为自动分区领导重新平衡启用 Kafka。也就是说，Kafka 检查首选的领导是否是 当前的 领导机。重新平衡可确保领先者在代理和代理之间均匀分布，不会过载。

您可以使用 AMQ Streams 的 Cruise Control 来找出副本分配，以便代理在整个集群中平均平衡负载。其计算考虑了领导者和追随者所经历的不同负载。失败的领导机会影响 Kafka 集群的平衡，因为剩余的代理会获得前导额外分区的额外工作。

要使 Cruise Control 发现的分配真正达到平衡，分区必须要由首选领导领导。Kafka 可以自动确保使用首选的领导机（在可能的情况下），根据需要更改当前的领导机。这样可确保集群保持 Cruise Control 找到的均衡状态。

您可以在触发重新平衡前，以秒为单位控制重新平衡检查的频率和代理允许的最大发生率。

```
#...
auto.leader.rebalance.enable=true
leader.imbalance.check.interval.seconds=300
leader.imbalance.per.broker.percentage=10
#...
```

代理的领导地位百分比是代理作为当前领导者的分区数量与首选领导者分区数量之间的比例。您可以将百分比设置为零，以确保始终选中首选领导者，假设他们同步。

如果重新平衡的检查需要更多控制，您可以禁用自动重新平衡。然后，您可以选择何时使用 kafka-leader-election.sh 命令行工具触发重新平衡。

注意

随 AMQ Streams 提供的 Grafana 仪表板显示没有活跃领导的复制分区和分区的指标。

12.7.1.12. 未清理领导选举机制

同步副本的领导选举被视为干净，因为它保证不会丢失数据。这是默认情况下会发生的情况。但是，如果没有可以承担领导地位的同步副本，情况会怎样？或许 ISR（同步副本）仅在领导磁盘终止时包含领导机。如果没有设置最少的同步副本数，且其硬盘驱动器出现故障时没有跟随分区领导器同步，则数据已经丢失。不仅 不能这种情况，而且不能选择新的领导，因为不存在同步的追随者。

您可以配置 Kafka 如何处理领导故障：

```
# ...
unclean.leader.election.enable=false
# ...
```

Unclean leader 选举默认被禁用，这意味着不同步的副本不能成为领导。使用干净的领导选举机制时，如果没有其他代理在旧领导丢失时处于 ISR 中，Kafka 会等待该领导者重新在线，然后再写入或读取消息。unclean leader 选举机制意味着不同步的副本可能会成为领导者，但您会面临丢失消息的风险。您选择的选择取决于您的需求是否有利于可用性或持久性。

您可以覆盖主题级别上特定主题的默认配置。如果无法承担数据丢失的风险，请保留默认配置。

12.7.1.13. 避免不必要的消费者组重新平衡

对于加入新消费者组的消费者，您可以添加延迟，以避免不必要的重新平衡到代理：

```
# ...
group.initial.rebalance.delay.ms=3000
# ...
```

延迟是协调会等待会员加入的时间。延迟时间越长，所有成员越有可能加入并避免重新平衡。但是，这一延迟也会阻止该组在句点结束前被消耗掉。

其它资源

- 使用 Kafka Static Quota 插件设置代理的限制

12.7.2. Kafka 生成器配置调整

使用基本制作者配置，以及为特定用例量身定制的可选属性。

调整配置以最大化吞吐量可能会增加延迟，反之亦然。您将需要试验并调优制作者配置，以获得所需的平衡。

12.7.2.1. 基本制作者配置

每个制作者都需要连接和序列化程序属性。通常而言，最好是添加客户端 ID 以进行跟踪，并对制作者使用压缩来减少请求中的批处理大小。

在基本制作者配置中：

- 无法保证分区中消息的顺序。
- 确认到达代理的消息不能保证持久性。

基本制作者配置属性

```
# ...
bootstrap.servers=localhost:9092 1
key.serializer=org.apache.kafka.common.serialization.StringSerializer 2
value.serializer=org.apache.kafka.common.serialization.StringSerializer 3
client.id=my-client 4
compression.type=gzip 5
# ...
```

1

（必需）告诉制作者使用 Kafka 代理的 host:port bootstrap 服务器地址连接到 Kafka 集群。制作者使用该地址来发现和连接集群中的所有代理。在服务器停机时使用逗号分隔列表来指定两个或三个地址，但不需要提供集群中所有代理的列表。

2

（必需）在将每条消息的密钥发送到代理前将其转换为字节。

3

（必需）在将每个消息发送到代理前将每条消息的值转换为字节。

4

（可选）客户端的逻辑名称，用于日志和指标来标识请求的来源。

5

（可选）压缩消息的编码器（发送并可能以压缩格式存储），然后在到达消费者时解压缩。压缩对于提高吞吐量和减少存储负载非常有用，但可能不适用于低延迟应用程序，因为压缩或解压成本可能过高。

12.7.2.2. 数据持久性

您可以使用消息发送确认，应用更大的数据持久性，以最大程度降低消息丢失的可能性。

```
# ...
acks=all 1
# ...
```

1

指定 acks=all 会强制分区领导将消息复制到一定数量的跟随者，然后确认消息请求已成功收到。由于附加检查，acks=all 会增加生产者发送消息和接收确认之间的延迟。

在将确认发送到生产者之前，需要将消息附加至其日志中的代理数量由主题的 min.insync.replicas 配置决定。典型的起点是将主题复制因数为 3，其他代理上有两个内联副本。在这种配置中，如果单个代

理不可用，生产者可以继续不受影响。如果第二个代理不可用，生产者将不会收到确认并且无法生成更多消息。

支持 acks=all 的主题配置

```
# ...
min.insync.replicas=2 (1)
# ...
```

(1) 使用 2 内同步的副本。默认值为 1。

注意

如果系统失败，则缓冲区中存在不正确的数据丢失的风险。

12.7.2.3. 订购交付

幂等制作者避免重复，因为消息只发送一次。为消息分配了 ID 和序列号，以确保传送顺序，即使出现故障也是如此。如果您使用 acks=all 来实现数据一致性，则启用幂等性对有序交付有利。

使用幂等方式订购交付

```
# ...
enable.idempotence=true (1)
max.in.flight.requests.per.connection=5 (2)
acks=all (3)
retries=2147483647 (4)
# ...
```

(1)

设置为 true，以启用幂等制作者。

2

通过幂等发送，即时请求数可能大于 1，同时仍然提供消息排序保证。默认值为 5 个 in-flight 请求。

3

将一个cks 设置为 所有.

4

设置重新发送失败消息请求的尝试次数。

如果您没有由于性能成本而使用 acks=all 和 幂等性，请将待机（未确认）请求数设置为 1 以保持排序。否则，只有在 Message- B 已写入代理后 Message- A 可能无法成功。

在没有幂等的情况下订购交付

```
# ...
enable.idempotence=false 1
max.in.flight.requests.per.connection=1 2
retries=2147483647
# ...
```

1

设置为 false，以禁用幂等制作者。

2

将 in-flight 请求数设置为正好 1。

12.7.2.4. 可靠性保证

仅对写入单个分区一次，Idempotence 非常有用。事务处理与幂等性一起使用时，允许在多个分区间

只写入一次。

事务可确保使用相同事务 ID 的消息只生成一次，并且将 所有 消息都成功写入到对应的日志中，或者其中任何消息 都不是.

```
# ...
enable.idempotence=true
max.in.flight.requests.per.connection=5
acks=all
retries=2147483647
transactional.id=UNIQUE-ID (1)
transaction.timeout.ms=900000 (2)
# ...
```

(1) 指定唯一的事务 ID。

(2) 在返回超时错误前，设置以毫秒为单位进行事务的最大允许时间。默认值为 900000 或 15 分钟。

transaction .id 的选择对于保持事务保证非常重要。每个事务 id 都应该用于一组唯一的主题分区。例如，这可以通过外部映射主题分区名称到事务 id 来实现，或者通过使用避免冲突的功能计算主题分区名称中的事务 ID。

12.7.2.5. 优化吞吐量和延迟

通常，系统的要求是满足给定延迟内一定比例消息的特定吞吐量目标。例如，以每秒 500,000条消息为目标，95% 的消息会在 2 秒内得到确认。

生产者的消息传递语义（消息排序和持久性）很可能根据您的应用程序的要求进行定义。例如，您可能没有选项在不破坏某些重要属性的情况下使用 acks=0 或 acks=1，或者无法保证应用程序提供。

Broker 重新启动对高百分比统计数据有显著影响。例如，在很长一段时间内，99 百分点延迟由围绕代理重启的行为占据主导地位。在设计基准测试时，或比较基准测试与生产中显示的性能数字时，需要考虑这一点。

根据您的目标，Kafka 提供了多个配置参数和技术来调节吞吐量和延迟的性能。

消息批处理（linger.ms 和 batch.size）

消息批处理会延迟发送消息，希望将发送更多目标为同一代理的消息，允许它们批处理到单个生成请求。批处理是在高吞吐量时返回的更高延迟之间的妥协。基于时间的批处理使用 linger .ms 配置，而基于大小的批处理则使用 batch.size 配置。

压缩（压缩.type）

消息压缩增加了制作者延迟（CPU 时间用于压缩消息），但会更小（可能进行磁盘写入），这可以提高吞吐量。压缩是否必要，以及要使用的最佳压缩程度取决于所发送的消息。压缩发生在调用 KafkaProducer.send（）的 线程上，因此如果此方法的延迟与您需要使用更多线程的应用程序相关。

pipelining(max.in.flight.requests.per.connection)

pipelining 意味着在收到对上一个请求的响应前发送更多请求。通常，更多流水线意味着更好的吞吐量，最高是一个阈值，达到其他效果（例如更糟糕的批处理）开始消除对吞吐量的影响。

降低延迟

当您的应用程序调用 KafkaProducer.send（） 时，消息为：

- *由任何拦截器处理*
- *serialized*
- *分配给分区*
- *已压缩*
- *添加到每个分区队列中的批量消息*

send（） 方法返回的时间点。因此, send（） 的时间由以下方法决定：

- *拦截器、序列程序和分区器花费的时间*

- 使用的压缩算法
- 等待缓冲区用于压缩所需的时间

批处理将保留在队列中，直到出现以下情况之一：

- 批处理已满（根据 batch.size）
- linger .ms 引入的延迟已过
- 发送方即将向同一代理发送其他分区的消息批处理，也可以添加此批处理
- 生产者被清空或关闭

查看批处理和缓冲的配置，以减轻 send（） 阻止对延迟的影响。

```
# ...
linger.ms=100 1
batch.size=16384 2
buffer.memory=33554432 3
# ...
```

1 linger 属性添加毫秒的延迟，以便在请求中累积和发送更大规模的消息。默认值为 0'。

2 如果使用最大 批处理.size，则在达到最大值时发送请求，或者消息已排队的时间超过 linger .ms （以较快者为准）。添加延迟可让批处理积累消息到批处理大小。

3 缓冲区的大小必须至少与批处理大小相同，并且能够适应缓冲区、压缩和内向请求。

增加吞吐量

通过调整消息传输和完成发送请求前等待的最长时间，提高消息请求的吞吐量。

您还可以通过编写自定义分区程序来替换默认分区，将消息定向到指定分区。

```
# ...
delivery.timeout.ms=120000 1
partitioner.class=my-custom-partitioner 2

# ...
```

1

等待完整发送请求的最长时间，以毫秒为单位。您可以将值设为 MAX_LONG，以将值委派给 Kafka 重试次数的未定义次数。默认值为 120000 或 2 分钟。

2

指定自定义分区器的类名称。

12.7.3. Kafka 使用者配置调整

使用基本使用者配置，以及根据特定用例量身定制的可选属性。

调优您的消费者时，您的主要顾虑是确保它们能高效地应对被窃取的数据量。与制作者调优一样，准备好进行增量更改，直到消费者按预期工作。

12.7.3.1. 基本使用者配置

每个消费者都需要连接和反序列化器属性。通常，最好添加客户端 ID 进行跟踪。

在使用者配置中，无论后续配置如何：

- 消费者从给定的偏移获取并按顺序使用消息，除非偏差被更改为跳过或重新读取消息。
- 代理不知道消费者是否处理了响应，即使对 Kafka 提交偏移也是如此，因为偏移可能会发送到集群中的不同代理。

基本使用者配置属性

```
# ...
bootstrap.servers=localhost:9092 1
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer 2
value.deserializer=org.apache.kafka.common.serialization.StringDeserializer 3
client.id=my-client 4
group.id=my-group-id 5
# ...
```

1

（必需）告诉使用者使用 Kafka 代理的 host:port bootstrap 服务器地址连接到 Kafka 集群。使用者使用该地址来发现并连接到集群中的所有代理。在服务器停机时使用逗号分隔列表来指定两个或三个地址，但不需要提供集群中所有代理的列表。如果您使用负载均衡器服务公开 Kafka 集群，则只需要该服务的地址，因为负载均衡器处理其可用性。

2

（必需）Deserializer 将从 Kafka 代理获取的字节转换为消息密钥。

3

（必需）Deserializer 将从 Kafka 代理获取的字节转换为消息值。

4

（可选）客户端的逻辑名称，用于日志和指标来标识请求的来源。id 也可用于根据处理时间配额对消费者进行限流。

5

（条件） 用户需要 组 ID 才能加入消费者组。

12.7.3.2. 使用消费者组扩展数据消耗

消费者组共享一个由一个或多个生产者从给定主题生成的大型数据流。用户通过 group.id 属性分组，允许消息分散到不同成员中。组中的一个消费者被选为领导，并决定如何将该分区分配给组中的使用者。每个分区只能分配给一个消费者。

如果您还没有足够的消费者作为分区，可以通过添加具有相同 group.id 的更多消费者实例来扩展数据消耗。将更多的消费者添加到组中，超过现有分区，但这意味着如果一个分区停止工作，就表示待机使用者处于待机状态。如果您能够以更少的消费者达到吞吐量目标，就可以节省资源。

同一消费者组中的消费者发送偏移提交和心跳到同一代理。因此组中的消费者数量越多，代理上的请求负载就越大。

```
# ...
group.id=my-group-id 1
# ...
```

1

使用组 ID 将使用者添加到消费者组中。

12.7.3.3. 消息排序保证

Kafka 代理从客户接收请求，要求代理从主题、分区和偏移位置列表中发送消息。

用户会按照提交至代理的顺序以单个分区中观察消息，这意味着 Kafka 仅在 单一分区中为消息提供排序保证。相反，如果消费者使用来自多个分区的消息，则使用者观察到的不同分区中消息的顺序不一定反映它们的发送顺序。

如果您需要严格排序一个主题的消息，请为每个使用者使用一个分区。

12.7.3.4. 优化吞吐量和延迟

控制客户端应用调用 KafkaConsumer.poll（） 时返回的消息数量。

使用 fetch.max.wait.ms 和 fetch.min.bytes 属性来增加消费者从 Kafka 代理获取的最小数据量。基于时间的批处理使用 fetch.max.wait.ms 配置，而基于大小的批处理则使用 fetch.min.bytes 配置。

如果消费者或代理中的 CPU 使用率较高，则可能是因为消费者的请求太多。您可以调整 fetch.max.wait.ms 和 fetch.min.bytes 属性，以便在较大的批处理中发送请求和消息。通过调整较高的吞吐量，可以降低延迟成本。如果产生的数据量较低，您也可以调整更高的值。

例如，如果您将 fetch.max.wait.ms 设置为 500ms，并且 fetch.min.bytes 设为 16384 字节，则当 Kafka 收到来自消费者的获取请求时，它将在达到任一阈值的第一个阈值时做出响应。

相反，您可以调整 fetch.max.wait.ms 和 fetch.min.bytes 属性来降低端到端延迟。

```
# ...
fetch.max.wait.ms=500 1
fetch.min.bytes=16384 2
# ...
```

1

代理在完成获取请求前将等待的最长时间，以毫秒为单位。默认值为 500 毫秒。

2

如果使用的最小批处理大小（以字节为单位），则会在达到最小值时发送请求，或者消息排队的时间超过 fetch.max.wait.ms （以更早者为准）。添加延迟可让批处理积累消息到批处理大小。

通过增大获取请求大小来降低延迟

使用 fetch.max.bytes 和 max.partition.fetch.bytes 属性来增加消费者从 Kafka 代理获取的最大数据量。

fetch.max.bytes 属性设置一次从代理获取的数据量上限，以字节为单位。

max.partition.fetch.bytes 设置每个分区返回的数据量的最大字节数，必须始终大于 broker 或 max.message.bytes 配置中设置的字节数。

客户端可消耗的最大内存量计算如下：

```
NUMBER-OF-BROKERS * fetch.max.bytes and NUMBER-OF-PARTITIONS *
max.partition.fetch.bytes
```

如果内存用量可以容纳它，您可以增加这两个属性的值。通过在每个请求中允许更多数据，可以提高延迟，因为获取请求的数量较少。

```
# ...
fetch.max.bytes=52428800 1
```

```
max.partition.fetch.bytes=1048576 (2)
# ...
```

(1) 为获取请求返回的最大数据量，以字节为单位。

(2) 每个分区返回的最大数据量，以字节为单位。

12.7.3.5. 在提交偏移时避免数据丢失或重复

Kafka 自动提交机制 允许使用者自动提交消息偏移。如果启用，消费者将以 5000ms 间隔提交从轮询代理收到的偏移。

自动提交机制很方便，但会带来数据丢失和复制风险。如果使用者已获取并转换了大量消息，但执行自动提交时，系统会对消费者缓冲区中已处理的消息崩溃，该数据将会丢失。如果系统在处理消息后崩溃，但在执行自动使用前，数据会在重新平衡后在另一个消费者实例上重复。

仅当在下一次轮询到代理或消费者关闭之前处理所有消息时，自动使用才可以避免数据丢失。

为最大程度减少数据丢失或重复的可能性，您可以将 enable.auto.commit 设置为 false，并开发客户端应用程序，使其对提交的偏移拥有更多控制权。或者，您可以使用 auto.commit.interval.ms 减少提交之间的间隔。

```
# ...
enable.auto.commit=false (1)
# ...
```

(1) 自动提交设为 false，以提供对提交偏移的更多控制。

通过将 enable.auto.commit 设置为 false，您可以在执行了 所有 处理并且消息已被使用后提交偏移。例如，您可以将应用程序设置为调用 Kafka commitSync 和 commit Async 提交 API。

commitSync API 在从轮询返回的消息批处理中提交偏移。完成批处理中的所有消息后，您将调用 API。如果使用 commitSync API，则应用不会轮询新消息，直到提交批处理中的最后一个偏移。如果这会对吞吐量造成负面影响，您可以降低提交的频率，也可以使用 commitAsync API。commitAsync API

不等待代理响应提交请求，但可能会在重新平衡时造成更多的重复。种常见的做法是将应用中的两个提交 API 与刚才在关闭使用者或重新平衡之前使用的 commitSync API 相结合，以确保最终提交成功。

12.7.3.5.1. 控制事务性消息

考虑在制作者一侧使用事务 id 和启用幂等性(enable.idempotence=true)来保证准确交付。在消费者方面，您可以使用 isolated .level 属性来控制消费者如何读取事务性消息。

isolated .level 属性有两个有效值：

- read_committed
- read_uncomcommit （默认）

使用 read_comsigned 来确保 只有提交的事务消息会被使用者读取。但是，这会导致端到端延迟增加，因为在代理写入了记录事务结果的事务标记（已承诺 或中止）之前，消费者将无法返回消息 。

```
# ...
enable.auto.commit=false
isolation.level=read_committed (1)
# ...
```

(1) 设置为 read_com commit，以使使用者只读取提交的邮件。

12.7.3.6. 从失败中恢复，以避免数据丢失

使用 session.timeout.ms 和 heartbeat.interval.ms 属性配置时间，以检查并从消费者组中的消费者故障中恢复。

session.timeout.ms 属性指定使用者组中用户的最大时间（毫秒）可以不与代理联系，然后才能被视为不活动，并在组中的活动消费者之间触发 重新平衡。当组重新平衡时，这些分区将重新分配给组的成员。

heartbeat.interval.m s 属性指定 心跳 互相检查之间的间隔，以毫秒为单位表示消费者活跃并连接。heartbeat 间隔必须小于会话超时间隔，通常为第三个。

如果您将 session.timeout.ms 属性设置为 less，则之前检测到失败消费者，并且重新平衡可以更快地进行。但是，请不要设置超时时间，以便代理无法及时收到心跳，并触发不必要的重新平衡。

减少心跳间隔降低了意外重新平衡的可能性，但更频繁的心跳会增加对代理资源的开销。

12.7.3.7. 管理偏移策略

使用 auto.offset.reset 属性控制消费者在未提交偏移时的行为方式，或者不再有效或删除已提交的偏移。

假设您第一次部署使用者应用，并且它从现有主题读取消息。由于这是第一次使用 group.id 时，__consumer_offsets 主题不包含此应用的任何偏移信息。新应用可以从日志开始时开始处理所有现有消息，或者仅处理新消息。默认重置值为 latest，它从分区末尾开始，因此表示丢失了一些消息。为避免数据丢失，但会增加处理量，请将 auto.offset.reset 设置为从分区 开始时开始。

另请考虑使用 early 选项 避免在为代理配置的偏移保留周期（偏移.retention.分钟）终止时丢失信息。如果消费者组或独立消费者不活跃，且在保留周期内未提交偏移，则之前提交的偏移会从 __consumer_offset 中删除。

```
# ...
heartbeat.interval.ms=3000 1
session.timeout.ms=10000 2
auto.offset.reset=earliest 3
# ...
```

1

根据预期的重新平衡，调整心跳间隔越低。

2

如果 Kafka 代理在超时期限到期前未收到 heartbeat，则消费者会从消费者组中移除，并启动重新平衡。如果代理配置具有 group.min.session.timeout.ms 和 group.max.session.timeout.ms，会话超时值必须在这个范围内。

3

设置为 更早的 以返回到分区的开头，并在未提交偏移时避免数据丢失。

如果单个获取请求中返回的数据量较大，则使用者处理数据之前可能会发生超时。在这种情况下，您可以降低 max.partition.fetch.bytes 或增加 session.timeout.ms。

12.7.3.8. 最小化重新平衡的影响

在组中活跃使用者之间重新平衡分区是以下时间：

- 消费者提交偏移
- 要成立的新消费者组
- 将分区分配给组成员的组领导
- 组中的消费者接收其分配并开始获取

显然，这个过程会增加服务的停机时间，特别是在客户组群集滚动重启期间重复发生时。

在这种情况下，您可以使用 静态成员资格 的概念来减少重新平衡的数量。重新平衡使用者组成员之间均匀分配主题分区。静态成员资格使用持久性，以便在会话超时后在重启期间识别使用者实例。

用户组协调可以使用使用 group. instance.id 属性指定的唯一 id 来识别新的消费者实例。在重启期间，会为消费者分配一个新成员 ID，但作为静态成员，它将继续使用相同的实例 ID，并分配相同的主题分区。

如果使用者应用程序至少没有调用每个 max.poll.interval.ms 毫秒，则消费者将被视为失败，从而导致重新平衡。如果应用无法及时处理轮询返回的所有记录，您可以使用 max.poll.interval.ms 属性来指定来自消费者的新消息轮询间隔（以毫秒为单位）。或者，您可以使用 max.poll.records 属性设置从消费者缓冲区返回的记录数上限，允许您的应用程序处理 max.poll.interval.ms 限制内的记录数量。

```
# ...
group.instance.id=_UNIQUE-ID_ 1
max.poll.interval.ms=300000 2
max.poll.records=500 3
# ...
```

1

2

设置检查消费者是否继续处理消息的时间间隔。

3

设置从使用者返回的已处理记录数。

12.8. 卸载 AMQ 流

此流程描述了如何卸载 AMQ Streams 并删除与部署相关的资源。

先决条件

要执行此步骤，请识别专门为部署创建的资源，并从 AMQ Streams 资源引用。

此类资源包括：

- secret（Custom CA 和证书、Kafka Connect secret 和其他 Kafka secret）
- 日志记录 ConfigMap（类型为 外部）

这些是 Kafka、Kafka Connect、KafkaConnect S2I、KafkaMirrorMaker 或 KafkaBridge 配置引用的资源。

流程

1. 删除 Cluster Operator Deployment、相关的 CustomResourceDefinition 和 RBAC 资源：

```
oc delete -f install/cluster-operator
```

警告

删除 CustomResourceDefinitions 会导致相应自定义资源（Kafka、KafkaConnect、Kafka Connect S2I、Kafka MirrorMaker 或 Kafka Bridge）的垃圾回收以及依赖它们的资源（Deployments、StatefulSets 和其他依赖资源）的垃圾回收。

2. *删除您在先决条件中确定的资源。*

12.9. 常见问题解答

12.9.1. 与 Cluster Operator 相关的问题

12.9.1.1. 为什么我需要集群管理员特权才能安装 AMQ Streams？

要安装 AMQ Streams，您需要创建以下集群范围的资源：

- *自定义资源定义(CRD)指示 OpenShift 关于 AMQ Streams 专用资源，如 Kafka 和 KafkaConnect*
- *ClusterRole 和 ClusterRoleBindings*

集群范围的资源（不限定在特定的 OpenShift 命名空间内），通常需要 集群管理员 特权才能安装。

作为集群管理员，您可以检查要安装的所有资源（在 /install/ 目录中），以确保 ClusterRole 不会 授予不必要的特权。

安装后，Cluster Operator 会作为常规 Deployment 运行，因此具有访问 Deployment 的权限的任何标准（非 admin）OpenShift 用户都可以进行配置。集群管理员可向标准用户授予管理 Kafka 自定义资源所需的权限。

另请参阅：

- 为什么 Cluster Operator 需要创建 ClusterRoleBinding？
- 标准 OpenShift 用户是否可以创建 Kafka 自定义资源？

12.9.1.2. 为什么 Cluster Operator 需要创建 ClusterRoleBinding？

OpenShift 内置了 特权升级防御，这意味着 Cluster Operator 无法为其授予本身没有的特权，特别是，它无法在它无法访问的命名空间中授予此类权限。因此，Cluster Operator 必须具有它编配 的所有组件所需的权限。

Cluster Operator 需要能够授予访问权限，以便：

- Topic Operator 可以管理 KafkaTopics，方法是在操作器运行的命名空间中创建 Roles 和 RoleBindings
- User Operator 可以管理 KafkaUsers，方法是在 Operator 运行的命名空间中创建 Roles 和 RoleBindings
- AMQ Streams 通过创建一个 ClusterRoleBinding来 发现 节点的故障域

使用机架感知分区分配时，代理 pod 需要能够获取有关在其上运行的 节点 的信息，例如 Amazon AWS 中的 Availability Zone。节点 是一个集群范围的资源，因此只能通过 ClusterRoleBinding 而不是命名空间范围的RoleBinding 授予对它的访问权限 。

12.9.1.3. 标准 OpenShift 用户是否可以创建 Kafka 自定义资源？

默认情况下，标准 OpenShift 用户没有管理由 Cluster Operator 处理的自定义资源所需的权限。集群管理员可以使用 OpenShift RBAC 资源为用户授予必要的特权。

如需更多信息，请参阅 OpenShift 上部署和升级 AMQ 流指南中的指定 AMQ 流 管理员。

12.9.1.4. 在 日志中获取锁定 警告意味着什么？

对于每个集群，Cluster Operator 一次只能执行一个操作。Cluster Operator 使用锁定来确保同一集群永远不会运行两个并行操作。其他操作必须等到当前操作完成后才会释放锁定。

INFO

集群操作示例包括 集群创建、滚动更新、缩减 和 纵向扩展。

如果锁定的等待时间太长，操作会超时，并在日志中输出以下警告信息：

```
2018-03-04 17:09:24 WARNING AbstractClusterOperations:290 - Failed to acquire lock for
kafka cluster lock::kafka::myproject::my-cluster
```

根据 STRIMZI_FULL_RECONCILIATION_INTERVAL_MS 和 STRIMZI_
OPERATION_TIMEOUT_MS 的具体配置，此警告消息偶尔可能会出现，而不指示任何根本问题。在下一次定期协调中占用超时的操作，以便操作可以获取锁定并再次执行。

如果定期出现此消息，即使不应为给定群集运行其他操作，也可能会指示由于错误而未正确释放锁定。如果出现这种情况，请尝试重启 Cluster Operator。

12.9.1.5. 为什么在使用 TLS 连接 NodePort 时主机名验证失败？

目前，使用启用了 TLS 加密的 NodePorts 进行集群外访问不支持 TLS 主机名验证。因此，验证主机名的客户端将无法连接。例如，Java 客户端将失败，但以下例外：

```
Caused by: java.security.cert.CertificateException: No subject alternative names matching IP
address 168.72.15.231 found
 at sun.security.util.HostnameChecker.matchIP(HostnameChecker.java:168)
 at sun.security.util.HostnameChecker.match(HostnameChecker.java:94)
 at sun.security.ssl.X509TrustManagerImpl.checkIdentity(X509TrustManagerImpl.java:455)
 at sun.security.ssl.X509TrustManagerImpl.checkIdentity(X509TrustManagerImpl.java:436)
 at sun.security.ssl.X509TrustManagerImpl.checkTrusted(X509TrustManagerImpl.java:252)
 at
sun.security.ssl.X509TrustManagerImpl.checkServerTrusted(X509TrustManagerImpl.java:136)
 at sun.security.ssl.ClientHandshaker.serverCertificate(ClientHandshaker.java:1501)
 ... 17 more
```

要连接，您必须禁用主机名验证。在 Java 客户端中，您可以通过将配置选项 ssl.endpoint.identification.algorithm 设置为空字符串来完成此操作。

在使用属性文件配置客户端时，您可以以这种方式进行配置：

```
ssl.endpoint.identification.algorithm=
```

当直接在 Java 中配置客户端时，将配置选项设置为空字符串：

```
props.put("ssl.endpoint.identification.algorithm", "");
```

第 13 章 自定义资源 API 参考

13.1. 常见配置属性

常见配置属性应用到多个资源。

13.1.1. replicas

使用 replicas 属性来配置副本。

复制的类型取决于资源。

- KafkaTopic 使用复制因子配置 Kafka 集群中每个分区的副本数。
- Kafka 组件使用副本来配置部署中的 pod 数量，以提供更好的可用性和可扩展性。

注意

在 OpenShift 上运行 Kafka 组件时，可能不需要运行多个副本来获取高可用性。当部署组件的节点崩溃时，OpenShift 将自动将 Kafka 组件容器集重新调度到其他节点。但是，运行带有多个副本的 Kafka 组件可能会提供更快的故障转移时间，因为其他节点将会启动并运行。

13.1.2. bootstrapServers

使用 bootstrapServers 属性来配置 bootstrap 服务器列表。

bootstrap 服务器列表可以引用未在同一 OpenShift 集群中部署的 Kafka 集群。它们还可以引用 AMQ Streams 未部署的 Kafka 集群。

如果在同一 OpenShift 集群中，每个列表必须理想包含名为 CLUSTER-NAME -kafka-bootstrap 和端口号的 Kafka 集群 bootstrap 服务。如果由 AMQ Streams 部署，但在不同的 OpenShift 集群上，列表内容取决于用于公开集群的方法（路由、入口、节点端口或负载均衡器）。

当在由 AMQ Streams 管理的 Kafka 集群中使用 Kafka 时，您可以根据给定集群的配置指定

bootstrap 服务器列表。

13.1.3. ssl

使用三个允许的 ssl 配置选项，将特定的 密码套件 用于 TLS 版本。密码套件组合了用于安全连接和数据传输的算法。

您还可以配置 ssl.endpoint.identification.algorithm 属性来 启用或禁用主机名验证。

SSL 配置示例

```
# ...
spec:
  config:
    ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" 1
    ssl.enabled.protocols: "TLSv1.2" 2
    ssl.protocol: "TLSv1.2" 3
    ssl.endpoint.identification.algorithm: HTTPS 4
# ...
```

1 用于 TLS 的加密套件使用 ECDHE 密钥交换机制、RSA 身份验证算法、AES 批量机密算法和 SHA384 MAC 算法的组合。

2 启用 SSl 协议 TLSv1.2。

3 指定用于生成 SSL 上下文的 TLSv1.2 协议。允许的值有 TLSv1.1 和 TLSv1.2。

4 通过将 设置为 HTTPS 来启用主机名验证。空字符串将禁用验证。

13.1.4. trustedCertificates

通过 set tls 配置 TLS 加密，请使用 trustedCertificates 属性提供带有密钥名称的 secret 列表，并在其中以 X.509 格式存储证书。

您可以使用 Cluster Operator 为 Kafka 集群创建的 secret，或者创建自己的 TLS 证书文件，然后从文件创建 Secret ：

```
oc create secret generic MY-SECRET \
--from-file=MY-TLS-CERTIFICATE-FILE.crt
```

TLS 加密配置示例

```
tls:
  trustedCertificates:
    - secretName: my-cluster-cluster-cert
      certificate: ca.crt
    - secretName: my-cluster-cluster-cert
      certificate: ca2.crt
```

如果证书存储在同一个 secret 中，则可以多次列出证书。

如果要启用 TLS，但使用 Java 附带的默认公共证书颁发机构集合，您可以将 trustedCertificates 指定为空数组：

使用默认 Java 证书启用 TLS 示例

```
tls:
  trustedCertificates: []
```

有关配置 TLS 客户端身份验证的详情，请参考 KafkaClientAuthenticationTls schema 参考。

13.1.5. 资源

您可以为组件请求 CPU 和内存资源。limits 指定给定容器可以消耗的最大资源。

在 Kafka 资源中设置 Topic Operator 和 User Operator 的资源请求和限值。

使用 reources.requests 和 resources.limits 属性来配置资源请求和限值。

对于每个部署的容器，AMQ Streams 允许您请求特定资源并定义这些资源的最大消耗。

AMQ Streams 支持以下资源类型的请求和限值：

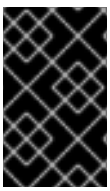
- cpu
- 内存

AMQ Streams 使用 OpenShift 语法来指定这些资源。

有关在 OpenShift 中管理计算资源的更多信息，请参阅为容器管理计算资源。

资源请求

请求指定要为给定容器保留的资源。保留资源可确保资源始终可用。



重要

如果资源请求针对的是 OpenShift 集群中可用的可用资源，则不会调度该容器集。

可以为一个或多个支持的资源配置请求。

资源请求配置示例

```
# ...
resources:
  requests:
    cpu: 12
    memory: 64Gi
# ...
```

资源限值

limits 指定给定容器可以消耗的最大资源。限制不是保留的，可能并不总是可用。容器只能在资源可用时才使用资源限制。资源限制应始终大于资源请求。

资源可以被配置为一个或多个支持的限制。

资源限制配置示例

```
# ...
resources:
  limits:
    cpu: 12
    memory: 64Gi
# ...
```

支持的 CPU 格式

支持 CPU 请求和限制，其格式如下：

- CPU 内核数作为整数（5 个 CPU 内核）或十进制（2.5 个 CPU 内核）。

- 数字或 millicpus / millicores(100m)，其中 1000 millicores 相同 1 个 CPU 内核。

CPU 单元示例

```
# ...
resources:
  requests:
    cpu: 500m
  limits:
    cpu: 2.5
# ...
```

注意

1 个 CPU 核心的计算能力可能因部署 OpenShift 的平台而异。

有关 CPU 规格的更多信息，请参阅 CPU 含义。

支持的内存格式

内存请求和限值以兆字节、千兆字节、兆字节和千兆字节为单位指定。

- 要指定内存（以 MB 为单位），请使用 M 后缀。例如 1000M。
- 要指定以 GB 为单位的内存，请使用 G 后缀。例如 1G ：
- 要以兆字节为单位指定内存，请使用 Mi 后缀。例如 1000Mi。
- 要以千兆字节为单位指定内存，请使用 Gi 后缀。例如 1Gi。

使用不同内存单元的资源示例

```
# ...
resources:
  requests:
    memory: 512Mi
  limits:
    memory: 2Gi
# ...
```

有关内存规格和其他支持单元的详情，请参阅 内存含义。

13.1.6. image

使用 image 属性配置组件使用的容器镜像。

建议仅在需要使用其他容器 registry 或自定义镜像的特殊情况下覆盖容器镜像。

例如，如果您的网络不允许访问 AMQ Streams 使用的容器存储库，您可以复制 AMQ Streams 镜像或从源构建它们。但是，如果配置的镜像与 AMQ Streams 镜像不兼容，它可能无法正常工作。

也可自定义容器镜像的副本，并用于调试。

您可以使用以下资源中的 image 属性来指定用于组件的容器镜像：

- Kafka.spec.kafka
- Kafka.spec.zookeeper
- Kafka.spec.entityOperator.topicOperator

- *Kafka.spec.entityOperator.userOperator*
- *Kafka.spec.entityOperator.tlsSidecar*
- *KafkaConnect.spec*
- *KafkaConnectS2I.spec*
- *KafkaMirrorMaker.spec*
- *KafkaMirrorMaker2.spec*
- *KafkaBridge.spec*

为 Kafka、Kafka Connect 和 Kafka MirrorMaker 配置 镜像 属性

Kafka、Kafka Connect（包括使用 S2I 支持的 Kafka Connect）和 Kafka MirrorMaker 支持多个 Kafka 版本。每个组件都需要自己的映像。在以下环境变量中配置了不同 Kafka 版本的默认镜像：

- *STRIMZI_KAFKA_IMAGES*
- *STRIMZI_KAFKA_CONNECT_IMAGES*
- *STRIMZI_KAFKA_CONNECT_S2I_IMAGES*
- *STRIMZI_KAFKA_MIRROR_MAKER_IMAGES*

这些环境变量包含 Kafka 版本及其相应镜像之间的映射。映射与 镜像 和 版本 属性一同使用：

- 如果在自定义资源中未给出镜像或版本，则版本将默认为 Cluster Operator 的默认 Kafka 版本，且镜像将是环境变量中与此版本对应的镜像。
- 如果为 image 提供了版本，则使用给定的镜像，且版本假定为 Cluster Operator 的默认 Kafka 版本。
- 如果提供了 version，但镜像没有指定，则使用与环境变量中给定版本对应的镜像。
- 如果同时给定了版本和映像，则使用给定的镜像。镜像被假定为包含给定版本的 Kafka 镜像。

可以在以下属性中配置不同组件的镜像和版本：

- 对于 spec.kafka.image 和 spec.kafka.version 中的 Kafka。
- 对于 Kafka Connect、Kafka Connect S2I 和 Kafka MirrorMaker，在 spec.image 和 spec.version 中。

警告

建议仅提供版本，不指定镜像属性。这降低了配置自定义资源时出错的机会。如果需要更改用于不同 Kafka 版本的镜像，最好配置 Cluster Operator 的环境变量。

在其他资源中配置 image 属性

对于其他自定义资源中的 image 属性，在部署期间将使用给定值。如果缺少 image 属性，将使用 Cluster Operator 配置中指定的镜像。如果 Cluster Operator 配置中没有定义镜像名称，则会使用默认值。

- *对于主题 Operator:*

  1. *在 Cluster Operator 配置中的 STRIMZI_DEFAULT_TOPIC_OPERATOR_IMAGE 环境变量中指定的容器镜像。*
  2. *registry.redhat.io/amq7/amq-streams-rhel7-operator:1.8.0 container image.*

- *对于 User Operator:*

  1. *在 Cluster Operator 配置中的 STRIMZI_DEFAULT_USER_OPERATOR_IMAGE 环境变量中指定的容器镜像。*
  2. *registry.redhat.io/amq7/amq-streams-rhel7-operator:1.8.0 container image.*

- *对于实体 Operator TLS sidecar:*

  1. *在 Cluster Operator 配置中的 STRIMZI_DEFAULT_TLS_SIDECAR_ENTITY_OPERATOR_IMAGE 环境变量中指定的容器镜像。*
  2. *registry.redhat.io/amq7/amq-streams-kafka-28-rhel7:1.8.0 container image.*

- *对于 Kafka Exporter:*

  1. *在 Cluster Operator 配置中的 STRIMZI_DEFAULT_KAFKA_EXPORTER_IMAGE 环境变量中指定的容器镜像。*
  2. *registry.redhat.io/amq7/amq-streams-kafka-28-rhel7:1.8.0 container image.*

- *对于 Kafka 网桥：*

  1.

在 Cluster Operator 配置中的 STRIMZI_DEFAULT_KAFKA_BRIDGE_IMAGE 环境变量中指定的容器镜像。

2. registry.redhat.io/amq7/amq-streams-bridge-rhel7:1.8.0 容器镜像。

- 对于 Kafka 代理初始化器：

1. 在 Cluster Operator 配置中的 STRIMZI_DEFAULT_KAFKA_INIT_IMAGE 环境变量中指定的容器镜像。

2. registry.redhat.io/amq7/amq-streams-rhel7-operator:1.8.0 container image.

容器镜像配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    image: my-org/my-image:latest
    # ...
  zookeeper:
    # ...
```

13.1.7. livenessProbe 和 readinessProbe 健康检查

使用 livenessProbe 和 readinessProbe 属性来配置 AMQ Streams 中支持的健康检查探测。

健康检查是定期测试，可验证应用的健康状况。当健康检查探测失败时，OpenShift 假定应用不健康，并尝试修复它。

有关探测的详情，请参阅 配置存活度和就绪度探测。

Live nessProbe 和 readinessProbe 均支持以下选项：

- initialDelaySeconds
- timeoutSeconds
- periodSeconds
- successThreshold
- failureThreshold

存活度和就绪度探测配置示例

```
# ...
readinessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
# ...
```

有关 livenessProbe 和 readinessProbe 选项的更多信息，请参阅 探测模式参考。

13.1.8. metricsConfig

使用 metricsConfig 属性来启用和配置 Prometheus 指标。

metricsConfig 属性包含对 ConfigMap 的引用，其中包含 Prometheus JMX 导出器的额外配置。AMQ Streams 支持使用 Prometheus JMX 导出器的 Prometheus 指标将 Apache Kafka 和

ZooKeeper 支持的 JMX 指标转换为 Prometheus 指标。

要在不进一步配置的情况下启用 Prometheus 指标导出，您可以在 metricsConfig.valueFrom.configMapKeyRef.key 下引用包含空文件的 ConfigMap。当引用空文件时，所有指标都会公开，只要它们没有被重命名。

带有 Kafka 指标配置的 ConfigMap 示例

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: my-configmap
data:
  my-key: |
    lowercaseOutputName: true
    rules:
    # Special cases and very specific rules
    - pattern: kafka.server<type=(.+), name=(.+), clientId=(.+), topic=(.+), partition=(.*)><>Value
      name: kafka_server_$1_$2
      type: GAUGE
      labels:
       clientId: "$3"
       topic: "$4"
       partition: "$5"
    # further configuration
```

Kafka 的指标配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    metricsConfig:
      type: jmxPrometheusExporter
      valueFrom:
        configMapKeyRef:
          name: my-config-map
          key: my-key
```

```
    # ...
zookeeper:
    # ...
```

启用指标数据后，它们会在端口 9404 上公开。

当资源中没有定义 metricsConfig（或已弃用的指标）属性时，Prometheus 指标会被禁用。

有关设置和部署 Prometheus 和 Grafana 的更多信息，请参阅 OpenShift 指南中的 Deploying 和 升级 AMQ Streams 中的 介绍 Metrics 到 Kafka。

13.1.9. jvmOptions

以下 AMQ Streams 组件在 Java 虚拟机(JVM)中运行：

- Apache Kafka
- Apache ZooKeeper
- Apache Kafka Connect
- Apache Kafka MirrorMaker
- AMQ Streams Kafka Bridge

要在不同的平台和架构中优化其性能，您可以在以下资源中配置 jvmOptions 属性：

- Kafka.spec.kafka

- *Kafka.spec.zookeeper*
- *KafkaConnect.spec*
- *KafkaConnectS2I.spec*
- *KafkaMirrorMaker.spec*
- *KafkaMirrorMaker2.spec*
- *KafkaBridge.spec*

您可以在配置中指定以下选项：

-Xms

JVM 启动时，最小初始分配堆大小.

-Xmx

最大堆大小.

-XX

JVM 的高级运行时选项.

javaSystemProperties

其他系统属性.

gcLoggingEnabled

启用垃圾收集器日志记录。

jvmOptions 的完整架构包括在 JvmOptions 架构引用 中。

注意

JVM 设置接受的单元（如 -Xmx 和 -Xms ）是对应镜像中 JDK Java 二进制文件接受的单元。因此，1g 或 1G 表示 1,073,741,824 字节和 Gi 不是有效的单元后缀。这与用于内存请求和限值 的单元不同，后者遵循 OpenShift 约定，1G 表示 1,000,000,000 字节，1Gi 表示 1,073,741,824 字节

-Xms 和 -Xmx 选项

用于 -Xms 和 -Xmx 的默认值取决于是否为容器配置 内存请求 限值。

- 如果有内存限制，JVM 的最小和最大内存将设置为与限值对应的值。
- 如果没有内存限制，JVM 的最小内存将设置为 128M。JVM 的最大内存未定义为允许内存根据需要增加。这是测试和开发环境中单节点环境的理想选择。

在设置 -Xmx 明确考虑以下内容前：

- JVM 的总内存使用量大约为最大堆的 4 [1]，如 -Xmx 配置。
- 如果在未设置适当的 OpenShift 内存限值的情况下设置 -Xmx，如果 OpenShift 节点遇到运行的其他 Pod 的内存压力，则可能会被终止。
- 如果设置 -Xmx 时不设置适当的 OpenShift 内存请求，则容器可能会调度到内存不足的节点。在这种情况下，容器不会立即启动，但如果 -Xms 设为 -Xm x，则不会立即崩溃，否则稍后将不立即崩溃。

建议您 ：

- 将内存请求和内存限值设置为相同的值
- 使用至少 4.5 的值请求 -Xmx

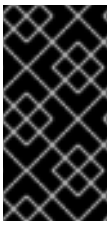
- 考虑将 -Xms 设置为与 -Xmx相同的值

在本例中，JVM 将 2 GiB（=2,147,483,648 字节）用于其堆。其总内存用量大约为 8GiB。

示例 -Xmx 和 -Xms 配置

```
# ...
jvmOptions:
  "-Xmx": "2g"
  "-Xms": "2g"
# ...
```

为初始(-Xms)和最大(-Xmx)堆大小设置相同的值可避免 JVM 启动后分配内存，其代价可能是分配超过真正需要的堆数。



重要

执行大量磁盘 I/O 的容器（如 Kafka 代理容器）需要可用内存才能用作操作系统页面缓存。在这样的容器中，请求的内存应当显著高于 JVM 使用的内存。

-XX 选项

-XX 选项用于配置 Apache Kafka 的 KAFKA_JVM_PERFORMANCE_OPTS 选项。

-XX 配置示例

```
jvmOptions:
  "-XX":
    "UseG1GC": true
    "MaxGCPauseMillis": 20
    "InitiatingHeapOccupancyPercent": 35
    "ExplicitGCInvokesConcurrent": true
```

由 -XX 配置生成的 JVM 选项

```
-XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -
XX:+ExplicitGCInvokesConcurrent -XX:-UseParNewGC
```

注意

如果没有指定 -XX 选项，则会使用 KAFKA_JVM_PERFORMANCE_OPTS 的默认 Apache Kafka 配置。

javaSystemProperties

javaSystemProperties 用于配置其他 Java 系统属性，如调试实用程序。

javaSystemProperties 配置示例

```
jvmOptions:
  javaSystemProperties:
    - name: javax.net.debug
      value: ssl
```

13.1.10. 垃圾收集器日志记录

jvmOptions 属性还允许您启用和禁用垃圾收集器(GC)日志记录。默认禁用 GC 日志。要启用它，请按如下方式设置 gcLoggingEnabled 属性：

GC 日志配置示例

```
# ...
jvmOptions:
  gcLoggingEnabled: true
# ...
```

13.2. 架构属性

13.2.1. Kafka 模式参考

属性	描述
spec	Kafka 和 ZooKeeper 集群和 Topic Operator 的规格。
kafkaSpec	
status	Kafka 和 ZooKeeper 集群和 Topic Operator 的状态。
KafkaStatus	

13.2.2. KafkaSpec 模式参考

用于： ***Kafka***

属性	描述
kafka	Kafka 集群的配置。
KafkaClusterSpec	
zookeeper	ZooKeeper 集群的配置.
ZookeeperClusterSpec	
entityOperator	配置实体 Operator.
EntityOperatorSpec	
clusterCa	集群证书颁发机构配置。
CertificateAuthority	
clientsCa	客户端证书颁发机构的配置。
CertificateAuthority	

属性	描述
cruiseControl **CruiseControlSpec**	清理控制部署的配置.在指定时部署 Cruise Control 实例。
kafkaExporter **KafkaExporterSpec**	配置 Kafka 导出器.Kafka Exporter 可以提供附加指标，例如主题/分区上消费者组的滞后。
maintenanceTimeWindows 字符串数组	用于维护任务的时间窗列表（即证书续订）。每次由 cron 表达式定义时窗口.

13.2.3. KafkaClusterSpec 模式参考

中使用的： KafkaSpec

KafkaClusterSpec 模式属性的完整列表

配置 Kafka 集群。

13.2.3.1. 监听程序

使用 listens 属性 配置监听程序，以提供对 Kafka 代理的访问。

无需身份验证的普通（未加密）侦听器配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
    # ...
  zookeeper:
    # ...
```

13.2.3.2. config

使用 config 属性将 Kafka 代理选项配置为密钥。

标准 Apache Kafka 配置可能会提供，仅限于不是由 AMQ Streams 直接管理的属性。

无法配置的选项与以下内容相关：

- 安全（加密、身份验证和授权）
- 监听程序配置
- 代理 ID 配置
- 日志数据目录的配置
- Broker 间通信
- zookeeper 连接

这些值可以是以下 JSON 类型之一：

- 字符串
- 数字
- 布尔值

您可以指定并配置 Apache Kafka 文档 中列出的选项，但那些直接由 AMQ Streams 管理的选项除外。具体来说，所有键为等于或以以下任一字符串开头的配置选项将被禁止：

- *监听程序*
- *advertised.*
- *代理.*
- *listener.*
- *host.name*
- *port*
- *inter.broker.listener.name*
- *sasl.*
- *ssl.*
- *安全性.*
- *password.*
- *principal.builder.class*
- *log.dir*

- *zookeeper.connect*
- *zookeeper.set.acl*
- *authorizer.*
- *super.user*

当 config 属性中存在禁止选项时，会忽略它，并把警告信息输出到 Cluster Operator 日志文件中。所有其他支持的选项都传递给 Kafka。

禁止的选项有例外。对于使用特定 密码套件 作为 TLS 版本进行客户端连接，您可以配置 allowed ssl 属性。您还可以配置 zookeeper.connection.timeout.ms 属性，以设置建立 ZooKeeper 连接的最长时间。

Kafka 代理配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    config:
      num.partitions: 1
      num.recovery.threads.per.data.dir: 1
      default.replication.factor: 3
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 1
      log.retention.hours: 168
      log.segment.bytes: 1073741824
      log.retention.check.interval.ms: 300000
      num.network.threads: 3
      num.io.threads: 8
      socket.send.buffer.bytes: 102400
      socket.receive.buffer.bytes: 102400
      socket.request.max.bytes: 104857600
      group.initial.rebalance.delay.ms: 0
      ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384"
      ssl.enabled.protocols: "TLSv1.2"
```

```
    ssl.protocol: "TLSv1.2"
    zookeeper.connection.timeout.ms: 6000
  # ...
```

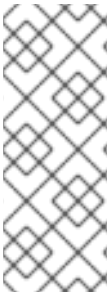
13.2.3.3. brokerRackInitImage

启用机架感知后，Kafka 代理容器集使用 init 容器从 OpenShift 集群节点收集标签。此容器使用的容器镜像可使用 brokerRackInitImage 属性进行配置。当缺少 brokerRackInitImage 字段时，会按照优先级顺序使用以下镜像：

1. 在 Cluster Operator 配置中的 STRIMZI_DEFAULT_KAFKA_INIT_IMAGE 环境变量中指定的容器镜像。
2. registry.redhat.io/amq7/amq-streams-rhel7-operator:1.8.0 container image.

brokerRackInitImage 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    rack:
      topologyKey: topology.kubernetes.io/zone
    brokerRackInitImage: my-org/my-image:latest
    # ...
```



注意

建议仅在特殊情况下覆盖容器镜像，这时您需要使用不同的容器注册表。例如，因为您的网络不允许访问 AMQ Streams 使用的容器 registry。在这种情况下，您应该复制 AMQ Streams 镜像或从源构建它们。如果配置的镜像与 AMQ Streams 镜像不兼容，它可能无法正常工作。

13.2.3.4. logging

Kafka 拥有自己的可配置日志记录器：

- *log4j.logger.org.I0Itec.zkclient.ZkClient*
- *log4j.logger.org.apache.zookeeper*
- *log4j.logger.kafka*
- *log4j.logger.org.apache.kafka*
- *log4j.logger.kafka.request.logger*
- *log4j.logger.kafka.network.Processor*
- *log4j.logger.kafka.server.KafkaApis*
- *log4j.logger.kafka.network.RequestChannel$*
- *log4j.logger.kafka.controller*
- *log4j.logger.kafka.log.LogCleaner*
- *log4j.logger.state.change.logger*
- *log4j.logger.kafka.authorizer.logger*

Kafka 使用 Apache log4j 日志记录器实施。

使用 logging 属性来配置日志记录器和日志记录器级别。

您可以通过直接（内线）指定日志记录器和级别来设置日志级别，或使用自定义（外部）ConfigMap。如果使用 ConfigMap，则将 logging.valueFrom.configMapKeyRef.name 属性设置为包含外部日志记录配置的 ConfigMap 的名称。在 ConfigMap 中，日志配置使用 log4j.properties 进行 描述。logging.valueFrom.configMapKeyRef.name 和 logging.valueFrom.configMapKeyRef.key 属性均是必需的。在运行 Cluster Operator 时，会使用自定义资源创建使用指定准确日志配置的 ConfigMap，然后在每次协调后重新创建。如果没有指定自定义 ConfigMap，则会使用默认日志设置。如果没有设置特定的日志记录器值，则会继承该日志记录器的上一级日志记录器设置。有关日志级别的更多信息，请参阅 Apache 日志记录服务。

此处我们会看到 内联 和外部 记录示例。

内联日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # ...
  kafka:
    # ...
    logging:
      type: inline
      loggers:
        kafka.root.logger.level: "INFO"
  # ...
```

外部日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # ...
  logging:
    type: external
    valueFrom:
      configMapKeyRef:
        name: customConfigMap
        key: kafka-log4j.properties
  # ...
```

任何未配置可用的日志记录器将其级别设置为 OFF。

如果使用 Cluster Operator 部署 Kafka，则会动态应用对 Kafka 日志级别的更改。

如果使用外部日志记录，当日志附加程序被更改时会触发滚动更新。

垃圾收集器(GC)

也可以使用 jvmOptions 属性 来启用（或禁用）垃圾收集器日志记录。

13.2.3.5. KafkaClusterSpec 模式属性

属性	描述
version 字符串	kafka 代理版本。默认值为 2.8.0。请参阅用户文档以了解升级或降级版本所需的流程。
replicas 整数	集群中的 pod 数量。
image 字符串	容器集的 docker 镜像。默认值为配置的 **Kafka.spec.kafka.version**。
监听程序 GenericKafkaListener 数组	配置 Kafka 代理的监听程序。

属性	描述
config	无法设置带有以下前缀的 Kafka 代理配置属性：监听器、公告的.、代理.、监听器.、host.name、端口、inter.broker.listener.name、sasl.、ssl.、security.、security.builder.class、log.dir、zookeeper.connect、zookeeper.set.acl、 zookeeper.ssl, zookeeper.clientCnxnSocket, authorizer., super.user, cruise.control.metrics.topic, cruise.control.metrics.reporter.bootstrap.servers（以下除外： zookeeper.connection.timeout.ms, ssl.cipher.suites, SSL.protocol, ssl.enabled.protocols,cruise.control.metrics.topic.num.partitions, cruise.control.metrics.topic.replication.factor, cruise.control.metrics.topic.retention.ms,cruise.control.metrics.topic.auto.create.reries, cruise.control.metrics.topic.auto.create.timeout.ms,cruise.control.metrics.topic.min.insync.replicas）。
map	
storage	存储配置（磁盘）。无法更新。类型取决于给定对象中 **storage.type** 属性的值，它必须是 [ephemeral, persistent-claim, jbod] 之一。
EphemeralStorage, **PersistentClaimStorage**, **JbodStorage**	
授权	Kafka 代理的授权配置。类型取决于给定对象中的 **authorization.type** 属性的值，它必须是 [simple、opa、keycloak、custom] 之一。
KafkaAuthorizationSimple, **KafkaAuthorizationOpa**, **KafkaAuthorizationKeycloak**, **KafkaAuthorizationCustom**	
rack	配置 **broker.rack** 代理配置。
rack	
brokerRackInitImage	用于初始化 **broker.rack** 的 init 容器镜像。
字符串	
livenessProbe	Pod 存活度检查.
probe	
readinessprobe	Pod 就绪度检查。
probe	
jvmOptions	容器集的 JVM 选项.

属性	描述
JvmOptions	
jmxOptions	Kafka 代理的 JMX 选项.
KafkaJmxOptions	
资源	要保留的 CPU 和内存资源。如需更多信息，请参阅核心/v1 资源要求的外部文档。
ResourceRequirements	
metricsConfig	指标配置.这个类型取决于给定对象中 **metricsConfig.type** 属性的值，必须是 [jmxPrometheusExporter] 中的一个。
JmxPrometheusExporterMetrics	
logging	Kafka 的日志配置。类型取决于给定对象中的 **logging.type** 属性的值，它必须是 [inline, external] 之一。
InlineLogging, **ExternalLogging**	
模板	Kafka 集群资源的模板。该模板允许用户指定 **StatefulSet**、**Pod** 和服务 生成的方式。
KafkaClusterTemplate	

13.2.4. GenericKafkaListener 模式参考

用于： ***KafkaClusterSpec***

GenericKafkaListener 模式属性的完整列表

配置监听程序以连接到 OpenShift 内部和外部的 Kafka 代理。

您可以在 Kafka 资源中配置监听程序。

显示监听器配置的 Kafka 资源示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
```

```
  name: my-cluster
spec:
  kafka:
    #...
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
      - name: external1
        port: 9094
        type: route
        tls: true
      - name: external2
        port: 9095
        type: ingress
        tls: true
        authentication:
          type: tls
        configuration:
          bootstrap:
            host: bootstrap.myingress.com
          brokers:
          - broker: 0
            host: broker-0.myingress.com
          - broker: 1
            host: broker-1.myingress.com
          - broker: 2
            host: broker-2.myingress.com
    #...
```

13.2.4.1. 监听程序

您可以使用 Kafka 资源中的 listens 属性配置 Kafka 代理 监听程序。侦听器定义为数组。

监听程序配置示例

```
listeners:
  - name: plain
    port: 9092
```

```
    type: internal
    tls: false
```

Kafka 集群中的名称和端口必须是唯一的。名称最多可包含 25 个字符，包含小写字母和数字。允许的端口号是 9092 及以上，但端口 9404 和 9999 除外，它们已用于 Prometheus 和 JMX。

通过为每个监听器指定唯一的名称和端口，您可以配置多个监听器。

13.2.4.2. type

该类型设置为 internal，或针对外部监听器，作为 路由、loadbalancer、nodeport 或 ingress。

internal

您可以使用 tls 属性使用或不加密配置内部监听程序。

内部 监听程序配置示例

```
#...
spec:
  kafka:
    #...
    listeners:
      #...
      - name: plain
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
    #...
```

Route

配置外部侦听器以使用 OpenShift Route 和 HAProxy 路由器公开 Kafka。

为每个 Kafka 代理 pod 创建一个专用 路由。创建一个额外的 Route 来充当 Kafka bootstrap 地址。Kafka 客户端可以使用这些 路由 连接端口 443 上的 Kafka。客户端在端口 443（默认路由器端口）上连接，但流量将路由到您配置的端口，本例中为 9094。

路由 监听程序配置示例

```
#...
spec:
  kafka:
    #...
    listeners:
      #...
      - name: external1
        port: 9094
        type: route
        tls: true
    #...
```

Ingress

配置一个外部监听程序以使用 Kubernetes Ingress 和 Kubernetes 的 NGINX Ingress Controller 公开 Kafka。

为每个 Kafka 代理 Pod 创建一个专用的 Ingress 资源。创建一个额外的 Ingress 资源来充当 Kafka bootstrap 地址。Kafka 客户端可以使用这些 Ingress 资源连接到端口 443 上的 Kafka。客户端在端口 443（默认控制器端口）上连接，但流量会路由到您配置的端口，以下示例中是 9095。

您必须使用 GenericKafkaListenerConfigurationBootstrap 和 GenericKafkaListenerConfiguration Broker 属性指定 bootstrap 和每个broker 服务使用的主机名。

ingress 侦听器配置示例

```
#...
spec:
  kafka:
```

```
#...
listeners:
  #...
  - name: external2
    port: 9095
    type: ingress
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        host: bootstrap.myingress.com
      brokers:
      - broker: 0
        host: broker-0.myingress.com
      - broker: 1
        host: broker-1.myingress.com
      - broker: 2
        host: broker-2.myingress.com
#...
```

注意

使用 Ingress 的外部监听程序目前只使用 Kubernetes 的 NGINX Ingress Controller 测试。

LoadBalancer

配置一个外部监听程序来公开 Kafka Loadbalancer 类型 服务。

为每个 Kafka 代理 pod 创建一个新的负载均衡器服务。创建一个额外的负载均衡器来充当 Kafka bootstrap 地址。Loadbalancers 侦听指定的端口号，以下示例中是端口 9094。

您可以使用 loadBalancerSourceRanges 属性配置 源范围，以限制对指定 IP 地址的访问。

loadbalancer 侦听器配置示例

```
#...
spec:
  kafka:
    #...
    listeners:
```

```
    - name: external3
      port: 9094
      type: loadbalancer
      tls: true
      configuration:
        loadBalancerSourceRanges:
          - 10.0.0.0/8
          - 88.208.76.87/32
    #...
```

nodeport

配置外部侦听器以使用 NodePort 类型 服务 公开 Kafka。

Kafka 客户端直接连接到 OpenShift 节点。创建额外的 NodePort 类型服务作为 Kafka bootstrap 地址。

在为 Kafka 代理 Pod 配置公告的地址时，AMQ Streams 使用运行给定 pod 的节点的地址。您可以使用 preferredNodePortAddressType 属性配置 第一个地址类型作为节点地址检查。

nodeport 侦听器配置示例

```
#...
spec:
  kafka:
    #...
    listeners:
      #...
      - name: external4
        port: 9095
        type: nodeport
        tls: false
        configuration:
          preferredNodePortAddressType: InternalDNS
    #...
```

注意

在使用节点端口公开 Kafka 集群时，当前不支持 TLS 主机名验证。

13.2.4.3. port

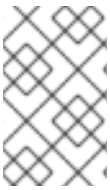
端口号是 Kafka 集群中使用的端口，它可能不是客户端用于访问的端口。

- LoadBalancer 侦听器 使用指定的端口号，如 内部 监听器一样
- Ingress 和 路由 监听器使用端口 443 访问
- NodePort 侦听器使用 OpenShift 分配的端口号

对于客户端连接，请使用监听器的 bootstrap 服务的地址和端口。您可以从 Kafka 资源的状态检索它。

检索客户端连接的地址和端口的命令示例

```
oc get kafka KAFKA-CLUSTER-NAME -o=jsonpath='{.status.listeners[?
(@.type=="external")].bootstrapServers}{"\n"}'
```



注意

无法将监听器配置为使用为 Interbroker 通信(9091)和指标(9404)设置的端口。

13.2.4.4. tls

TLS 属性是必需的。

默认情况下不启用 TLS 加密。要启用它，请将 tls 属性设置为 true。

TLS 加密始终与 路由 监听程序一起使用。

13.2.4.5. 身份验证

监听器的身份验证可指定为：

- 双向 TLS(tls)
- SCRAM-SHA-512 (scram-sha-512)
- 基于令牌的 OAuth 2.0(oauth)。

13.2.4.6. networkPolicyPeers

使用 networkPolicyPeers 配置网络策略，以限制对网络级别的监听器的访问。以下示例显示了用于普通 和 a tls 侦听器的 networkPolicyPeers 配置。

```
listeners:
  #...
  - name: plain
    port: 9092
    type: internal
    tls: true
    authentication:
      type: scram-sha-512
    networkPolicyPeers:
      - podSelector:
          matchLabels:
            app: kafka-sasl-consumer
      - podSelector:
          matchLabels:
            app: kafka-sasl-producer
  - name: tls
    port: 9093
    type: internal
    tls: true
    authentication:
      type: tls
    networkPolicyPeers:
      - namespaceSelector:
```

```
        matchLabels:
          project: myproject
      - namespaceSelector:
          matchLabels:
            project: myproject2
# ...
```

在这个示例中：

- 只有与标签 app: kafka-sasl-consumer 和 app: kafka-sasl-producer 匹配的应用程序 pod 可以连接到 普通的 监听程序。应用程序 pod 必须与 Kafka 代理在同一命名空间中运行。
- 只有命名空间中运行的应用容器集与 labels 项目匹配：myproject 和 project ： myproject2 可以连接到 tls 侦听器。

networkPolicyPeers 字段的语法与 NetworkPolicy 资源中的 from 字段相同。

13.2.4.7. GenericKafkaListener 架构属性

属性	描述
name 字符串	侦听器的名称。名称将用于识别侦听器和相关 OpenShift 对象。该名称必须在给定的 Kafka 集群内唯一。名称可以包含小写字符和数字，并且最多可包含 11 个字符。
port 整数	Kafka 内侦听器使用的端口号。在给定 Kafka 集群中，端口号必须是唯一的。允许的端口号是 9092 及以上，但端口 9404 和 9999 除外，它们已用于 Prometheus 和 JMX。根据监听程序类型，端口号可能与连接 Kafka 客户端的端口号不同。

属性	描述
type 字符串（[ingress、internal、route、loadbalancer、nodeport] 之一）	侦听器的类型。目前支持的类型有 **internal**、**route**、**loadbalancer**、**nodeport** 和 **ingress**。 • **内部** 类型仅在 OpenShift 集群内部公开 Kafka。 • **路由** 类型使用 OpenShift Route 来公开 Kafka。 • **LoadBalancer 类型** 使用 LoadBalancer 类型服务来公开 Kafka。 • **NodePort 类型** 使用 NodePort 类型服务来公开 Kafka。 • **Ingress** 类型使用 OpenShift Nginx Ingress 来公开 Kafka。
tls 布尔值	在监听器上启用 TLS 加密。这是必填属性。
身份验证 **KafkaListenerAuthenticationTls**, **KafkaListenerAuthenticationScramSha512**, **KafkaListenerAuthenticationOAuth**	此侦听器的身份验证配置.这个类型取决于给定对象中的 **authentication.type** 属性的值，必须是 [tls、scram-sha-512、oauth] 中的一个。
配置 **GenericKafkaListenerConfiguration**	其他监听器配置.
networkPolicyPeers NetworkPolicyPeer 数组	能够连接到此监听器的同级服务器列表。此列表中的对等点使用逻辑 OR 操作组合。如果此字段为空或缺失，则允许此监听器所有连接。如果此字段存在并且至少包含一项，侦听器仅允许与此列表中至少匹配一项的流量。如需更多信息，请参阅 networking.k8s.io/v1 networkpolicypeer 的外部文档。

13.2.5. KafkaListenerAuthenticationTls 模式参考

用于： ***GenericKafkaListener***

type 属性是一个光盘，它区分使用 KafkaListenerAuthenticationTls 类型与 KafkaListenerAuthenticationScramSha512、KafkaListenerAuthenticationOAuth。它必须具有类型 KafkaListenerAuthenticationTls 的值 tls。

属性	描述
type	必须是 **tls**。
字符串	

13.2.6. KafkaListenerAuthenticationScramSha512 schema reference

用于： *GenericKafkaListener*

***type** 属性是一个光盘，它区分使用 KafkaListenerAuthenticationScramSha512 类型与 KafkaListenerAuthenticationTls、KafkaListenerAuthenticationOAuth。它必须具有类型 KafkaListenerAuthenticationScramSha512 的值 scr am-sha-512。*

属性	描述
type	必须是 scr **am-sha-512**。
字符串	

13.2.7. KafkaListenerAuthenticationOAuth 模式参考

用于： *GenericKafkaListener*

***type** 属性是一个光盘，它区分使用 KafkaListenerAuthenticationOAuth 类型与 KafkaListenerAuthenticationTls、KafkaListenerAuthenticationScramSha512。它必须具有类型 KafkaListenerAuthenticationOAuth 的值 oauth。*

属性	描述
accessTokenIsJwt	配置访问令牌是否被视为 JWT。如果授权服务器返回不透明令牌，则必须将此设置为 **false**。默认值为 **true**。
布尔值	
checkAccessTokenType	配置是否执行访问令牌类型检查。如果授权服务器在 JWT 令牌中不包含 'typ' 声明，这应当设为 **false**。默认值为 **true**。
布尔值	

属性	描述
checkAudience 布尔值	启用或禁用受众检查。使用者检查可识别令牌的接收者。如果启用了使用者检查，还必须使用 **clientId** 属性配置 OAuth 客户端 ID。Kafka 代理将拒绝在 **aud**(audience)claim.Default 值为 **false** 中没有 **clientId** 的令牌。
checkIssuer 布尔值	启用或禁用签发者检查。默认情况下，使用 **validIssuerUri** 配置的值检查签发者。默认值为 **true**。
clientAudience 字符串	向授权服务器的令牌端点发出请求时使用的受众。用于代理验证，并使用 **clientId** 和 **secret** 方法通过 PLAIN 配置 OAuth 2.0。
clientId 字符串	Kafka 代理可用于向授权服务器进行身份验证并使用内省端点 URI 的 OAuth 客户端 ID。
clientScope 字符串	向授权服务器的令牌端点发出请求时使用的范围。用于代理验证，并使用 **clientId** 和 **secret** 方法通过 PLAIN 配置 OAuth 2.0。
clientSecret **GenericSecretSource**	到包含 OAuth 客户端机密的 OpenShift Secret 的链接，Kafka 代理可使用该 secret 与授权服务器进行身份验证并使用内省端点 URI。
customClaimCheck 字符串	jsonpath 过滤器查询应用到 JWT 令牌或内省端点的响应，以进行额外的令牌验证。默认情况下不设置。
disableTlsHostnameVerification 布尔值	启用或禁用 TLS 主机名验证。默认值为 **false**。
enableECDSA 布尔值	**enableECDSA 属性已弃用。**通过安装 BouncyCastle 加密提供程序来启用或禁用 ECDSA 支持。始终启用 ECDSA 支持。BouncyCastle 库不再与 AMQ Streams 打包。值将被忽略。
enableOauthBearer 布尔值	通过 SASL_OAUTHBEARER 启用或禁用 OAuth 身份验证。默认值为 **true**。
enablePlain	通过 SASL_PLAIN 启用或禁用 OAuth 身份验证。在使用此机制时，不支持再验证。默认值为 **false**。

属性	描述
布尔值	
fallbackUserNameClaim 字符串	如果 **userNameClaim** 指定的声明不存在，则用于用户 ID 的回退用户名声明。当 **client_credentials** 身份验证只在另一声明中提供客户端 ID 时，这很有用。只有在设置了 **userNameClaim** 时，它才会生效。
fallbackUserNamePrefix 字符串	与 **fallbackUserNameClaim** 值一起使用的前缀来构造用户 ID。这只有在 **fallbackUserNameClaim** 为 true 时生效，并且该值存在该声明。将用户名和客户端 ID 映射到相同的用户 id 空间有助于防止名称冲突。
introspectionEndpointUri 字符串	令牌内省端点的 URI，可用于验证不透明非 JWT 令牌。
jwksEndpointUri 字符串	JWKS 证书端点的 URI，可用于本地 JWT 验证。
jwksExpirySeconds 整数	配置 JWKS 证书的有效期。到期间隔必须至少为 60 秒，然后 **jwksRefreshSeconds** 中指定的刷新间隔必须至少为 60 秒。默认值为 360 秒。
jwksMinRefreshPauseSeconds 整数	连续两次刷新之间的最小暂停。当遇到未知签名密钥时，会立即调度刷新，但会始终等待这一最小暂停。默认值为 1 秒。
jwksRefreshSeconds 整数	配置刷新的 JWKS 证书的频率。刷新间隔必须至少缩短 60 秒，然后 **jwksExpirySeconds** 中指定的到期间隔必须最少。默认值为 300 秒。
maxSecondsWithoutReauthentication 整数	经过身份验证的会话保持有效的最大秒数，无需重新身份验证。这可启用 Apache Kafka 重新身份验证功能，并使会话在访问令牌过期时过期。如果访问令牌在最长时间前过期，或者达到最大时间，客户端必须重新进行身份验证，否则服务器将丢弃连接。默认情况下不设置 - 经过身份验证的会话不会在访问令牌过期时过期。这个选项只适用于 SASL_OAUTHBEARER 身份验证机制（当 **enableOauthBearer** 为 **true**时）。
tlsTrustedCertificates **CertSecretSource** 数组	TLS 连接到 OAuth 服务器的可信证书。

属性	描述
tokenEndpointUri 字符串	客户端通过 **clientId** 和 **secret** 验证时要用于 SASL_PLAIN 机制的 Token 端点 URI。如果设置，客户端可以通过 SASL_PLAIN 将 **用户名** 设置为 **clientId**，将 **密码** 设置为客户端 **secret**，或者将 **用户名** 设置为帐户用户名，**密码** 用于访问前缀为 **$accessToken:** 的令牌：如果未设置此选项，则 **密码** 始终解释为访问令牌（不含前缀），用户名为帐户 **用户名**（称为"no-client-credentials"模式）。
type 字符串	必须是 **oauth**。
userInfoEndpointUri 字符串	当内省端点不返回可用于用户 ID 的信息时，用作回退的 User Info Endpoint 的 URI 用于获取用户 ID。
userNameClaim 字符串	来自 JWT 身份验证令牌的声明名称、内省端点响应或用户 Info Endpoint 响应，它们将用于提取用户 ID。默认值为 **sub**。
validIssuerUri 字符串	用于身份验证的令牌签发者的 URI。
validTokenType 字符串	Introspection Endpoint 返回的 **token_type** 属性的有效值。没有默认值，默认情况下不检查。

13.2.8. GenericSecretSource 模式参考

用于：KafkaClientAuthenticationOAuth、KafkaListenerAuthenticationOAuth

属性	描述
key 字符串	机密值存储在 OpenShift Secret 中的键。
secretName 字符串	包含机密值的 OpenShift Secret 的名称。

13.2.9. CertSecretSource 模式参考

用于： KafkaAuthorizationKeycloak, KafkaBridgeTls, KafkaClientAuthenticationOAuth, KafkaConnectTls, KafkaListenerAuthenticationOAuth, KafkaMirrorMaker2Tls, KafkaMirrorMakerTls

属性	描述
certificate	Secret 中的文件证书的名称。
字符串	
secretName	包含证书的 Secret 名称。
字符串	

13.2.10. GenericKafkaListenerConfiguration 模式参考

用于： GenericKafkaListener

GenericKafkaListenerConfiguration 模式属性的完整列表

Kafka 侦听器的配置.

13.2.10.1. brokerCertChainAndKey

brokerCertChainAndKey 属性仅用于启用了 TLS 加密的监听程序。您可以使用该属性来提供您自己的 Kafka 侦听程序证书。

启用 TLS 加密的负载均衡器 外部监听程序配置示例

```
listeners:
  #...
  - name: external
    port: 9094
    type: loadbalancer
    tls: true
    authentication:
      type: tls
    configuration:
```

```
      brokerCertChainAndKey:
        secretName: my-secret
        certificate: my-listener-certificate.crt
        key: my-listener-key.key
# ...
```

13.2.10.2. externalTrafficPolicy

externalTrafficPolicy 属性用于 loadbalancer 和 nodeport 侦听器。在 OpenShift 外部公开 Kafka 时，您可以选择 Local 或 Cluster。local 避免跃点到其他节点并保留客户端 IP，而 群集 则不会。默认值为 Cluster。

13.2.10.3. loadBalancerSourceRanges

loadBalancerSourceRanges 属性仅用于 负载均衡器侦听器。当在 OpenShift 外部公开 Kafka 时，除了标签和注释外，还会使用源范围来自定义服务的创建方式。

为负载均衡器监听程序配置源范围示例

```
listeners:
  #...
  - name: external
    port: 9094
    type: loadbalancer
    tls: false
    configuration:
      externalTrafficPolicy: Local
      loadBalancerSourceRanges:
        - 10.0.0.0/8
        - 88.208.76.87/32
      # ...
# ...
```

13.2.10.4. class

class 属性仅用于 入口 侦听器。您可以使用类属性配置 Ingress 类。

使用 Ingress 类 nginx-internal 类型的 ingress 外部侦听器示例

```
listeners:
  #...
  - name: external
    port: 9094
    type: ingress
    tls: true
    configuration:
      class: nginx-internal
    # ...
# ...
```

13.2.10.5. preferredNodePortAddressType

preferredNodePortAddressType 属性仅用于 nodeport 监听程序。

使用监听程序配置中的 preferredNodePortAddressType 属性来指定检查为节点地址的第一个地址类型。例如，如果您的部署没有 DNS 支持，或者您只想通过内部 DNS 或 IP 地址公开代理，则此属性很有用。如果找到此类型的地址，则会使用它。如果没有找到首选地址类型，AMQ Streams 会按照标准优先级顺序逐步完成类型：

1. **ExternalDNS**
2. **ExternalIP**
3. **主机名**
4. **InternalDNS**
5. **InternalIP**

使用首选节点端口地址类型配置的外部监听程序示例

```
listeners:
  #...
  - name: external
    port: 9094
    type: nodeport
    tls: false
    configuration:
      preferredNodePortAddressType: InternalDNS
      # ...
# ...
```

13.2.10.6. useServiceDnsDomain

useServiceDnsDomain 属性仅用于 内部 监听器。它定义是否使用包含集群服务后缀（通常. cluster.local）的完全限定 DNS 名称。使用ServiceDnsDomain 设置为 false 时，公告的地址在没有服务后缀的情况下生成；例如， my -cluster-kafka-0.my-cluster-kafka-brokers.myproject.svc。使用 ServiceDnsDomain 设置为 true 时，公告的地址通过服务后缀生成；例如， my -cluster-kafka-0.my-cluster-kafka-brokers.myproject.svc.cluster.local。默认为 false。

内部监听程序示例配置为使用 Service DNS 域

```
listeners:
  #...
  - name: plain
    port: 9092
    type: internal
    tls: false
    configuration:
      useServiceDnsDomain: true
      # ...
# ...
```

如果您的 OpenShift 集群使用与. cluster.local 不同的服务后缀，您可以在 Cluster Operator 配置中使用 KUBERNETES_SERVICE_DNS_DOMAIN 环境变量来配置后缀。详情请查看 第 5.1.1 节 “Cluster Operator 配置”。

13.2.10.7. GenericKafkaListenerConfiguration 模式属性

属性	描述
brokerCertChainAndKey **CertAndKeySecretSource**	引用存放用于此侦听器的证书和私钥的 **Secret**。证书可以选择包含整个链。此字段只能用于启用 TLS 加密的监听程序。
externalTrafficPolicy 字符串（[本地、集群] 中的一个）	指定服务将外部流量路由到节点本地或集群范围的端点。**群集** 可能会导致第二个跃点到另一节点，并屏蔽客户端源 IP。**local** 避免了 LoadBalancer 和 Nodeport 类型服务的第二个跃点，并且保留客户端源 IP（当基础架构支持时）。如果未指定，OpenShift 将默认使用 **Cluster**。This 字段只能用于 **loadbalancer** 或 **nodeport** 类型侦听器。
loadBalancerSourceRanges 字符串数组	CIDR 范围列表（如 **10.0.0.0/8** 或 **130.211.204.1/32**），客户端可以从中连接到负载均衡器类型侦听器。如果平台支持，通过负载均衡器的流量将限制为指定的 CIDR 范围。此字段仅适用于 loadbalancer 类型服务，如果云供应商不支持该功能，则忽略该字段。如需更多信息，请参阅 https://v1-17.docs.kubernetes.io/docs/tasks/access-application-cluster/configure-cloud-provider-firewall/。此字段只能用于 **loadbalancer** 类型侦听器。
bootstrap **GenericKafkaListenerConfigurationBootstrap**	Bootstrap 配置.
代理 **GenericKafkaListenerConfigurationBroker** 数组	按代理配置.
ipFamilyPolicy string (one of [RequireDualStack, SingleStack, PreferDualStack])	指定服务使用的 IP 系列策略。可用选项包括 **SingleStack**、**PreferDualStack** 和 **RequireDualStack**。**SingleStack 适用于** 单个 IP 系列。**PreferDualStack** 是两个 IP 系列，位于双栈配置的群集上，或者单堆栈集群中只有一个 IP 系列。**RequireDualStack** 会失败，除非双栈配置的群集上有两个 IP 系列。如果未指定，OpenShift 将根据服务类型选择默认值。适用于 OpenShift 1.20 及更新版本.
ipFamilies 字符串（[IPv6、IPv4]）数组的一个或多个	指定服务使用的 IP 系列。可用的选项有 **IPv4** 和 **IPv6。如果未指定，OpenShift 将根据 'ipFamilyPolicy** 设置选择默认值。适用于 OpenShift 1.20 及更新版本.

属性	描述
class 字符串	配置用于定义将使用哪个 **Ingress** 控制器的 Ingress 类。这个字段只能用于 **ingress** 类型监听程序。如果没有指定，将使用默认的 Ingress 控制器。
结束程序 字符串数组	为这个监听程序创建的 **LoadBalancer 类型服务配置** 的终结器列表。如果平台支持，finalr **service.kubernetes.io/load-balancer-cleanup** 可确保与 service 一起删除外部负载均衡器。如需更多信息，请参阅 https://kubernetes.io/docs/tasks/access-application-cluster/create-external-load-balancer/#garbage-collecting-load-balancers。此字段只能用于 **loadbalancer** 类型侦听器。
maxConnectionCreationRate 整数	我们随时允许在此监听器中允许的最大连接创建率。如果达到限制，新连接将会节流。只在 Kafka 2.7.0 及更新版本中支持新连接。
maxConnections 整数	我们在代理中允许此监听器的最大连接数。如果达到限制，新连接将被阻止。
preferredNodePortAddressType 字符串（[外部DNS、ExternalDNS、Hostname、InternalIP、InternalDNS] 之一）	定义应当将哪种地址类型用作节点地址。可用类型有： **ExternalDNS**、ExternalIP、**InternalDNS**、**InternalIP** 和 **Hostname**。默认情况下，这些地址将按以下顺序使用（将使用找到的第一个地址）：* **ExternalDNS** * **ExternalIP** * **InternalDNS** * **InternalIP** * **Hostname** 此字段用于选择首选地址类型，先进行检查。如果没有为此地址类型找到地址，则按默认顺序检查其他类型。此字段只能与 **nodeport** 类型侦听器一起使用。
useServiceDnsDomain 布尔值	配置 OpenShift 服务 DNS 域是否应该使用。如果设置为 **true**，生成的地址将包含服务 DNS 域后缀（默认为 **.cluster.local**，可以使用环境变量 **KUBERNETES_SERVICE_DNS_DOMAIN**）进行配置。默认值为 **false**。This 字段只能用于 **内部** 类型监听程序。

13.2.11. CertAndKeySecretSource schema reference

用于： ***GenericKafkaListenerConfiguration***、***KafkaClientAuthenticationTls***

属性	描述
certificate	Secret 中的文件证书的名称。
字符串	
key	Secret 中的私钥名称。
字符串	
secretName	包含证书的 Secret 名称。
字符串	

13.2.12. GenericKafkaListenerConfigurationBootstrap schema reference

用于： *GenericKafkaListenerConfiguration*

GenericKafkaListenerConfigurationBootstrap 架构属性的完整列表

nodePort、host、loadBalancerIP 和 注解 属性等效的代理服务在 GenericKafkaListenerConfigurationBroker 模式中配置。

13.2.12.1. alternativeNames

您可以为 bootstrap 服务指定备用名称。名称添加到代理证书中，可用于 TLS 主机名验证。alternativeNames 属性适用于所有类型的监听程序。

使用额外 bootstrap 地址配置的外部 路由 监听程序示例

```
listeners:
  #...
  - name: external
    port: 9094
    type: route
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        alternativeNames:
```

```
        - example.hostname1
        - example.hostname2
# ...
```

13.2.12.2. host

host 属性与 路由 和 入口 侦听器一起使用，以指定 bootstrap 和每个代理服务使用的主机名。

入口 监听程序配置需要 host 属性值，因为 Ingress 控制器不会自动分配任何主机名。确保主机名解析到 Ingress 端点。AMQ Streams 不会对请求的主机可用并正确路由到 Ingress 端点的任何验证执行任何验证。

入口监听程序的主机配置示例

```
listeners:
  #...
  - name: external
    port: 9094
    type: ingress
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        host: bootstrap.myingress.com
      brokers:
      - broker: 0
        host: broker-0.myingress.com
      - broker: 1
        host: broker-1.myingress.com
      - broker: 2
        host: broker-2.myingress.com
# ...
```

默认情况下，OpenShift 自动分配路由 侦听器主机。但是，您可以通过指定主机来覆盖分配的路由主机。

AMQ Streams 不执行任何对请求的主机可用的验证。您必须确保它们是免费的，并且可以使用。

路由监听程序的主机配置示例

```
# ...
listeners:
  #...
  - name: external
    port: 9094
    type: route
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        host: bootstrap.myrouter.com
      brokers:
      - broker: 0
        host: broker-0.myrouter.com
      - broker: 1
        host: broker-1.myrouter.com
      - broker: 2
        host: broker-2.myrouter.com
# ...
```

13.2.12.3. nodePort

默认情况下，OpenShift 自动分配用于 bootstrap 和代理服务的端口号。您可以通过指定请求的端口号来为 节点端口 监听程序覆盖分配的节点端口。

AMQ Streams 不在请求的端口上执行任何验证。您必须确保它们的免费可用。

使用节点端口覆盖配置的外部监听器示例

```
# ...
listeners:
  #...
  - name: external
    port: 9094
    type: nodeport
```

```
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        nodePort: 32100
      brokers:
      - broker: 0
        nodePort: 32000
      - broker: 1
        nodePort: 32001
      - broker: 2
        nodePort: 32002
# ...
```

13.2.12.4. loadBalancerIP

使用 loadBalancerIP 属性请求特定的 IP 地址。当您需要将负载平衡器与特定 IP 地址搭配使用时，请使用此属性。如果云供应商不支持该功能，则会忽略 loadBalancerIP 字段。

具有特定负载均衡器 IP 地址请求的负载均衡器类型的外部侦听器示例

```
# ...
listeners:
  #...
  - name: external
    port: 9094
    type: loadbalancer
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        loadBalancerIP: 172.29.3.10
      brokers:
      - broker: 0
        loadBalancerIP: 172.29.3.1
      - broker: 1
        loadBalancerIP: 172.29.3.2
      - broker: 2
        loadBalancerIP: 172.29.3.3
# ...
```

13.2.12.5. annotations

使用 annotations 属性向 OpenShift 资源添加与监听器相关的注释。例如，您可以使用这些注解来检测 DNS 工具，如 外部 DNS，它们会自动为负载均衡器服务分配 DNS 名称。

使用 注解的 loadbalancer 类型的外部侦听器示例

```
# ...
listeners:
  #...
  - name: external
    port: 9094
    type: loadbalancer
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        annotations:
          external-dns.alpha.kubernetes.io/hostname: kafka-bootstrap.mydomain.com.
          external-dns.alpha.kubernetes.io/ttl: "60"
      brokers:
      - broker: 0
        annotations:
          external-dns.alpha.kubernetes.io/hostname: kafka-broker-0.mydomain.com.
          external-dns.alpha.kubernetes.io/ttl: "60"
      - broker: 1
        annotations:
          external-dns.alpha.kubernetes.io/hostname: kafka-broker-1.mydomain.com.
          external-dns.alpha.kubernetes.io/ttl: "60"
      - broker: 2
        annotations:
          external-dns.alpha.kubernetes.io/hostname: kafka-broker-2.mydomain.com.
          external-dns.alpha.kubernetes.io/ttl: "60"
# ...
```

13.2.12.6. GenericKafkaListenerConfigurationBootstrap schema properties

属性	描述
alternativeNames 字符串数组	bootstrap 服务的其他备用名称。备用名称将添加到 TLS 证书的主题备用名称列表中。

属性	描述
主机 字符串	bootstrap 主机。此字段将用于 Ingress 资源或 Route 资源，以指定所需的主机名。此字段只能与 **路由** （可选）或 **ingress** （必需）类型侦听器一起使用。
nodePort 整数	bootstrap 服务的节点端口。此字段只能用于 **nodeport** 类型监听程序。
loadBalancerIP 字符串	使用此字段中指定的 IP 地址请求 loadbalancer。此功能取决于底层云供应商是否支持在创建负载均衡器时指定 **loadBalancerIP**。如果云供应商不支持这个功能，则此字段将被忽略。This 字段只能用于 **loadbalancer** 类型监听程序。
annotations map	添加到 **Ingress**、**Route** 或 **Service** 资源的注解。您可以使用此字段配置 DNS 供应商，如外部 DNS。此字段只能用于 **loadbalancer**、node**port**、**route** 或 **ingress** 类型监听程序。
labels map	添加到 **Ingress**、**Route** 或 **Service** 资源中的 标签。此字段只能用于 **loadbalancer**、node**port**、**route** 或 **ingress** 类型监听程序。

13.2.13. GenericKafkaListenerConfigurationBroker 模式参考

用于： *GenericKafkaListenerConfiguration*

GenericKafkaListenerConfigurationBroker 架构属性的完整列表

您可以在 GenericKafkaListenerConfigurationBootstrap 模式中 看到 nodePort、host、loadBalancerIP 和 annotations 属性的示例配置，后者配置 bootstrap 服务覆盖。

代理公告的地址

默认情况下，AMQ Streams 会尝试自动确定 Kafka 集群公告给客户端的主机名和端口。这并不适用于所有情况，因为运行 AMQ Streams 的基础架构可能无法提供可以通过其访问 Kafka 的正确主机名或端口。

您可以指定一个代理 ID，并在监听器 的配置 属性中自定义公告的主机名和端口。然后，AMQ

Streams 将自动在 Kafka 代理中配置公告的地址，并将其添加到代理证书中，以便用于 TLS 主机名验证。覆盖公告的主机和端口可供所有类型的监听器使用。

配置了针对公告地址覆盖的外部 路由 监听程序示例

```
listeners:
  #...
  - name: external
    port: 9094
    type: route
    tls: true
    authentication:
      type: tls
    configuration:
      brokers:
      - broker: 0
        advertisedHost: example.hostname.0
        advertisedPort: 12340
      - broker: 1
        advertisedHost: example.hostname.1
        advertisedPort: 12341
      - broker: 2
        advertisedHost: example.hostname.2
        advertisedPort: 12342
# ...
```

13.2.13.1. GenericKafkaListenerConfigurationBroker 模式属性

属性	描述
broker 整数	kafka 代理的 ID（代理标识符）。代理 ID 从 0 开始，对应于代理副本数。
advertisedHost 字符串	代理的 advertised **.brokers** 中将使用的主机名。
advertisedPort 整数	代理的 advertised **.brokers 中将使用** 的端口号。

属性	描述
主机 字符串	代理主机。此字段将用于 Ingress 资源或 Route 资源，以指定所需的主机名。此字段只能与 **路由** （可选）或 **ingress** （必需）类型侦听器一起使用。
nodePort 整数	每个broker 服务的节点端口。此字段只能用于 **nodeport** 类型监听程序。
loadBalancerIP 字符串	使用此字段中指定的 IP 地址请求 loadbalancer。此功能取决于底层云供应商是否支持在创建负载均衡器时指定 **loadBalancerIP**。如果云供应商不支持这个功能，则此字段将被忽略。This 字段只能用于 **loadbalancer** 类型监听程序。
annotations map	添加到 **Ingress** 或 **Service** 资源的注解。您可以使用此字段配置 DNS 供应商，如外部 DNS。此字段只能用于 **loadbalancer**、**node port** 或 **ingress** 类型监听程序。
labels map	添加到 **Ingress**、**Route** 或 **Service 资源中的** 标签。此字段只能用于 **loadbalancer**、node**port**、**route** 或 **ingress** 类型监听程序。

13.2.14. EphemeralStorage 架构参考

用于：JbodStorage、Kafka ClusterSpec、ZookeeperClusterSpec

type 属性是一个缺点，它区分了使用 EphemeralStorage 和 PersistentClaimStorage。它必须具有 type EphemeralStorage 的值。

属性	描述
id 整数	存储标识号.仅对类型为 'jbod" 的存储中定义的存储卷才强制使用。
sizeLimit 字符串	当 type=ephemeral 时，定义此 EmptyDir 卷所需的本地存储总量（例如 1Gi）。
type	必须 **是临时的**。

属性	描述
字符串	

13.2.15. PersistentClaimStorage 架构参考

用于： *JbodStorage*、*Kafka* *ClusterSpec*、*ZookeeperClusterSpec*

***type** 属性是一个缺点，它区分 **PersistentClaimStorage** 类型与 **EphemeralStorage** 的使用。它必须具有类型 **PersistentClaimStorage** 的值 **persistent-claim**。*

属性	描述
type	必须是 **永久声明**。
字符串	
Size	当 type=persistent-claim 时，定义持久性卷声明的大小（如 1Gi）。type=persistent-claim 时是必需的.
字符串	
selector	指定要使用的特定持久性卷。它包含键：值对，代表用于选择此类卷的标签。
map	
deleteClaim	指定在取消部署集群时是否需要删除持久性卷声明。
布尔值	
class	用于动态卷分配的存储类。
字符串	
id	存储标识号.仅对类型为 'jbod" 的存储中定义的存储卷才强制使用。
整数	
overrides	覆盖单个代理.**overrides** 字段允许为不同的代理指定不同的配置。
PersistentClaimStorageOverride 数组	

13.2.16. PersistentClaimStorageOverride schema reference

使用于： PersistentClaimStorage

属性	描述
class	用于这个代理的动态卷分配的存储类。
字符串	
broker	kafka 代理的 ID（代理标识符）。
整数	

13.2.17. JbodStorage schema reference

用于： KafkaClusterSpec

type 属性是一个缺点，它区分了 JbodStorage 类型与 EphemeralStorage、PersistentClaimStorage 的使用。它必须具有类型 Jbo dStorage 的值 jbo d。

属性	描述
type	必须为 **jbod**。
字符串	
卷	作为代表 JBOD 磁盘阵列的存储对象的卷列表。
EphemeralStorage, PersistentClaimStorage 数组	

13.2.18. KafkaAuthorizationSimple schema 参考

用于： KafkaClusterSpec

KafkaAuthorizationSimple schema 属性的完整列表

AMQ Streams 中的简单授权使用 AclAuthorizer 插件，即 Apache Kafka 提供的默认访问控制列表 (ACL)授权插件。ACL 允许您定义哪些用户有权在细粒度级别上访问哪些资源。

将 Kafka 自定义资源配置为使用简单授权。将 授权 部分中的 type 属性设置为 simple 值，再配置超级用户列表。

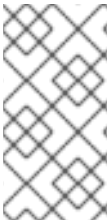
为 KafkaUser 配置访问规则，如 ACLRule schema 参考 中所述。

13.2.18.1. superUsers

用户主体列表被视为超级用户，因此始终允许他们无需查询 ACL 规则。如需更多信息，请参阅 Kafka 授权。

简单授权配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    authorization:
      type: simple
      superUsers:
        - CN=client_1
        - user_2
        - CN=client_3
    # ...
```



注意

Kafka .spec.kafka 中的 config 属性中的 super. user 配置选项将被忽略。在 授权 属性中指定超级用户。如需更多信息，请参阅 Kafka 代理配置。

13.2.18.2. KafkaAuthorizationSimple schema 属性

type 属性是一个光盘，它区分使用 KafkaAuthorizationSimple 类型与 KafkaAuthorizationOpa、KafkaAuthorizationKeycloak、KafkaAuthorizationCustom。它的值 对于 KafkaAuthorizationSimple 类型必须简单。

属性	描述
type	必须 简单。
字符串	
superUsers	超级用户列表。应包含应获得无限访问权限的用户主体列表。
字符串数组	

13.2.19. KafkaAuthorizationOpa 模式参考

用于： *KafkaClusterSpec*

KafkaAuthorizationOpa 模式属性的完整列表

要使用 Open Policy Agent 授权，请将 授权 部分中的 type 属性设置为值 opa，并根据需要配置 OPA 属性。

13.2.19.1. url

用于连接到开放策略代理服务器的 URL。URL 必须包含授权者将查询的策略。必需。

13.2.19.2. allowOnError

定义授权者无法查询 Open Policy Agent 时应默认允许或拒绝 Kafka 客户端，例如，当它暂时不可用时。默认为 false - 将拒绝所有操作。

13.2.19.3. initialCacheCapacity

授权者用于避免针对每个请求查询 Open Policy 代理的本地缓存的初始容量。默认值为 5000。

13.2.19.4. maximumCacheSize

授权者用于避免针对每个请求查询 Open Policy 代理的本地缓存的最大容量。默认值为 50000。

13.2.19.5. expireAfterMs

本地缓存中保留的记录的过期时间，以避免针对每个请求查询 Open Policy 代理。定义从 Open Policy Agent 服务器重新加载缓存的授权决策的频率。以毫秒为单位.默认为 3600000 毫秒（1 小时）。

13.2.19.6. superUsers

被视为超级用户的用户主体列表，以便始终允许他们而无需查询 open Policy Agent 策略。如需更多信息，请参阅 Kafka 授权。

Open Policy Agent 授权器配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    authorization:
      type: opa
      url: http://opa:8181/v1/data/kafka/allow
      allowOnError: false
      initialCacheCapacity: 1000
      maximumCacheSize: 10000
      expireAfterMs: 60000
      superUsers:
        - CN=fred
        - sam
        - CN=edward
    # ...
```

13.2.19.7. KafkaAuthorizationOpa 模式属性

type 属性是一个光盘，它区分使用 KafkaAuthorizationOpa 类型与 KafkaAuthorizationSimple、KafkaAuthorizationKeycloak、KafkaAuthorizationCustom。它必须具有类型 KafkaAuthorizationOpa 的值 opa。

属性	描述
type	必须是 **opa**。

属性	描述
字符串	
url 字符串	用于连接到开放策略代理服务器的 URL。URL 必须包含授权者将查询的策略。这个选项是必需的。
allowOnError 布尔值	定义当授权者无法查询 Open Policy Agent 时（例如，当暂时不可用时），是否应允许或拒绝 Kafka 客户端。默认为 **false** - 将拒绝所有操作。
initialCacheCapacity 整数	授权者用于避免查询开放策略代理的本地缓存的初始容量，每个请求默认值为 **5000**。
maximumCacheSize 整数	授权者用于避免针对每个请求查询 Open Policy 代理的本地缓存的最大容量。默认值为 **50000**。
expireAfterMs 整数	本地缓存中保留的记录的过期时间，以避免针对每个请求查询 Open Policy 代理。定义从 Open Policy Agent 服务器重新加载缓存的授权决策的频率。以毫秒为单位.默认值为 **3600000**。
superUsers 字符串数组	超级用户列表，特别是具有无限访问权限的用户主体列表。

13.2.20. KafkaAuthorizationKeycloak 模式参考

用于： ***KafkaClusterSpec***

type 属性是一个光盘，它区分使用 KafkaAuthorizationKeycloak 类型与 KafkaAuthorizationSimple, KafkaAuthorizationOpa, KafkaAuthorizationCustom。它必须具有类型 KafkaAuthorizationKey cloak 的值 key cloak。

属性	描述
type 字符串	必须为 **keycloak**。

属性	描述
clientId 字符串	Kafka 客户端 ID，可用于对 OAuth 服务器进行身份验证并使用令牌端点 URI。
tokenEndpointUri 字符串	授权服务器令牌端点 URI.
tlsTrustedCertificates CertSecretSource 数组	TLS 连接到 OAuth 服务器的可信证书。
disableTlsHostnameVerification 布尔值	启用或禁用 TLS 主机名验证。默认值为 **false**。
delegateToKafkaAcls 布尔值	如果红帽单点登录授权服务策略 DENIED，则是否应将授权决定委派给"简单"授权者。默认值为 **false**。
grantsRefreshPeriodSeconds 整数	两个连续延长的时间（以秒为单位）运行。默认值为 60。
grantsRefreshPoolSize 整数	用于刷新（active 会话）时授予的线程数。线程越多，并行性越高，作业越快完成。但是，使用更多线程在授权服务器上放置更重的负载。默认值为 5。
superUsers 字符串数组	超级用户列表。应包含应获得无限访问权限的用户主体列表。

13.2.21. KafkaAuthorizationCustom 模式参考

用于： ***KafkaClusterSpec***

KafkaAuthorizationCustom schema 属性的完整列表

要在 AMQ Streams 中使用自定义授权，您可以配置自己的 授权程序插件 来定义访问控制列表 (ACL)。

ACL 允许您定义哪些用户有权在细粒度级别上访问哪些资源。

将 Kafka 自定义资源配置为使用自定义授权。将 授权 部分中的 type 属性设置为 custom 值，再设置以下属性：

重要

自定义授权器必须实施 org.apache.kafka.server.authorizer.Authorizer 界面，并支持使用 super.users 配置属性的 super.users 配置属性配置。

13.2.21.1. authorizerClass

（必需）实施 org.apache.kafka.server.authorizer.Authorizer 接口以支持自定义 ACL 的 Java 类。

13.2.21.2. superUsers

用户主体列表被视为超级用户，因此始终允许他们无需查询 ACL 规则。如需更多信息，请参阅 Kafka 授权。

您可以使用 Kafka.spec.kafka.config 添加用于初始化自定义授权器的配置。

Kafka.spec 下的自定义授权配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    authorization:
      type: custom
      authorizerClass: io.mycompany.CustomAuthorizer
      superUsers:
        - CN=client_1
        - user_2
        - CN=client_3
    # ...
```

```
config:
  authorization.custom.property1=value1
  authorization.custom.property2=value2
# ...
```

除了 Kafka 自定义资源配置外，包含自定义授权器类的 JAR 文件及其依赖关系必须在 Kafka 代理的类路径上可用。

AMQ Streams Maven 构建过程提供了一种将自定义第三方库添加到生成的 Kafka 代理容器镜像的机制，方法是将它们添加为 docker-images/kafka/kafka-third 第三方-libs 目录下的 pom.xml 文件中的依赖项。目录包含不同 Kafka 版本的不同文件夹。选择相应的文件夹。在修改 pom.xml 文件之前，第三方库必须在 Maven 存储库中可用，并且 Maven 存储库必须能被 AMQ Streams 构建流程访问。

注意

Kafka .spec.kafka 中的 config 属性中的 super. user 配置选项将被忽略。在 授权 属性中指定超级用户。如需更多信息，请参阅 Kafka 代理配置。

13.2.21.3. KafkaAuthorizationCustom schema 属性

type 属性是一个光盘，它区分使用 KafkaAuthorizationCustom 类型与 KafkaAuthorizationSimple, KafkaAuthorizationOpa, KafkaAuthorizationKeycloak。它必须具有类型 KafkaAuthorizationCustom 的自定义 值。

属性	描述
type 字符串	必须是 **自定义** 的。
authorizerClass 字符串	授权实施类，必须在类路径中可用。
superUsers 字符串数组	超级用户列表，这些用户是拥有无限访问权限的用户主体。

13.2.22. rack 架构参考

用于： KafkaClusterSpec、KafkaConnectS2ISpec、KafkaConnectSpec

Rack 模式属性的完整列表

机架 选项配置机架感知。机 架 可以表示可用性区域、数据中心或数据中心内的实际机架。机架 通过 topologyKey 配置。topologyKey 在 OpenShift 节点上标识一个标签，其值中包含拓扑名称。此类标签的示例为 topology.kubernetes.io/zone （或较旧版本的 failure-domain.beta.kubernetes.io/zone ），其中包含 OpenShift 节点运行的可用区的名称。您可以将 Kafka 集群配置为了解它运行的 机架，并启用其他功能，如将分区副本分散到不同的机架或使用来自最接近的副本的消息。

如需有关 OpenShift 节点标签的更多信息，请参阅 Well-Known Labels、Annotations 和 Taints。请咨询您的 OpenShift 管理员，了解代表节点要部署到的区域或机架的节点标签。

13.2.22.1. 在机架间分散分区副本

配置机架感知后，AMQ Streams 将为每个 Kafka 代理设置 broker.rack 配置。broker.rack 配置为每个代理分配一个机架 ID。当配置了 broker.rack 时，Kafka 代理会尽可能将分区副本分散到多个不同的机架中。当副本分散到多个机架时，多个副本同时失败的可能性会低于它们在同一机架中。分发副本可以提高弹性，这对于可用性和可靠性至关重要。要在 Kafka 中启用 rack 感知，请在 Kafka 自定义资源的 .spec.kafka 部分添加 rack 选项，如下例所示。

Kafka 的机架 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    rack:
      topologyKey: topology.kubernetes.io/zone
    # ...
```

注意

在某些情况下，当 pod 被删除或重启时，在其中运行代理的机 架 可能会改变。因此，在不同机架中运行的副本可能会共享同一个机架。使用 Cruise Control 和 KafkaRebalance 资源及 RackAwareGoal 以确保副本在不同的机架之间分布。

当在 Kafka 自定义资源中启用机架意识时，AMQ Streams 将自动添加 OpenShift preferredDuringSchedulingIgnoredDuringExecution affinity 规则，以在不同的机架之间分发 Kafka 代理。但是，首选 规则不能保证代理会被分散。根据确切的 OpenShift 和 Kafka 配置，您应该为 ZooKeeper 和 Kafka 添加额外的 关联性 规则或为 ZooKeeper 和 Kafka 配置 topologySpreadConstraints，以确保节点被正确分发到尽量多的机架。如需更多信息，请参阅 第 2.7 节 “配置 pod 调度”。

13.2.22.2. 使用来自最接近副本的消息

机架意识也可以用于消费者从最接近的副本中获取数据。这可用于在 Kafka 集群跨越多个数据中心时减少网络负载，也可以在公共云中运行 Kafka 时降低成本。但是，它可能会导致延迟增加。

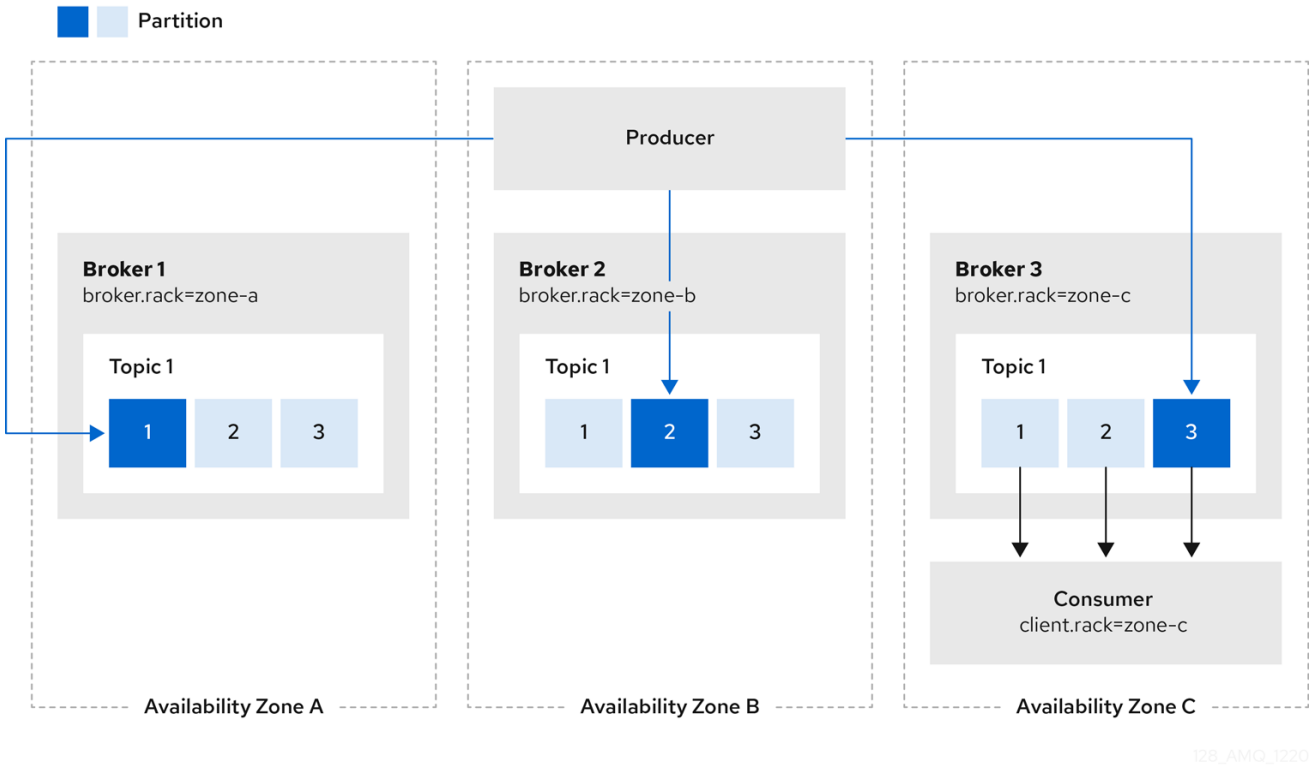
为了能够从最接近的副本中消耗，必须在 Kafka 集群中配置机架感知，并且必须启用 RackAwareReplicaSelector。副本选择器插件提供允许客户端使用最接近的副本的逻辑。默认实施使用 LeaderSelector 来始终为客户端选择领导副本。为 replica.selector.class Specify RackAwareReplicaSelector 从默认实现中切换。

使用启用副本感知选择器的机 架 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    rack:
      topologyKey: topology.kubernetes.io/zone
    config:
      # ...
      replica.selector.class: org.apache.kafka.common.replica.RackAwareReplicaSelector
    # ...
```

除了 Kafka 代理配置外，还需要在消费者中指定 client.rack 选项。client.rack 选项应当指定使用者运行的 机架 ID。RackAwareReplicaSelector 与 broker.rack 和 client.rack ID 匹配，以查找最接近的副本并从其中消耗。如果同一机架中有多个副本，RackAwareReplicaSelector 总是选择最新的副本。如果没有指定机架 ID，或者无法找到具有相同机架 ID 的副本，它将回退到领导副本。

图 13.1. 显示从同一可用区中的副本消耗的客户端示例



在 Kafka Connect 中使用来自最接近的副本的信息也可以在使用信息的 sink 连接器中使用。当使用 AMQ Streams 部署 Kafka Connect 时，您可以使用 KafkaConnect 或 KafkaConnect S2I 自定义资源中的 rack 部分来自动配置 client.rack 选项。

Kafka Connect 的机 架 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
# ...
spec:
  kafka:
    # ...
    rack:
      topologyKey: topology.kubernetes.io/zone
    # ...
```

在 KafkaConnect 或 KafkaConnect S2I 自定义资源中启用 rack 识别功能不会设置任何关联性规则，但您也可以配置 关联性 或 topologySpreadConstraints。如需更多信息，请参阅 第 2.7 节 “配置 pod 调度”。

13.2.22.3. rack 架构属性

属性	描述
topologyKey 字符串	与分配给 OpenShift 集群节点的标签匹配的键。标签的值用于在 Kafka Connect 中设置代理的 **broker.rack** 配置和 **client.rack** 配置。

13.2.23. 探测 模式参考

用于： CruiseControlSpec, EntityTopicOperatorSpec, EntityUserOperatorSpec, KafkaBridgeSpec, KafkaClusterSpec, KafkaConnectS2ISpec, KafkaConnectSpec, KafkaExporterSpec, KafkaMirrorMaker2Spec, KafkaMirrorMakerSpec, TlsSidecar, ZookeeperClusterSpec

属性	描述
failureThreshold 整数	成功后，将探测视为失败的最少连续失败。默认值为 3。最小值为 1.
initialDelaySeconds 整数	首先检查健康检查的第一个前的初始延迟。默认为 15 秒。最小值为 0。
periodSeconds 整数	执行探测的频率（以秒为单位）。默认值为 10 秒。最小值为 1.
successThreshold 整数	最少连续成功探测才会在失败后认为探测成功。默认为 1。必须为 1 代表存活度.最小值为 1.
timeoutSeconds 整数	每次尝试健康检查的超时时间。默认值为 5 秒。最小值为 1.

13.2.24. JvmOptions 架构参考

用于： CruiseControlSpec, EntityTopicOperatorSpec, EntityUserOperatorSpec, KafkaBridgeSpec, KafkaClusterSpec, KafkaConnectS2ISpec, KafkaConnectSpec, KafkaMirrorMaker2Spec, KafkaMirrorMakerSpec, ZookeeperClusterSpec

属性	描述
-XX	JVM 的 -XX 选项映射。
map	
-xms	到 JVM 的 -Xms 选项.
字符串	
-Xmx	JVM 的 -Xmx 选项.
字符串	
gcLoggingEnabled	指定是否启用 Garbage Collection 日志记录。默认值为 false。
布尔值	
javaSystemProperties	包含其他系统属性的映射，将使用 **-D** 选项传递到 JVM。
SystemProperty 数组	

13.2.25. SystemProperty 架构参考

在 中使用： JvmOptions

属性	描述
name	系统属性名称。
字符串	
value	系统属性值。
字符串	

13.2.26. KafkaJmxOptions 模式参考

用于：

KafkaClusterSpec、KafkaConnectS2ISpec、KafkaConnectSpec、KafkaMirrorMaker2Spec

KafkaJmxOptions 模式属性的完整列表

配置 JMX 连接选项。

JMX 指标通过在 9999 上打开 JMX 端口从 Kafka 代理、Kafka Connect 和 MirrorMaker 2.0 获取。使用 jmxOptions 属性配置受密码保护或未受保护的 JMX 端口。使用密码保护可防止未授权容器集访问端口。

然后，您可以获取有关组件的指标。

例如，对于每个 Kafka 代理，您可以从客户端获取字节（以秒为单位）用量数据，或从代理网络获取请求率。

要启用 JMX 端口的安全性，请将 身份验证 字段中的 type 参数设置为 password。

密码保护的 JMX 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    jmxOptions:
      authentication:
        type: "password"
    # ...
  zookeeper:
    # ...
```

然后，您可以通过指定您要寻址的代理来将 pod 部署到集群中，并使用无头服务获取 JMX 指标。

例如，要从代理 0 获取 JMX 指标，您需要指定：

```
"CLUSTER-NAME-kafka-0.CLUSTER-NAME-kafka-brokers"
```

CLUSTER-NAME-kafka-0 是代理 pod 的名称，CLUSTER-NAME-kafka-brokers 是无头服务的名称，用于返回代理 Pod 的 IP。

如果 JMX 端口受到保护，您可以通过在 pod 部署中引用 JMX Secret 来获取用户名和密码。

对于未受保护的 JMX 端口，请使用空对象 {} 打开无头服务上的 JMX 端口。您可以部署 pod 并以与受保护端口相同的方式获取指标，但在这种情况下，任何 pod 都可以从 JMX 端口读取。

打开端口 JMX 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    jmxOptions: {}
    # ...
  zookeeper:
    # ...
```

其它资源

- 有关使用 JMX 公开的 Kafka 组件指标的更多信息，请参阅 Apache Kafka 文档。

13.2.26.1. KafkaJmxOptions 架构属性

属性	描述
身份验证	用于连接 JMX 端口的身份验证配置。类型取决于给定对象中 **authentication.type** 属性的值，它必须是 [password] 之一。

属性	描述
KafkaJmxAuthenticationPassword	

13.2.27. KafkaJmxAuthenticationPassword 架构参考

用于： *KafkaJmxOptions*

type 属性是一个分级器，它区分 KafkaJmxAuthenticationPassword 类型和将来可能添加的其他子类型。它必须具有类型 KafkaJmxAuthenticationPassword 的值 密码。

属性	描述
type	必须为 **password**。
字符串	

13.2.28. JmxPrometheusExporterMetrics schema reference

用来： *CruiseControlSpec, KafkaClusterSpec, KafkaConnectS2ISpec, KafkaConnectSpec, KafkaMirrorMaker2Spec, KafkaMirrorMakerSpec, ZookeeperClusterSpec*

type 属性是一个分级器，它区分 JmxPrometheusExporterMetrics 类型的使用以及将来可能添加的其他子类型。它必须具有 Jm xPrometheusExporter 类型的值jm xPrometheusExporterMetrics。

属性	描述
type	必须为 **jmxPrometheusExporter**。
字符串	
valueFrom	存储 Prometheus JMX Exporter 配置的 ConfigMap 条目。有关此配置结构的详情，请查看 JMX 导出器文档。
ExternalConfigurationReference	

13.2.29. ExternalConfigurationReference 模式参考

用于： *ExternalLogging, JmxPrometheusExporterMetrics*

属性	描述
configMapKeyRef ConfigMapKeySelector	引用包含配置的 ConfigMap 中的键。如需更多信息，请参阅 core/v1 configmapkeyselector 的外部文档。

13.2.30. InlineLogging schema 参考

用于： ***CruiseControlSpec***, ***EntityTopicOperatorSpec***, ***EntityUserOperatorSpec***, ***KafkaBridgeSpec***, ***KafkaClusterSpec***, ***KafkaConnectS2ISpec***, ***KafkaConnectSpec***, ***KafkaMirrorMaker2Spec***, ***KafkaMirrorMakerSpec***, ***ZookeeperClusterSpec***

type 属性是一种分化器，区分 InlineLogging 类型和 ExternalLogging。它必须具有类型 InlineLogging 的值。

属性	描述
type 字符串	必须 **内联.**
日志记录器 map	从日志记录器名称到日志记录器级别的映射.

13.2.31. ExternalLogging 架构参考

用于： ***CruiseControlSpec***, ***EntityTopicOperatorSpec***, ***EntityUserOperatorSpec***, ***KafkaBridgeSpec***, ***KafkaClusterSpec***, ***KafkaConnectS2ISpec***, ***KafkaConnectSpec***, ***KafkaMirrorMaker2Spec***, ***KafkaMirrorMakerSpec***, ***ZookeeperClusterSpec***

type 属性是一个分级器，将 ExternalLogging 类型与 InlineLogging 区分开来。它必须具有类型 ExternalLogging 的值 external。

属性	描述
type 字符串	必须是 **外部的**。

属性	描述
valueFrom	存储日志配置的 **ConfigMap** 条目。
ExternalConfigurationReference	

13.2.32. KafkaClusterTemplate 模式参考

用于： ***KafkaClusterSpec***

属性	描述
statefulset	Kafka **StatefulSet** 模板。
StatefulSetTemplate	
Pod	Kafka **Pod** 模板。
PodTemplate	
bootstrapService	Kafka bootstrap **Service** 模板。
InternalServiceTemplate	
brokersService	Kafka **代理服务** 模板。
InternalServiceTemplate	
externalBootstrapService	Kafka 外部 bootstrap **服务模板**。
ExternalServiceTemplate	
perPodService	用于从 OpenShift 外部访问的 Kafka 每个 pod **服务** 的模板。
ExternalServiceTemplate	
externalBootstrapRoute	Kafka 外部 bootstrap **Route** 模板。
ResourceTemplate	
perPodRoute	用于从 OpenShift 外部访问的 Kafka 每个 pod **路由** 的模板。
ResourceTemplate	

属性	描述
externalBootstrapIngress	Kafka 外部 bootstrap **入口** 模板。
ResourceTemplate	
perPodIngress	用于从 OpenShift 外部访问每个 **pod** 的 Kafka 模板。
ResourceTemplate	
persistentVolumeClaim	所有 Kafka **PersistentVolumeClaims** 的模板。
ResourceTemplate	
podDisruptionBudget	Kafka **PodDisruptionBudget** 模板。
PodDisruptionBudgetTemplate	
kafkaContainer	Kafka 代理容器模板。
ContainerTemplate	
initContainer	Kafka init 容器的模板。
ContainerTemplate	
clusterCaCert	使用 Kafka 集群证书公钥的 Secret 模板。
ResourceTemplate	
serviceAccount	Kafka 服务帐户的模板。
ResourceTemplate	
clusterRoleBinding	Kafka ClusterRoleBinding 模板。
ResourceTemplate	

13.2.33. StatefulSetTemplate 模式参考

用于：KafkaClusterTemplate、ZookeeperClusterTemplate

属性	描述
metadata	应用到资源的元数据。
MetadataTemplate	
podManagementPolicy	用于这个 StatefulSet 的 PodManagementPolicy。有效值为 **Parallel** 和 **OrderedReady**。默认值为 **Parallel**。
string (one of [OrderedReady, Parallel])	

13.2.34. MetadataTemplate 架构参考

用来：DeploymentTemplate、ExternalServiceTemplate、InternalServiceTemplate、PodDisruptionBudgetTemplate、PodTemplate、ResourceTemplate 、 StatefulSetTemplate

MetadataTemplate 架构属性的完整列表

标签 和 注解 用于标识和组织资源，并在 metadata 属性中进行配置。

例如：

```
# ...
template:
  statefulset:
    metadata:
      labels:
        label1: value1
        label2: value2
      annotations:
        annotation1: value1
        annotation2: value2
# ...
```

标签 和 注解 字段可以包含不包含保留字符串 strimzi.io 的任何标签或注解。包含 strimzi.io 的标签和注解由 AMQ Streams 内部使用，无法配置。

13.2.34.1. MetadataTemplate 架构属性

属性	描述

属性	描述
labels	添加至资源模板的标签。可用于不同的资源，如 **StatefulSet**、 **Deployment**、**Pod** 和 **Services**。
map	
annotations	添加到资源模板的注解。可用于不同的资源，如 **StatefulSet**、 **Deployment**、**Pod** 和 **Services**。
map	

13.2.35. PodTemplate 架构参考

用来： CruiseControlTemplate, EntityOperatorTemplate, KafkaBridgeTemplate, KafkaClusterTemplate, KafkaConnectTemplate, KafkaExporterTemplate, KafkaMirrorMakerTemplate, ZookeeperClusterTemplate

PodTemplate 架构属性的完整列表

为 Kafka Pod 配置模板。

PodTemplate 配置示例

```
# ...
template:
  pod:
    metadata:
      labels:
        label1: value1
      annotations:
        anno1: value1
    imagePullSecrets:
      - name: my-docker-credentials
    securityContext:
      runAsUser: 1000001
      fsGroup: 0
    terminationGracePeriodSeconds: 120
# ...
```

13.2.35.1. hostAliases

使用 hostAliases 属性来指定主机和 IP 地址的列表，它们注入到容器集的 /etc/hosts 文件中。

当用户请求集群外的连接时，此配置对于 Kafka Connect 或 MirrorMaker 特别有用。

hostAliases 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
#...
spec:
  # ...
  template:
    pod:
      hostAliases:
      - ip: "192.168.1.86"
        hostnames:
        - "my-host-1"
        - "my-host-2"
      #...
```

13.2.35.2. PodTemplate 架构属性

属性	描述
metadata	应用到资源的元数据。
MetadataTemplate	
imagePullSecrets	对同一命名空间中的 secret 的引用列表，用于拉取此 Pod 使用的任何镜像。当 Cluster Operator 中的 **STRIMZI_IMAGE_PULL_SECRETS** 环境变量和 **imagePullSecrets** 选项被指定时，只使用 **imagePullSecrets** 变量，并且会忽略 **STRIMZI_IMAGE_PULL_SECRETS** 变量。如需更多信息，请参阅内核/v1 本地对象引用的外部文档。
LocalObjectReference 数组	
securityContext	配置 Pod 级别安全属性和通用容器设置。如需更多信息，请参阅 core/v1 podsecuritycontext 的外部文档。
PodSecurityContext	

属性	描述
terminationGracePeriodSeconds 整数	宽限期是向容器集中运行的进程发送终止信号后，以及进程通过 kill 信号强制停止的时间（以秒为单位）。将此值设置为比您的进程预期的清理时间更长。值必须是非负整数。零值表示立即删除。您可能需要为非常大的 Kafka 集群增加宽限期，以便 Kafka 代理有足够的时间在终止前将其工作转移到另一个代理。默认值为 30 秒。
关联性 关联性	pod 的关联性规则。如需更多信息，请参阅有关 内核/v1 关联性的外部文档。
容限（tolerations） 容限 数组	pod 的容限。如需更多信息，请参阅有关 内核/v1 容限的外部文档。
priorityClassName 字符串	用于为 pod 分配优先级的优先级类的名称。如需有关优先级类的更多信息，请参阅 Pod 优先级和抢占。
schedulerName 字符串	用于分配此 **Pod** 的调度程序的名称。如果未指定，将使用默认调度程序。
hostAliases HostAlias 数组	容器集的 HostAliases.HostAliases 是主机和 IP 的可选列表，在指定后将注入到 Pod 的主机文件中。如需更多信息，请参阅 core/v1 HostAlias 的外部文档。
enableServiceLinks 布尔值	指明有关服务的信息应该注入到 Pod 的环境变量中。
topologySpreadConstraints TopologySpreadConstraint 数组	pod 的拓扑分布约束。如需更多信息，请参阅 core/v1 topologyspreadconstraint 的外部文档。

13.2.36. InternalServiceTemplate 模式参考

用来：
CruiseControlTemplate、*KafkaBridgeTemplate*、*KafkaClusterTemplate*、*KafkaConnectTemplate*、*ZookeeperClusterTemplate*

属性	描述
metadata	应用到资源的元数据。
MetadataTemplate	
ipFamilyPolicy	指定服务使用的 IP 系列策略。可用选项包括 **SingleStack**、**PreferDualStack** 和 **RequireDualStack**。**SingleStack** 适用于 单个 IP 系列。**PreferDualStack** 是两个 IP 系列，位于双栈配置的群集上，或者单堆栈集群中只有一个 IP 系列。**RequireDualStack** 会失败，除非双栈配置的群集上有两个 IP 系列。如果未指定，OpenShift 将根据服务类型选择默认值。适用于 OpenShift 1.20 及更新版本.
string (one of [RequireDualStack, SingleStack, PreferDualStack])	
ipFamilies	指定服务使用的 IP 系列。可用的选项有 **IPv4** 和 **IPv6。如果未指定，OpenShift 将根据 'ipFamilyPolicy** 设置选择默认值。适用于 OpenShift 1.20 及更新版本.
字符串（[IPv6、IPv4]）数组的一个或多个	

13.2.37. ExternalServiceTemplate 模式引用

用于： ***KafkaClusterTemplate***

ExternalServiceTemplate 架构属性的完整列表

当使用 loadbalancers 或节点端口在 OpenShift 外部公开 Kafka 时，除了使用标签和注释外，您还可以使用属性来自定义服务的创建方式。

显示自定义外部服务的示例

```
# ...
template:
  externalBootstrapService:
    externalTrafficPolicy: Local
    loadBalancerSourceRanges:
      - 10.0.0.0/8
      - 88.208.76.87/32
  perPodService:
    externalTrafficPolicy: Local
    loadBalancerSourceRanges:
```

```
      - 10.0.0.0/8
      - 88.208.76.87/32
  # ...
```

13.2.37.1. ExternalServiceTemplate 架构属性

属性	描述
metadata	应用到资源的元数据。
MetadataTemplate	

13.2.38. resourcetemplate 架构引用

用于： CruiseControlTemplate, EntityOperatorTemplate, KafkaBridgeTemplate, KafkaClusterTemplate, KafkaConnectTemplate, KafkaExporterTemplate, KafkaMirrorMakerTemplate, KafkaUserTemplate, ZookeeperClusterTemplate

属性	描述
metadata	应用到资源的元数据。
MetadataTemplate	

13.2.39. PodDisruptionBudgetTemplate 模式参考

用来： CruiseControlTemplate、KafkaBridgeTemplate、KafkaClusterTemplate、KafkaConnectTemplate、KafkaMirrorMakerTemplate、ZookeeperClusterTemplate

PodDisruptionBudgetTemplate 模式属性的完整列表

AMQ Streams 为每个新的 StatefulSet 或 Deployment 创建一个 PodDisruptionBudget。默认情况下，pod 中断预算只允许单个 pod 在给定时间不可用。您可以通过更改 PodDisruptionBudget.spec 资源中的 maxUnavailable 属性的默认值来增加允许的不可用 pod 数量。

PodDisruptionBudget 模板示例

```
# ...
template:
    podDisruptionBudget:
        metadata:
            labels:
                key1: label1
                key2: label2
            annotations:
                key1: label1
                key2: label2
        maxUnavailable: 1
# ...
```

13.2.39.1. PodDisruptionBudgetTemplate 架构属性

属性	描述
metadata	应用到 **PodDistruptionBugetTemplate** 资源的元数据。
MetadataTemplate	
maxUnavailable	不可用 pod 的最大数量，以允许自动 pod 驱除。如果驱除后无法使用 **maxUnavailable** 数量的 pod 或更少 pod，则允许 Pod 驱除。将此值设置为 0 可防止所有自愿驱除，因此 pod 必须手动驱除。默认为 1。
整数	

13.2.40. ContainerTemplate 架构参考

用来： ***CruiseControlTemplate, EntityOperatorTemplate, KafkaBridgeTemplate, KafkaClusterTemplate, KafkaConnectTemplate, KafkaExporterTemplate, KafkaMirrorMakerTemplate, ZookeeperClusterTemplate***

ContainerTemplate 架构属性的完整列表

您可以为容器设置自定义安全上下文和环境变量。

环境变量在 ***env*** *属性下定义，作为具有* ***名称和值*** *字段的对象列表。以下示例显示了为* ***Kafka*** *代理容器设置的两个自定义环境变量和自定义安全上下文：*

```
# ...
template:
  kafkaContainer:
    env:
    - name: EXAMPLE_ENV_1
      value: example.env.one
    - name: EXAMPLE_ENV_2
      value: example.env.two
    securityContext:
      runAsUser: 2000
# ...
```

前缀为 KAFKA_ 的环境变量是 AMQ Streams 内部的环境变量，应当避免。如果您设置了 AMQ Streams 已使用的自定义环境变量，则会忽略它，并在日志中记录警告。

13.2.40.1. ContainerTemplate 架构属性

属性	描述
env	应用于容器的环境变量。
ContainerEnvVar 数组	
securityContext	容器的安全上下文.如需更多信息，请参阅内核/v1 安全上下文的外部文档。
SecurityContext	

13.2.41. ContainerEnvVar 模式参考

中使用的： ***ContainerTemplate***

属性	描述
name	环境变量键。
字符串	
value	环境变量值。
字符串	

13.2.42. ZookeeperClusterSpec 模式参考

中使用的： KafkaSpec

ZookeeperClusterSpec 模式属性的完整列表

配置 ZooKeeper 集群。

13.2.42.1. config

使用 配置 属性将 ZooKeeper 选项配置为键。

标准 Apache ZooKeeper 配置可能会提供，仅限于那些不由 AMQ Streams 直接管理的属性。

无法配置的选项与以下内容相关：

- 安全（加密、身份验证和授权）
- 监听程序配置
- 数据目录的配置
- zookeeper 集群组成

这些值可以是以下 JSON 类型之一：

- 字符串
- 数字
- 布尔值

您可以指定和配置 ZooKeeper 文档 中列出的选项，但由 AMQ Streams 直接管理的选项除外。具体来说，所有键为等于或以以下任一字符串开头的配置选项将被禁止：

- 服务器.
- dataDir
- dataLogDir
- clientPort
- authProvider
- quorum.auth
- requireClientAuthScheme

当 config 属性中存在禁止选项时，会忽略它，并把警告信息输出到 Cluster Operator 日志文件中。所有其他支持的选项都传递给 ZooKeeper。

禁止的选项有例外。对于使用特定 密码套件 作为 TLS 版本进行客户端连接，您可以配置 allowed ssl 属性。

ZooKeeper 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
  zookeeper:
    # ...
    config:
      autopurge.snapRetainCount: 3
```

```
    autopurge.purgeInterval: 1
    ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384"
    ssl.enabled.protocols: "TLSv1.2"
    ssl.protocol: "TLSv1.2"
  # ...
```

13.2.42.2. logging

zookeeper 具有可配置的日志记录器：

- zookeeper.root.logger

zookeeper 使用 Apache log4j 日志记录器实施。

使用 logging 属性来配置日志记录器和日志记录器级别。

您可以通过直接（内线）指定日志记录器和级别来设置日志级别，或使用自定义（外部）ConfigMap。如果使用 ConfigMap，则将 logging.valueFrom.configMapKeyRef.name 属性设置为包含外部日志记录配置的 ConfigMap 的名称。在 ConfigMap 中，日志配置使用 log4j.properties 进行 描述。logging.valueFrom.configMapKeyRef.name 和 logging.valueFrom.configMapKeyRef.key 属性均是必需的。在运行 Cluster Operator 时，会使用自定义资源创建使用指定准确日志配置的 ConfigMap，然后在每次协调后重新创建。如果没有指定自定义 ConfigMap，则会使用默认日志设置。如果没有设置特定的日志记录器值，则会继承该日志记录器的上一级日志记录器设置。有关日志级别的更多信息，请参阅 Apache 日志记录服务。

此处我们会看到 内联 和外部 记录示例。

内联日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # ...
  zookeeper:
    # ...
    logging:
      type: inline
```

```
    loggers:
      zookeeper.root.logger: "INFO"
  # ...
```

外部日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # ...
  zookeeper:
    # ...
    logging:
      type: external
      valueFrom:
        configMapKeyRef:
          name: customConfigMap
          key: zookeeper-log4j.properties
  # ...
```

垃圾收集器(GC)

也可以使用 jvmOptions 属性 来启用（或禁用）垃圾收集器日志记录。

13.2.42.3. ZookeeperClusterSpec 模式属性

属性	描述
replicas	集群中的 pod 数量。
整数	
image	容器集的 docker 镜像。
字符串	
storage	存储配置（磁盘）。无法更新。类型取决于给定对象中 **storage.type** 属性的值，它必须是 [ephemeral, persistent-claim] 之一。
EphemeralStorage, PersistentClaimStorage	

属性	描述
config	ZooKeeper 代理配置.无法设置具有以下前缀的属性：server.、dataDir、dataLogDir、clientPort、authProvider、quorum.auth、requireClientAuthScheme、snapshot.trust.empty、standaloneEnabled、reconfigEnabled、4lw.commands.whitelist、secureClientPort. SSL.、serverCnxnFactory, sslQuorum（除了：ssl.protocol, ssl.quorum.protocol, ssl.enabledProtocols, ssl.quorum.enabledProtocols, ssl.ciphersuites, ssl.ciphersuites, SSL.quorum.ciphersuites、ssl.hostnameVerification、ssl.quorum.hostnameVerification。
map	
livenessProbe	Pod 存活度检查.
probe	
readinessprobe	Pod 就绪度检查。
probe	
jvmOptions	容器集的 JVM 选项.
JvmOptions	
资源	要保留的 CPU 和内存资源。如需更多信息，请参阅核心/v1 资源要求的外部文档。
ResourceRequirements	
metricsConfig	指标配置.这个类型取决于给定对象中 **metricsConfig.type** 属性的值，必须是 [jmxPrometheusExporter] 中的一个。
JmxPrometheusExporterMetrics	
logging	ZooKeeper 的日志配置.类型取决于给定对象中的 **logging.type** 属性的值，它必须是 [inline, external] 之一。
InlineLogging, **ExternalLogging**	
模板	ZooKeeper 集群资源的模板。该模板允许用户指定 **StatefulSet、Pod 和服务** 生成的方式。
ZookeeperClusterTemplate	

13.2.43. ZookeeperClusterTemplate schema reference

中使用的： ***ZookeeperClusterSpec***

属性	描述
statefulset	ZooKeeper **StatefulSet** 模板.
StatefulSetTemplate	
Pod	ZooKeeper **Pod** 模板。
PodTemplate	
clientService	ZooKeeper 客户端 **服务模板.**
InternalServiceTemplate	
nodesService	ZooKeeper 节点 **服务模板.**
InternalServiceTemplate	
persistentVolumeClaim	所有 ZooKeeper **PersistentVolumeClaims** 模板.
ResourceTemplate	
podDisruptionBudget	ZooKeeper **PodDisruptionBudget** 模板。
PodDisruptionBudgetTemplate	
zookeeperContainer	ZooKeeper 容器的模板。
ContainerTemplate	
serviceAccount	ZooKeeper 服务帐户的模板。
ResourceTemplate	

13.2.44. EntityOperatorSpec 模式参考

中使用的： ***KafkaSpec***

属性	描述
topicOperator	配置主题 Operator。
EntityTopicOperatorSpec	

属性	描述
userOperator	User Operator 配置.
EntityUserOperatorSpec	
tlsSidecar	TLS sidecar 配置.
TlsSidecar	
模板	Entity Operator 资源模板。该模板允许用户指定如何生成 **Deployment** 和 **Pod**。
EntityOperatorTemplate	

13.2.45. EntityTopicOperatorSpec schema reference

在以下位置使用： EntityOperatorSpec

EntityTopicOperatorSpec 模式属性的完整列表

配置主题 Operator。

13.2.45.1. logging

Topic Operator 具有可配置的日志记录器：

- *rootLogger.level*

Topic Operator 使用 Apache log4j2 日志记录器实现。

使用 Kafka 资源 Kafka 资源的 principalOperator.topicOperator 字段中的 logging 属性来配置日志记录器和日志记录器级别。

您可以通过直接（内线）指定日志记录器和级别来设置日志级别，或使用自定义（外部）ConfigMap。如果使用 ConfigMap，则将 logging.valueFrom.configMapKeyRef.name 属性设置为包含外部日志记录配置的 ConfigMap 的名称。在 ConfigMap 中，日志配置使用 log4j2.properties 进行 描述。logging.valueFrom.configMapKeyRef.name 和 logging.valueFrom.configMapKeyRef.key

属性均是必需的。在运行 Cluster Operator 时，会使用自定义资源创建使用指定准确日志配置的 ConfigMap，然后在每次协调后重新创建。如果没有指定自定义 ConfigMap，则会使用默认日志设置。如果没有设置特定的日志记录器值，则会继承该日志记录器的上一级日志记录器设置。有关日志级别的更多信息，请参阅 Apache 日志记录服务。

此处我们会看到 内联 和外部 记录示例。

内联日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    topicOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
      logging:
        type: inline
        loggers:
          rootLogger.level: INFO
  # ...
```

外部日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    topicOperator:
```

```
    watchedNamespace: my-topic-namespace
    reconciliationIntervalSeconds: 60
    logging:
      type: external
      valueFrom:
        configMapKeyRef:
          name: customConfigMap
          key: topic-operator-log4j2.properties
# ...
```

垃圾收集器(GC)

也可以使用 jvmOptions 属性 来启用（或禁用）垃圾收集器日志记录。

13.2.45.2. EntityTopicOperatorSpec schema properties

属性	描述
watchedNamespace	Topic Operator 应监视的命名空间。
字符串	
image	用于主题 Operator 的镜像。
字符串	
reconciliationIntervalSeconds	定期协调之间的间隔。
整数	
zookeeperSessionTimeoutSeconds	ZooKeeper 会话超时。
整数	
startupProbe	Pod 启动检查。
probe	
livenessProbe	Pod 存活度检查.
probe	
readinessprobe	Pod 就绪度检查。

属性	描述
probe	
资源	要保留的 CPU 和内存资源。如需更多信息，请参阅核心/v1 资源要求的外部文档。
ResourceRequirements	
topicMetadataMaxAttempts	获取主题元数据的尝试数量。
整数	
logging	日志记录配置.类型取决于给定对象中的 **logging.type** 属性的值，它必须是 [inline, external] 之一。
InlineLogging, **ExternalLogging**	
jvmOptions	容器集的 JVM 选项.
JvmOptions	

13.2.46. EntityUserOperatorSpec 模式参考

在以下位置使用： ***EntityOperatorSpec***

EntityUserOperatorSpec 模式属性的完整列表

配置 User Operator。

13.2.46.1. logging

User Operator 具有可配置的日志记录器：

- *rootLogger.level*

User Operator 使用 Apache log4j2 日志记录器实施。

使用 Kafka 资源的 principal Operator.userOperator 字段中的 logging 属性来配置日志记录器和日志记录器级别。

您可以通过直接（内线）指定日志记录器和级别来设置日志级别，或使用自定义（外部）ConfigMap。如果使用 ConfigMap，则将 logging.valueFrom.configMapKeyRef.name 属性设置为包含外部日志记录配置的 ConfigMap 的名称。在 ConfigMap 中，日志配置使用 log4j2.properties 进行 描述。logging.valueFrom.configMapKeyRef.name 和 logging.valueFrom.configMapKeyRef.key 属性均是必需的。在运行 Cluster Operator 时，会使用自定义资源创建使用指定准确日志配置的 ConfigMap，然后在每次协调后重新创建。如果没有指定自定义 ConfigMap，则会使用默认日志设置。如果没有设置特定的日志记录器值，则会继承该日志记录器的上一级日志记录器设置。有关日志级别的更多信息，请参阅 Apache 日志记录服务。

此处我们会看到 内联 和外部 记录示例。

内联日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    userOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
      logging:
        type: inline
        loggers:
          rootLogger.level: INFO
  # ...
```

外部日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
```

```
kafka:
  # ...
zookeeper:
  # ...
entityOperator:
  # ...
  userOperator:
    watchedNamespace: my-topic-namespace
    reconciliationIntervalSeconds: 60
    logging:
      type: external
      valueFrom:
        configMapKeyRef:
          name: customConfigMap
          key: user-operator-log4j2.properties
# ...
```

垃圾收集器(GC)

也可以使用 jvmOptions 属性 来启用（或禁用）垃圾收集器日志记录。

13.2.46.2. EntityUserOperatorSpec 模式属性

属性	描述
watchedNamespace	User Operator 应监视的命名空间。
字符串	
image	用于 User Operator 的镜像。
字符串	
reconciliationIntervalSeconds	定期协调之间的间隔。
整数	
zookeeperSessionTimeoutSeconds	ZooKeeper 会话超时。
整数	
secretPrefix	添加到 KafkaUser 名称中的前缀，以用作 Secret 名称。
字符串	

属性	描述
livenessProbe	Pod 存活度检查.
probe	
readinessprobe	Pod 就绪度检查。
probe	
资源	要保留的 CPU 和内存资源。如需更多信息，请参阅核心/v1 资源要求的外部文档。
ResourceRequirements	
logging	日志记录配置.类型取决于给定对象中的 **logging.type** 属性的值，它必须是 [inline, external] 之一。
InlineLogging, **ExternalLogging**	
jvmOptions	容器集的 JVM 选项.
JvmOptions	

13.2.47. TlsSidecar 架构参考

使用： CruiseControlSpec, EntityOperatorSpec

TlsSidecar 架构属性的完整列表

配置 TLS sidecar，这是在 pod 中运行的容器，但具有支持性。在 AMQ Streams 中，TLS sidecar 使用 TLS 来加密和解密组件和 ZooKeeper 之间的通信。

TLS sidecar 用于：

- *实体 Operator*
- *Sything Control*

TLS sidecar 使用以下 tlsSidecar 属性进行配置：

- *Kafka.spec.entityOperator*
- *Kafka.spec.cruiseControl*

TLS sidecar 支持以下附加选项：

- *image*
- *资源*
- *logLevel*
- *readinessProbe*
- *livenessProbe*

resources 属性指定为 TLS sidecar 分配的 内存和 CPU 资源。

image 属性配置 要使用的容器镜像。

readinessProbe 和 livenessProbe 属性为 TLS sidecar 配置 健康检查探测。

logLevel 属性指定日志级别。支持以下日志记录级别：

- *emerg*

- *alert*
- *crit*
- *err*
- *warning*
- *备注*
- *info*
- *debug*

默认值为 notice。

TLS sidecar 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  entityOperator:
    # ...
    tlsSidecar:
      resources:
        requests:
          cpu: 200m
          memory: 64Mi
        limits:
          cpu: 500m
          memory: 128Mi
    # ...
  cruiseControl:
    # ...
    tlsSidecar:
```

```
    image: my-org/my-image:latest
    resources:
      requests:
        cpu: 200m
        memory: 64Mi
      limits:
        cpu: 500m
        memory: 128Mi
    logLevel: debug
    readinessProbe:
      initialDelaySeconds: 15
      timeoutSeconds: 5
    livenessProbe:
      initialDelaySeconds: 15
      timeoutSeconds: 5
  # ...
```

13.2.47.1. TlsSidecar 架构属性

属性	描述
image	容器的 Docker 镜像。
字符串	
livenessProbe	Pod 存活度检查.
probe	
logLevel	TLS sidecar 的日志级别。默认值为 **notice**。
字符串（[emerg、debug、crit、err、alert、warning、notice、info] 之一）	
readinessprobe	Pod 就绪度检查。
probe	
资源	要保留的 CPU 和内存资源。如需更多信息，请参阅核心/v1 资源要求的外部文档。
ResourceRequirements	

13.2.48. EntityOperatorTemplate 模式参考

在以下位置使用： EntityOperatorSpec

属性	描述
Deployment	Entity Operator **部署** 模板.
ResourceTemplate	
Pod	Entity Operator **Pod** 模板。
PodTemplate	
topicOperatorContainer	Entity Topic Operator 容器的模板。
ContainerTemplate	
userOperatorContainer	Entity User Operator 容器的模板。
ContainerTemplate	
tlsSidecarContainer	Entity Operator TLS sidecar 容器的模板。
ContainerTemplate	
serviceAccount	Entity Operator 服务帐户的模板。
ResourceTemplate	

13.2.49. certificateAuthority schema 参考

中使用的： KafkaSpec

配置集群中如何使用 TLS 证书。这适用于集群内部通信的证书，也适用于通过 Kafka.spec.kafka.listeners.tls 访问客户端的证书。

属性	描述
generateCertificateAuthority	如果为 true，则会自动生成证书颁发机构证书。否则，用户将需要为 Secret 提供 CA 证书。默认值为 true。
布尔值	

属性	描述
generateSecretOwnerReference 布尔值	如果为 **true**，Cluster 和 Client CA Secrets**会** 被配置为 **Kafka** 资源。如果在 **true** 时删除 **Kafka** 资源，则 CA Secret 也会被删除。如果为 **false**，则禁用 **ownerReference**。如果 **Kafka** 资源在 **false** 时被删除，则 CA Secret 将保留并可供重复使用。默认值为 **true**。
validityDays 整数	生成的证书的天数应有效。默认值为 365。
renewalDays 整数	证书续订期间的天数。这是证书过期前的天数，可在其中执行续订操作。当 **generateCertificateAuthority** 为 true 时，这将导致新证书的生成。当 **generateCertificateAuthority** 为 true 时，这会导致在 WARN 级别上记录待处理证书过期时间。默认值为 30。
certificateExpirationPolicy 字符串（[replace-key,new-certificate] 之一）	**generateCertificateAuthority=true** 时应如何处理 CA 证书过期时间。默认为重新使用现有私钥生成新 CA 证书。

13.2.50. CruiseControlSpec 模式参考

中使用的： ***KafkaSpec***

属性	描述
image 字符串	容器集的 docker 镜像。
tlsSidecar **TlsSidecar**	TLS sidecar 配置.
资源 ResourceRequirements	要保留用于 Cruise 控制容器的 CPU 和内存资源。如需更多信息，请参阅核心/v1 资源要求的外部文档。
livenessProbe	对 Cruise 控制容器的 Pod 存活度检查。

属性	描述
probe	
readinessprobe	对 Cruise 控制容器的 Pod 就绪度检查。
probe	
jvmOptions	清理控制容器的 JVM 选项.
JvmOptions	
logging	用于崩溃控制的日志记录配置(Log4j 2)类型取决于给定对象中的 **logging.type** 属性的值，它必须是 [inline, external] 之一。
InlineLogging, **ExternalLogging**	
模板	模板，以指定如何生成 Cruise Control 资源、**部署和 Pod**。
CruiseControlTemplate	
brokerCapacity	Cruise 控制 **代理功能** 配置.
BrokerCapacity	
config	清理控制配置.有关配置选项的完整列表，请参阅 https://github.com/linkedin/cruise-control/wiki/Configurations。请注意，无法使用以下前缀设置以下属性：bootstrap.servers、client.id、zookeeper.、network.、security.、fail.brokers.zk.path、webserver.http.、webserver.api.urlprefix、webserver.session.path、webserver.accesslog.、2.step. request.reason.required,metric.reporter.sampler.bootstrap.servers, metric.reporter.topic, partition.metric.sample.store.topic, broker.metric.sample.store.topic,capacity.config.file, self.healing. anomaly.detection.ssl.（除：ssl.cipher.suites 除外） SSL.protocol, ssl.enabled.protocols, webserver.http.cors.enabled,webserver.http.cors.origin, webserver.http.cors.exposeheaders。
map	
metricsConfig	指标配置.这个类型取决于给定对象中 **metricsConfig.type** 属性的值，必须是 [jmxPrometheusExporter] 中的一个。
JmxPrometheusExporterMetrics	

13.2.51. CruiseControlTemplate 模式参考

用法：Cru iseControlSpec

属性	描述
Deployment	用于清理控制 **部署的模板.**
ResourceTemplate	
Pod	清理控制 **Pod** 模板。
PodTemplate	
apiService	Cruise Control API **服务模板**。
InternalServiceTemplate	
podDisruptionBudget	Cruise Control **PodDisruptionBudget** 模板。
PodDisruptionBudgetTemplate	
cruiseControlContainer	Cruise Control 容器的模板。
ContainerTemplate	
tlsSidecarContainer	Cruise Control TLS sidecar 容器的模板。
ContainerTemplate	
serviceAccount	Cruise Control 服务帐户的模板。
ResourceTemplate	

13.2.52. BrokerCapacity schema 参考

用法：Cru iseControlSpec

属性	描述
disk	磁盘的代理容量（以字节为单位），例如 100Gi。
字符串	

属性	描述
cpuUtilization	CPU 资源利用率的代理容量以百分比（0 到 100）为单位。
整数	
inboundNetwork	入站网络吞吐量的代理容量（以每秒字节为单位），例如 10000KB/s。
字符串	
outboundNetwork	代理容量，以每秒字节为单位实现出站网络吞吐量，例如 10000KB/s。
字符串	

13.2.53. KafkaExporterSpec 模式参考

中使用的： ***KafkaSpec***

属性	描述
image	容器集的 docker 镜像。
字符串	
groupRegex	用于指定要收集哪些消费者组的正则表达式。默认值为 **.***。
字符串	
topicRegex	用于指定要收集哪些主题的正则表达式。默认值为 **.***。
字符串	
资源	要保留的 CPU 和内存资源。如需更多信息，请参阅核心/v1 资源要求的外部文档。
ResourceRequirements	
logging	仅记录具有给定严重性或以上的日志消息。有效级别：[**debug**、**info**、warn、**error**、**fatal**]。默认日志级别为 **info**。
字符串	
enableSaramaLogging	启用 Sarama 日志记录，即 Kafka Exporter 使用的 Go 客户端库。
布尔值	

属性	描述
模板	自定义部署模板和容器集.
KafkaExporterTemplate	
livenessProbe	Pod 存活度检查。
probe	
readinessprobe	Pod 就绪度检查。
probe	

13.2.54. KafkaExporterTemplate 模式参考

用于： ***KafkaExporterSpec***

属性	描述
Deployment	Kafka 导出器 **部署的模板.**
ResourceTemplate	
Pod	Kafka 导出器 **Pod** 模板.
PodTemplate	
service	**service 属性已弃用。**Kafka Exporter 服务已被删除。Kafka Exporter **Service** 模板.
ResourceTemplate	
Container	Kafka Exporter 容器的模板。
ContainerTemplate	
serviceAccount	Kafka Exporter 服务帐户的模板。
ResourceTemplate	

13.2.55. KafkaStatus 架构参考

用于： ***Kafka***

属性	描述
conditions	状态条件列表。
条件 数组	
observedGeneration	生成 Operator 最后协调的 CRD。
整数	
监听程序	内部和外部监听器的地址。
ListenerStatus 数组	
clusterId	Kafka 集群 Id.
字符串	

13.2.56. 条件 架构参考

用于： ***KafkaBridgeStatus*, *KafkaConnectorStatus*, *KafkaConnectS2IStatus*, *KafkaConnectStatus*, *KafkaMirrorMaker2Status*, *KafkaMirrorMakerStatus*, *KafkaRebalanceStatus*, *KafkaStatus*, *KafkaTopicStatus*, *KafkaUserStatus***

属性	描述
type	条件的唯一标识符，用于区分资源中的其他条件。
字符串	
status	条件的状态，可以是 True、False 或 Unknown。
字符串	
lastTransitionTime	上次类型条件从一个状态变为另一个状态的时间。所需格式为 UTC 时区中的 'yyyy-MM-ddTHH:mm:ssZ'。
字符串	
reason	该状况上次转换的原因（CamelCase 中的单个单词）。
字符串	
message	指示状况最后一次转换的详细信息的人类可读消息。
字符串	

13.2.57. ListenerStatus 模式参考

用于： *KafkaStatus*

属性	描述
type 字符串	侦听器的类型。可以是以下三种类型之一：**plain**、**tls** 和 **external**。
地址 **ListenerAddress** 数组	此侦听器的地址列表。
bootstrapServers 字符串	用于使用此侦听器连接 Kafka 集群的 **host:port** 对的逗号分隔列表。
证书 字符串数组	TLS 证书列表，可用于在连接到给定侦听器时验证服务器的身份。仅为 **tls 和外部** 监听程序设置。

13.2.58. ListenerAddress 模式参考

用于： *ListenerStatus*

属性	描述
主机 字符串	Kafka bootstrap 服务的 DNS 名称或 IP 地址。
port 整数	Kafka bootstrap 服务的端口。

13.2.59. KafkaConnect 模式参考

属性	描述
spec	Kafka Connect 集群的规格。

属性	描述
KafkaConnectSpec	
status	Kafka Connect 集群的状态。
KafkaConnectStatus	

13.2.60. KafkaConnectSpec 模式参考

用于： *KafkaConnect*

KafkaConnectSpec schema 属性的完整列表

配置 Kafka Connect 集群。

13.2.60.1. config

使用 config 属性将 Kafka 选项配置为密钥。

标准 Apache Kafka Connect 配置可能会提供，仅限于那些不是由 AMQ Streams 直接管理的属性。

无法配置的选项与以下内容相关：

- *Kafka 集群 bootstrap 地址*
- *安全（加密、身份验证和授权）*
- *侦听器/REST 接口配置*
- *插件路径配置*

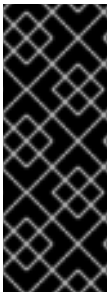
这些值可以是以下 JSON 类型之一：

- 字符串
- 数字
- 布尔值

您可以指定并配置 Apache Kafka 文档 中列出的选项，但那些直接由 AMQ Streams 管理的选项除外。具体来说，键等于或以以下任一字符串开头的配置选项将被禁止：

- ssl.
- sasl.
- 安全性.
- 监听程序
- plugin.path
- rest.
- bootstrap.servers

当 config 属性中存在禁止选项时，会忽略它，并把警告信息输出到 Cluster Operator 日志文件中。所有其他选项将传递给 Kafka Connect。



重要

Cluster Operator 不会验证提供的 config 对象中的密钥或值。当提供无效的配置时，Kafka Connect 集群可能无法启动或变得不稳定。在这种情形中，修复 KafkaConnect.spec.config 或 KafkaConnectS2I.spec.config 对象中的配置，然后 Cluster Operator 可将新配置部署到所有 Kafka Connect 节点。

某些选项有默认值：

- 默认值 connect-cluster 的 Group.id
- 默认值为 connect-cluster-offsets的 offset.storage.topic
- 带有默认值 connect-cluster-configs的 config.storage.topic
- status.storage.topic，默认值为 connect-cluster-status
- key.converter with default value org.apache.kafka.connect.json.JsonConverter
- value.converter with default value org.apache.kafka.connect.json.JsonConverter

如果 KafkaConnect.spec. config 或 KafkaConnectS2I.spec.config 属性中不存在这些选项，则会自动配置这些选项。

禁止的选项有例外。您可以使用三个允许的 ssl 配置选项用于使用 TLS 版本的特定 密码套件 进行客户端连接。密码套件组合了用于安全连接和数据传输的算法。您还可以配置 ssl.endpoint.identification.algorithm 属性来 启用或禁用主机名验证。

Kafka Connect 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
```

```
  name: my-connect
spec:
  # ...
  config:
    group.id: my-connect-cluster
    offset.storage.topic: my-connect-cluster-offsets
    config.storage.topic: my-connect-cluster-configs
    status.storage.topic: my-connect-cluster-status
    key.converter: org.apache.kafka.connect.json.JsonConverter
    value.converter: org.apache.kafka.connect.json.JsonConverter
    key.converter.schemas.enable: true
    value.converter.schemas.enable: true
    config.storage.replication.factor: 3
    offset.storage.replication.factor: 3
    status.storage.replication.factor: 3
    ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384"
    ssl.enabled.protocols: "TLSv1.2"
    ssl.protocol: "TLSv1.2"
    ssl.endpoint.identification.algorithm: HTTPS
  # ...
```

对于使用特定 密码套件 作为 TLS 版本进行客户端连接，您可以配置 allowed ssl 属性。您还可以 配置 ssl.endpoint.identification.algorithm 属性来 启用或禁用主机名验证。

13.2.60.2. logging

Kafka Connect（以及支持 Source2Image 的 Kafka Connect）具有自己的可配置日志记录器：

- connect.root.logger.level
- log4j.logger.org.reflections

根据运行的 Kafka Connect 插件，还会添加更多日志记录器。

使用 curl 请求获取从任何 Kafka 代理 pod 运行的 Kafka Connect loggers 的完整列表：

```
curl -s http://<connect-cluster-name>-connect-api:8083/admin/loggers/
```

Kafka Connect 使用 Apache log4j 日志记录器实施。

使用 logging 属性来配置日志记录器和日志记录器级别。

您可以通过直接（内线）指定日志记录器和级别来设置日志级别，或使用自定义（外部）ConfigMap。如果使用 ConfigMap，则将 logging.valueFrom.configMapKeyRef.name 属性设置为包含外部日志记录配置的 ConfigMap 的名称。在 ConfigMap 中，日志配置使用 log4j.properties 进行 描述。logging.valueFrom.configMapKeyRef.name 和 logging.valueFrom.configMapKeyRef.key 属性均是必需的。在运行 Cluster Operator 时，会使用自定义资源创建使用指定准确日志配置的 ConfigMap，然后在每次协调后重新创建。如果没有指定自定义 ConfigMap，则会使用默认日志设置。如果没有设置特定的日志记录器值，则会继承该日志记录器的上一级日志记录器设置。有关日志级别的更多信息，请参阅 Apache 日志记录服务。

此处我们会看到 内联 和外部 记录示例。

内联日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
spec:
  # ...
  logging:
    type: inline
    loggers:
      connect.root.logger.level: "INFO"
  # ...
```

外部日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
spec:
  # ...
  logging:
    type: external
    valueFrom:
      configMapKeyRef:
```

```
      name: customConfigMap
      key: connect-logging.log4j
  # ...
```

任何未配置可用的日志记录器将其级别设置为 OFF。

如果使用 Cluster Operator 部署 Kafka Connect，则会动态地更改到 Kafka Connect 日志记录级别。

如果使用外部日志记录，当日志附加程序被更改时会触发滚动更新。

垃圾收集器(GC)

也可以使用 jvmOptions 属性 来启用（或禁用）垃圾收集器日志记录。

13.2.60.3. KafkaConnectSpec schema 属性

属性	描述
version	Kafka Connect 版本。默认值为 2.8.0。请参阅用户文档以了解升级或降级版本所需的流程。
字符串	
replicas	Kafka Connect 组中的 pod 数量。
整数	
image	容器集的 docker 镜像。
字符串	
bootstrapServers	引导服务器以进行连接。这应该以逗号分隔的 *<hostname>:><port>* 对列表指定。
字符串	
tls	TLS 配置。
KafkaConnectTls	

属性	描述
身份验证 **KafkaClientAuthenticationTls**, **KafkaClientAuthenticationScramSha512**, **KafkaClientAuthenticationPlain**, **KafkaClientAuthenticationOAuth**	Kafka Connect 的身份验证配置.这个类型取决于给定对象中的 **authentication.type** 属性的值，必须是 [tls, scram-sha-512, plain, oauth] 中的一个。
config map	Kafka Connect 配置。无法设置带有以下前缀的属性：ssl.、sasl.、security.、listeners、plugin.path、re.、bootstrap.servers、consumer.interceptor.classes、producer.interceptor.classes（除：ssl.endpoint.identification.algorithm 除外）SSL.cipher.suites、ssl.protocol、ssl.enabled.protocols。
资源 ResourceRequirements	CPU 和内存资源以及请求的初始资源的最大限值。如需更多信息，请参阅核心/v1 资源要求的外部文档。
livenessProbe **probe**	Pod 存活度检查.
readinessprobe **probe**	Pod 就绪度检查。
jvmOptions **JvmOptions**	容器集的 JVM 选项.
jmxOptions **KafkaJmxOptions**	JMX 选项.
logging **InlineLogging**, **ExternalLogging**	Kafka Connect 的日志配置.类型取决于给定对象中的 **logging.type** 属性的值，它必须是 [inline, external] 之一。
tracing **JaegerTracing**	Kafka Connect 中的追踪配置。类型取决于给定对象中 **tracing.type** 属性的值，它必须是 [jaeger] 之一。
模板	Kafka Connect 和 Kafka Connect S2I 资源的模板。该模板允许用户指定如何生成 **Deployment**、**Pod** 和 **Service**。

属性	描述
KafkaConnectTemplate	
externalConfiguration **ExternalConfiguration**	将数据从 Secret 或 ConfigMap 传递给 Kafka Connect Pod，并使用它们来配置连接器。
build **build**	配置连接容器镜像的构建方式。可选。
clientRackInitImage 字符串	用于初始化 **client.rack** 的 init 容器的镜像。
metricsConfig **JmxPrometheusExporterMetrics**	指标配置.这个类型取决于给定对象中 **metricsConfig.type** 属性的值，必须是 [jmxPrometheusExporter] 中的一个。
rack **rack**	配置节点标签，该标签将用作 client.rack consumer 配置。

13.2.61. KafkaConnectTls 模式参考

用于：KafkaConnectS2ISpec、KafkaConnectSpec

KafkaConnectTls schema 属性的完整列表

配置 TLS 可信证书，以连接 Kafka 连接到集群。

13.2.61.1. trustedCertificates

使用 trustedCertificates 属性提供 secret 列表。

13.2.61.2. KafkaConnectTls schema 属性

属性	描述
trustedCertificates	TLS 连接的可信证书。
CertSecretSource 数组	

13.2.62. KafkaClientAuthenticationTls 模式参考

用来： KafkaBridgeSpec, KafkaConnectS2ISpec, KafkaConnectSpec, KafkaMirrorMaker2ClusterSpec, KafkaMirrorMakerConsumerSpec, KafkaMirrorMakerProducerSpec

KafkaClientAuthenticationTls schema 属性的完整列表

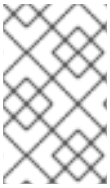
要配置 TLS 客户端身份验证，请将 type 属性设置为 value tls。TLS 客户端身份验证使用 TLS 证书进行身份验证。

13.2.62.1. certificateAndKey

证书在 certificateAndKey 属性中指定的，始终从 OpenShift 机密加载。在机密中，证书必须以 X509 格式存储在两个不同的密钥下：public 和 private。

您可以使用 User Operator 创建的 secret，或者您可以使用自己的 TLS 证书文件创建用于身份验证的密钥，然后从文件创建 Secret ：

```
oc create secret generic MY-SECRET \
--from-file=MY-PUBLIC-TLS-CERTIFICATE-FILE.crt \
--from-file=MY-PRIVATE.key
```



注意

TLS 客户端身份验证只能用于 TLS 连接。

TLS 客户端身份验证配置示例

```
authentication:
  type: tls
```

```
certificateAndKey:
  secretName: my-secret
  certificate: my-public-tls-certificate-file.crt
  key: private.key
```

13.2.62.2. KafkaClientAuthenticationTls schema 属性

type 属性是一个光盘，它区分使用 KafkaClientAuthenticationTls 类型与 KafkaClientAuthenticationScramSha512、KafkaClientAuthenticationPlain、KafkaClientAuthenticationOAuth。它必须具有类型 KafkaClientAuthenticationTls 的值 tls。

属性	描述
certificateAndKey	引用存放证书和私钥对的 **Secret**。
CertAndKeySecretSource	
type	必须是 **tls**。
字符串	

13.2.63. KafkaClientAuthenticationScramSha512 schema reference

用来：KafkaBridgeSpec, KafkaConnectS2ISpec, KafkaConnectSpec, KafkaMirrorMaker2ClusterSpec, KafkaMirrorMakerConsumerSpec, KafkaMirrorMakerProducerSpec

KafkaClientAuthenticationScramSha512 模式属性的完整列表

要配置基于 SASL 的 SCRAM-SHA-512 身份验证，请将 type 属性设置为 scr am-sha-512。SCRAM-SHA-512 验证机制需要用户名和密码。

13.2.63.1. username

指定 username 属性中的用户名。

13.2.63.2. passwordSecret

在 passwordSecret 属性中，指定一个指向包含密码的 Secret 的链接。

您可以使用 User Operator 创建的 secret。

如果需要，您可以创建一个包含明文密码的文本文件，用于验证：

```
echo -n PASSWORD > MY-PASSWORD.txt
```

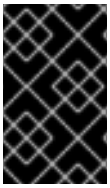
然后，您可以从文本文件创建 Secret，为密码设置您自己的字段名称(key)：

```
oc create secret generic MY-CONNECT-SECRET-NAME --from-file=MY-PASSWORD-FIELD-NAME=./MY-PASSWORD.txt
```

Kafka Connect 的 SCRAM-SHA-512 客户端身份验证的 Secret 示例

```
apiVersion: v1
kind: Secret
metadata:
  name: my-connect-secret-name
type: Opaque
data:
  my-connect-password-field: LFTIyFRFIMmU2N2Tm
```

secretName 属性包含 Secret 的名称，password 属性 包含密码存储在 Secret 中的密钥名称。



重要

不要在 password 属性中指定实际密码。

Kafka Connect 基于 SASL 的 SCRAM-SHA-512 客户端身份验证配置示例

```
authentication:
```

```
type: scram-sha-512
username: my-connect-username
passwordSecret:
  secretName: my-connect-secret-name
  password: my-connect-password-field
```

13.2.63.3. KafkaClientAuthenticationScramSha512 schema properties

type 属性是一个光盘，它区分使用 KafkaClientAuthenticationScramSha512 类型与 KafkaClientAuthenticationTls、KafkaClientAuthenticationPlain、KafkaClientAuthenticationOAuth。它必须具有类型 KafkaClientAuthenticationScramSha512 的值 scr am-sha-512。

属性	描述
passwordSecret	引用存放密码的 **Secret**。
PasswordSecretSource	
type	必须是 scr **am-sha-512**。
字符串	
username	用于身份验证的用户名。
字符串	

13.2.64. PasswordSecretSource 模式参考

用于：KafkaClientAuthenticationPlain、KafkaClientAuthenticationScramSha512

属性	描述
password	保存密码的 Secret 中的密钥名称。
字符串	
secretName	包含密码的 Secret 名称。
字符串	

13.2.65. KafkaClientAuthenticationPlain 模式参考

用来： KafkaBridgeSpec, KafkaConnectS2ISpec, KafkaConnectSpec, KafkaMirrorMaker2ClusterSpec, KafkaMirrorMakerConsumerSpec, KafkaMirrorMakerProducerSpec

KafkaClientAuthenticationPlain 模式属性的完整列表

要配置基于 SASL 的 PLAIN 身份验证，请将 type 属性设置为 纯文本。SASL PLAIN 身份验证机制需要用户名和密码。

警告

SASL PLAIN 机制将以明文方式在网络上传输用户名和密码。仅当启用了 TLS 加密时，才使用 SASL PLAIN 身份验证。

13.2.65.1. username

指定 username 属性中的用户名。

13.2.65.2. passwordSecret

在 passwordSecret 属性中，指定一个指向包含密码的 Secret 的链接。

您可以使用 User Operator 创建的 secret。

如果需要，以明文形式创建一个包含密码的文本文件，以用于验证：

```
echo -n PASSWORD > MY-PASSWORD.txt
```

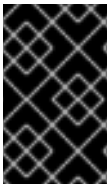
然后，您可以从文本文件创建 Secret，为密码设置您自己的字段名称(key)：

```
oc create secret generic MY-CONNECT-SECRET-NAME --from-file=MY-PASSWORD-FIELD-NAME=./MY-PASSWORD.txt
```

Kafka Connect PLAIN 客户端身份验证的 Secret 示例

```
apiVersion: v1
kind: Secret
metadata:
  name: my-connect-secret-name
type: Opaque
data:
  my-password-field-name: LFTIyFRFlMmU2N2Tm
```

secretName 属性包含 Secret 的名称，password 属性 包含密码存储在 机密 中的密钥名称。



重要

不要在 password 属性中指定实际密码。

基于 SASL 的 PLAIN 客户端身份验证配置示例

```
authentication:
  type: plain
  username: my-connect-username
  passwordSecret:
    secretName: my-connect-secret-name
    password: my-password-field-name
```

13.2.65.3. KafkaClientAuthenticationPlain 架构属性

type 属性是一个光盘，它区分使用 KafkaClientAuthenticationPlain 类型与 KafkaClientAuthenticationTls、KafkaClientAuthenticationScramSha512、KafkaClientAuthenticationOAuth。它对于类型 KafkaClientAuthenticationPlain 的值必须是 明文 的。

属性	描述
passwordSecret	引用存放密码的 **Secret**。
PasswordSecretSource	
type	必须是 **普通的**。
字符串	
username	用于身份验证的用户名。
字符串	

13.2.66. KafkaClientAuthenticationOAuth 模式参考

用来： KafkaBridgeSpec, KafkaConnectS2ISpec, KafkaConnectSpec, KafkaMirrorMaker2ClusterSpec, KafkaMirrorMakerConsumerSpec, KafkaMirrorMakerProducerSpec

KafkaClientAuthenticationOAuth 模式属性的完整列表

要配置 OAuth 客户端身份验证，请将 type 属性设置为 oauth。

OAuth 身份验证可使用以下选项之一进行配置：

- *客户端 ID 和 secret*
- *客户端 ID 和刷新令牌*
- *访问令牌*
- *TLS*

客户端 ID 和 secret

您可以在 tokenEndpointUri 属性中配置授权服务器的地址，以及身份验证中使用的客户端 ID 和客户端机密。OAuth 客户端将连接到 OAuth 服务器，使用客户端 ID 和机密进行身份验证，并获取它将用于与 Kafka 代理进行身份验证的访问令牌。在 clientSecret 属性中，指定一个指向包含客户端 secret 的 Secret 的链接。

使用客户端 ID 和客户端 secret 进行 OAuth 客户端身份验证示例

```
authentication:
  type: oauth
  tokenEndpointUri: https://sso.myproject.svc:8443/auth/realms/internal/protocol/openid-connect/token
  clientId: my-client-id
  clientSecret:
    secretName: my-client-oauth-secret
    key: client-secret
```

（可选）在需要时可以指定 范围和 受众。

客户端 ID 和刷新令牌

您可以在 tokenEndpointUri 属性中配置 OAuth 服务器的地址，并与 OAuth 客户端 ID 和刷新令牌一起配置。OAuth 客户端将连接到 OAuth 服务器，使用客户端 ID 进行身份验证并刷新令牌，并获取它将用于与 Kafka 代理进行身份验证的访问令牌。在 refreshToken 属性中，指定包含刷新令牌的 Secret 链接。

+ .使用客户端 ID 和刷新令牌进行 OAuth 客户端身份验证示例

```
authentication:
  type: oauth
  tokenEndpointUri: https://sso.myproject.svc:8443/auth/realms/internal/protocol/openid-connect/token
  clientId: my-client-id
  refreshToken:
    secretName: my-refresh-token-secret
    key: refresh-token
```

访问令牌

您可以直接配置用于与 Kafka 代理进行身份验证的访问令牌。在本例中，您不用指定 tokenEndpointUri。在 accessToken 属性中，指定包含访问令牌的 Secret 链接。

仅使用访问令牌进行 OAuth 客户端身份验证示例

```
authentication:
  type: oauth
  accessToken:
    secretName: my-access-token-secret
    key: access-token
```

TLS

使用 HTTPS 协议访问 OAuth 服务器不需要额外的配置，只要其使用的 TLS 证书由可信证书颁发机构签名，并且其主机名列在证书中。

如果您的 OAuth 服务器使用自签名证书，或者由不信任的证书颁发机构签名，您可以在自定义资源中配置可信证书列表。The tlsTrustedCertificates 属性包含一个 secret 列表，其密钥名称存储在证书的下面。证书必须以 X509 格式存储。

提供的 TLS 证书示例

```
authentication:
  type: oauth
  tokenEndpointUri: https://sso.myproject.svc:8443/auth/realms/internal/protocol/openid-
connect/token
  clientId: my-client-id
  refreshToken:
    secretName: my-refresh-token-secret
    key: refresh-token
  tlsTrustedCertificates:
    - secretName: oauth-server-ca
      certificate: tls.crt
```

OAuth 客户端将默认验证 OAuth 服务器的主机名是否与证书主题或其它 DNS 名称匹配。如果不需要，您可以禁用主机名验证。

禁用 TLS 主机名验证示例

```
authentication:
  type: oauth
  tokenEndpointUri: https://sso.myproject.svc:8443/auth/realms/internal/protocol/openid-connect/token
  clientId: my-client-id
  refreshToken:
    secretName: my-refresh-token-secret
    key: refresh-token
  disableTlsHostnameVerification: true
```

13.2.66.1. KafkaClientAuthenticationOAuth 模式属性

type 属性是一个光盘，它区分使用 KafkaClientAuthenticationOAuth 类型与 KafkaClientAuthenticationTls、KafkaClientAuthenticationScramSha512、KafkaClientAuthenticationPlain。它必须具有类型 KafkaClientAuthenticationOAuth 的值 **oauth**。

属性	描述
accessToken GenericSecretSource	包含从授权服务器获取的访问令牌的 OpenShift Secret 链接。
accessTokenIsJwt 布尔值	配置是否应将访问令牌视为 JWT。如果授权服务器返回不透明令牌，这应当设为 **false**。默认值为 **true**。
受众 字符串	针对授权服务器进行身份验证时使用的 OAuth 使用者。某些授权服务器需要明确设置受众。可能的值取决于授权服务器的配置方式。默认情况下，**在执行** 令牌端点请求时不指定使用者。
clientId 字符串	Kafka 客户端 ID，可用于对 OAuth 服务器进行身份验证并使用令牌端点 URI。
clientSecret GenericSecretSource	含有 OAuth 客户端机密的 OpenShift Secret 链接，Kafka 客户端可以使用该机密对 OAuth 服务器进行身份验证并使用令牌端点 URI。
disableTlsHostnameVerification	启用或禁用 TLS 主机名验证。默认值为 **false**。

属性	描述
布尔值	
maxTokenExpirySeconds 整数	将访问令牌的寿命限制为指定的秒数。如果授权服务器返回不透明令牌，则应设置此项。
refreshToken **GenericSecretSource**	包含刷新令牌的 OpenShift Secret 链接，可用于从授权服务器获取访问令牌。
scope 字符串	针对授权服务器进行身份验证时使用的 OAuth 范围。些授权服务器要求进行此设置。可能的值取决于授权服务器的配置方式。默认情况下，执行令牌端点请求时不指定 **范围**。
tlsTrustedCertificates **CertSecretSource** 数组	TLS 连接到 OAuth 服务器的可信证书。
tokenEndpointUri 字符串	授权服务器令牌端点 URI.
type 字符串	必须是 **oauth**。

13.2.67. Jaegertracing 模式参考

用来： ***KafkaBridgeSpec, KafkaConnectS2ISpec, KafkaConnectSpec, KafkaMirrorMaker2Spec, KafkaMirrorMakerSpec***

type 属性是一个分级器，它区分 JaegerTracing 类型的使用以及将来可能添加的其他子类型。它必须具有类型 JaegerTracing 的 jaeger 值。

属性	描述
type 字符串	必须是 **Jaeger**。

13.2.68. KafkaConnectTemplate 模式参考

用于： *KafkaConnectS2ISpec*、*KafkaConnectSpec*、*KafkaMirrorMaker2Spec*

属性	描述
Deployment **DeploymentTemplate**	Kafka Connect **部署** 模板.
Pod **PodTemplate**	Kafka Connect **Pod** 模板.
apiService **InternalServiceTemplate**	Kafka Connect API **Service** 模板.
connectContainer **ContainerTemplate**	Kafka Connect 容器的模板。
initContainer **ContainerTemplate**	Kafka init 容器的模板。
podDisruptionBudget **PodDisruptionBudgetTemplate**	Kafka Connect **PodDisruptionBudget** 模板。
serviceAccount **ResourceTemplate**	Kafka Connect 服务帐户的模板。
clusterRoleBinding **ResourceTemplate**	Kafka Connect ClusterRoleBinding 模板。
buildPod **PodTemplate**	Kafka Connect 构建 **Pod** 模板.构建容器集仅在 OpenShift 中使用。
buildContainer **ContainerTemplate**	Kafka Connect Build 容器的模板。构建容器仅在 OpenShift 中使用。

属性	描述
buildConfig	用于构建新容器镜像的 Kafka Connect BuildConfig 模板。BuildConfig 仅在 OpenShift 中使用。
ResourceTemplate	
buildServiceAccount	Kafka Connect Build 服务帐户的模板。
ResourceTemplate	

13.2.69. DeploymentTemplate 架构参考

用于：KafkaBridgeTemplate、KafkaConnectTemplate、KafkaMirrorMakerTemplate

属性	描述
metadata	应用到资源的元数据。
MetadataTemplate	
deploymentStrategy	用于此 Deployment 的 DeploymentStrategy。有效值为 **RollingUpdate** 和 **Recreate**。默认值为 **RollingUpdate**。
字符串（[RollingUpdate、Recreate] 中的一个）	

13.2.70. ExternalConfiguration 架构参考

用于：KafkaConnectS2ISpec、KafkaConnectSpec、KafkaMirrorMaker2Spec

ExternalConfiguration 架构属性的完整列表

配置外部存储属性，为 Kafka Connect 连接器定义配置选项。

您可以将 ConfigMap 或 Secret 挂载到 Kafka Connect pod 中，作为环境变量或卷。卷和环境变量在 KafkaConnect .spec 和 KafkaConnectS2I.spec 中的 externalConfiguration 属性中配置。

应用后，可在开发连接器时使用环境变量和卷。

13.2.70.1. env

使用 env 属性指定一个或多个环境变量。这些变量可以包含 ConfigMap 或 Secret 中的值。

包含环境变量值的 Secret 示例

```
apiVersion: v1
kind: Secret
metadata:
  name: aws-creds
type: Opaque
data:
  awsAccessKey: QUtJQVhYWFhYWFhYWFhYWFg=
  awsSecretAccessKey: Ylhsd1IYTnpkMjl5WkE=
```

注意

用户定义的环境变量的名称不能以 KAFKA_ 或 STRIMZI_ 开头。

要将值从 Secret 挂载到环境变量，请使用 valueFrom 属性和 secretKeyRef。

设置为 Secret 中值的示例环境变量

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  externalConfiguration:
    env:
      - name: AWS_ACCESS_KEY_ID
        valueFrom:
          secretKeyRef:
            name: aws-creds
            key: awsAccessKey
      - name: AWS_SECRET_ACCESS_KEY
        valueFrom:
```

```
        secretKeyRef:
          name: aws-creds
          key: awsSecretAccessKey
```

挂载 Secret 的常见用例是供连接器与 Amazon AWS 通信。连接器需要能够读取 AWS_ACCESS_KEY_ID 和 AWS_SECRET_ACCESS_KEY。

要将 ConfigMap 的值挂载到环境变量，请使用 valueFrom 属性中的 configMapKeyRef，如下例所示。

设置为 ConfigMap 中值的环境变量示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  externalConfiguration:
    env:
      - name: MY_ENVIRONMENT_VARIABLE
        valueFrom:
          configMapKeyRef:
            name: my-config-map
            key: my-key
```

13.2.70.2. 卷

使用卷将 ConfigMap 或 Secret 挂载到 Kafka Connect pod。

在以下情况下，使用卷而不是环境变量会很有用：

- 挂载用来配置 Kafka Connect 连接器的属性文件

- 使用 TLS 证书挂载信任存储或密钥存储

卷挂载到 Kafka Connect 容器中，路径 /opt/kafka/external-configuration/<volume-name>。例如，名为 connector -config 的卷中的文件将显示在 /opt/kafka/external-configuration/connector-config 目录中。

配置 提供程序 从配置外部加载值。使用供应商机制避免通过 Kafka Connect REST 接口传递受限信息。

- FileConfigProvider 从文件中的属性加载配置值。
- DirectoryConfigProvider 从目录结构中的独立文件加载配置值。

如果要添加多个供应商（包括自定义供应商），请使用以逗号分隔的列表。您可以使用自定义供应商从其他文件位置加载值。

使用 FileConfigProvider 加载属性值

在本例中，名为 mysecret 的 Secret 包含指定数据库名称和密码的连接器属性：

带有数据库属性的 Secret 示例

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
stringData:
  connector.properties: |- 1
    dbUsername: my-username 2
    dbPassword: my-password
```

1

属性文件格式的连接器配置。

2

配置中使用的数据库用户名和密码属性。

Secret 和 FileConfigProvider 配置供应商在 Kafka Connect 配置中指定。

- Secret 被挂载到名为 connector -config 的卷。
- 为 fileConfigProvider 指定别名 文件。

外部卷示例设置为来自 Secret 的值

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    config.providers: file 1
    config.providers.file.class: org.apache.kafka.common.config.provider.FileConfigProvider
2
  #...
  externalConfiguration:
    volumes:
      - name: connector-config 3
        secret:
          secretName: mysecret 4
```

1

配置提供程序的别名用于定义其他配置参数。

2

FileConfigProvider 提供属性文件的值。参数使用 config.providers 中的别名，格式为 config.providers.${alias}.class。

3

4

Secret 的名称。

在连接器配置中引用 Secret 中属性值的占位符。占位符结构为 file:PATH-AND-FILE-NAME:PROPERTY。FileConfigProvider 在连接器配置中读取并提取挂载的 Secret 中的数据库 用户名和密码 属性值。

显示外部值占位符的连接器配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-source-connector
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
  class: io.debezium.connector.mysql.MySqlConnector
  tasksMax: 2
  config:
    database.hostname: 192.168.99.1
    database.port: "3306"
    database.user: "${file:/opt/kafka/external-configuration/connector-config/mysecret:dbUsername}"
    database.password: "${file:/opt/kafka/external-configuration/connector-config/mysecret:dbPassword}"
    database.server.id: "184054"
    #...
```

使用 DirectoryConfigProvider 从单独的文件中加载属性值

在本例中，Secret 在单独的文件中包含 TLS 信任存储和密钥存储用户凭证。

带有用户凭证的 Secret 示例

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
```

```
  labels:
    strimzi.io/kind: KafkaUser
    strimzi.io/cluster: my-cluster
type: Opaque
data: 1
  ca.crt: # Public key of the client CA
  user.crt: # User certificate that contains the public key of the user
  user.key: # Private key of the user
  user.p12: # PKCS #12 archive file for storing certificates and keys
  user.password: # Password for protecting the PKCS #12 archive file
```

Secret 和 DirectoryConfigProvider 配置供应商在 Kafka Connect 配置中指定。

- *Secret 被挂载到名为 connector -config 的卷。*
- *DirectoryConfigProvider 被授予别名 目录。*

为用户凭证文件设置的外部卷示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    config.providers: directory
    config.providers.directory.class:
org.apache.kafka.common.config.provider.DirectoryConfigProvider 1
  #...
  externalConfiguration:
    volumes:
      - name: connector-config
        secret:
          secretName: mysecret
```

1 1

凭据的占位符在连接器配置中引用。占位符结构为 directory:PATH:FILE-NAME。DirectoryConfigProvider 在连接器配置中读取并提取挂载的 Secret 中的凭证。

显示外部值占位符的连接器配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-source-connector
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
  class: io.debezium.connector.mysql.MySqlConnector
  tasksMax: 2
  config:
    security.protocol: SSL
    ssl.truststore.type: PEM
    ssl.truststore.location: "${directory:/opt/kafka/external-configuration/connector-config:ca.crt}"
    ssl.keystore.type: PEM
    ssl.keystore.location: ${directory:/opt/kafka/external-configuration/connector-config:user.key}"
    #...
```

13.2.70.3. ExternalConfiguration 架构属性

属性	描述
env	在 Kafka Connect Pod 中提供来自 Secret 或 ConfigMap 的数据作为环境变量。
ExternalConfigurationEnv 数组	
卷	在 Kafka Connect Pod 中提供来自 Secret 或 ConfigMap 的数据作为卷。
ExternalConfigurationVolumeSource 数组	

13.2.71. ExternalConfigurationEnv 模式参考

用于： ExternalConfiguration

属性	描述
name 字符串	传递给 Kafka Connect Pod 的环境变量的名称。环境变量的名称不能以 **KAFKA_ 或 STRIMZI_** 开头。
valueFrom **ExternalConfigurationEnvVarSource**	传递给 Kafka Connect Pod 的环境变量的值。它可以作为对 Secret 或 ConfigMap 字段的引用来传递。该字段必须指定一个 Secret 或 ConfigMap。

13.2.72. ExternalConfigurationEnvVarSource 模式参考

用于： ***ExternalConfigurationEnv***

属性	描述
configMapKeyRef ConfigMapKeySelector	引用 ConfigMap 中的键。如需更多信息，请参阅 core/v1 configmapkeyselector 的外部文档。
secretKeyRef SecretKeySelector	在机密中引用密钥.如需更多信息，请参阅 core/v1 secretkeyselector 的外部文档。

13.2.73. ExternalConfigurationVolumeSource 模式引用

用于： ***ExternalConfiguration***

属性	描述
configMap ConfigMapVolumeSource	引用 ConfigMap 中的键。只需要指定一个 Secret 或 ConfigMap。如需更多信息，请参阅 core/v1 configmapvolumesource 的外部文档。
name 字符串	添加到 Kafka Connect Pod 的卷名称。
Secret SecretVolumeSource	在机密中引用密钥.只需要指定一个 Secret 或 ConfigMap。如需更多信息，请参阅 core/v1 secretvolumesource 的外部文档。

13.2.74. 构建 架构参考

用于： KafkaConnectS2ISpec、 KafkaConnectSpec

构建 架构属性的完整列表

为 Kafka Connect 部署配置额外的连接器。

13.2.74.1. output

要使用额外的连接器插件构建新容器镜像，AMQ Streams 需要一个容器 registry，从中可以推送、存储和从中拉取镜像。AMQ Streams 不运行自己的容器 registry，因此必须提供 registry。AMQ Streams 支持私有容器 registry，以及 Quay 或 Docker Hub 等公共 registry。容器 registry 在 KafkaConnect 自定义资源的 .spec.build.output 部分中进行配置。输出 配置支持两种类型：docker 和 imagestream。

使用 Docker registry

要使用 Docker 注册表，您必须将 type 指定为 docker，并使用新容器镜像的全名指定 image 字段。全名必须包括：

- registry 的地址
- 端口号（如果侦听非标准端口）
- 新容器镜像的标签

有效容器镜像名称示例：

- docker.io/my-org/my-image/my-tag
- quay.io/my-org/my-image/my-tag

- *image-registry.image-registry.svc:5000/myproject/kafka-connect-build:latest*

每个 Kafka Connect 部署都必须使用单独的镜像，该镜像可以在最基本的级别表示不同的标签。

如果 registry 需要身份验证，请使用 pushSecret 使用 registry 凭证设置 Secret 的名称。对于 Secret，使用 kubernetes.io/dockerconfigjson 类型和 a .dockerconfigjson 文件来包含 Docker 凭证。有关从私有 registry 中拉取镜像的更多信息，请参阅 基于现有 Docker 凭证创建 Secret。

输出配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      type: docker 1
      image: my-registry.io/my-org/my-connect-cluster:latest 2
      pushSecret: my-registry-credentials 3
  #...
```

1 *（必需）AMQ Streams 使用的输出类型。*

2 *（必需）所用镜像的全名，包括存储库和标签。*

3 *（可选）带有容器注册表凭据的 secret 名称。*

使用 OpenShift ImageStream

您可以使用 OpenShift ImageStream 来存储新的容器镜像，而不是 Docker。在部署 Kafka Connect 前，必须手动创建 ImageStream。要使用 ImageStream，请将 type 设置为 imagestream，

并使用 image 属性指定 ImageStream 的名称以及使用的标签。例如，my -connect-image-stream:latest。

输出配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      type: imagestream 1
      image: my-connect-build:latest 2
  #...
```

1

（必需）AMQ Streams 使用的输出类型。

2

（必需）ImageStream 和标签的 Name。

13.2.74.2. plugins

连接器插件是一组文件，用于定义连接特定类型外部系统所需的实施。容器镜像所需的连接器插件必须使用 KafkaConnect 自定义资源的 .spec.build.plugins 属性进行配置。每个连接器插件都必须具有一个在 Kafka Connect 部署中唯一的名称。此外，还必须列出插件工件。这些工件由 AMQ Streams 下载，添加到新容器镜像中，并在 Kafka Connect 部署中使用。连接器插件工件也可以包含其他组件，如 (de)serializers。每个连接器插件都会下载到单独的目录中，以便正确 沙盒 不同的连接器及其依赖项。每个插件必须至少配置一个 工件。

有两个连接器 插件的插件 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
```

```
spec:
  #...
  build:
    output:
      #...
    plugins: 1
      - name: debezium-postgres-connector
        artifacts:
          - type: tgz
            url: https://repo1.maven.org/maven2/io/debezium/debezium-connector-
postgres/1.3.1.Final/debezium-connector-postgres-1.3.1.Final-plugin.tar.gz
            sha512sum:
962a12151bdf9a5a30627eebac739955a4fd95a08d373b86bdcea2b4d0c27dd6e1edd5cb548045e1
15e33a9e69b1b2a352bee24df035a0447cb820077af00c03
      - name: camel-telegram
        artifacts:
          - type: tgz
            url: https://repo.maven.apache.org/maven2/org/apache/camel/kafkaconnector/camel-
telegram-kafka-connector/0.7.0/camel-telegram-kafka-connector-0.7.0-package.tar.gz
            sha512sum:
a9b1ac63e3284bea7836d7d24d84208c49cdf5600070e6bd1535de654f6920b74ad950d51733e802
0bf4187870699819f54ef5859c7846ee4081507f48873479
  #...
```

1

（必需）连接器插件及其工件的列表。

AMQ Streams 支持以下类型的工件： JAR 文件，被直接下载和使用 * TGZ 存档，这些存档会被下载和解包 * 其他工件（可直接下载和使用）

重要

AMQ Streams 不对下载的工件进行任何安全扫描。出于安全考虑，您应首先手动验证工件，并配置 checksum 验证，以确保自动构建和在 Kafka Connect 部署中使用相同的工件。

使用 JAR 工件

JAR 工件表示 JAR 文件，它被下载并添加到容器镜像中。要使用 JAR 工件，请将 type 属性设置为 jar，再使用 url 属性指定下载位置。

另外，您可以指定工件的 SHA-512 校验和。如果指定，AMQ Streams 将在构建新容器镜像时验证工

件的校验和。

JAR 工件示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins:
      - name: my-plugin
        artifacts:
          - type: jar (1)
            url: https://my-domain.tld/my-jar.jar (2)
            sha512sum: 589...ab4 (3)
          - type: jar
            url: https://my-domain.tld/my-jar2.jar
  #...
```

(1) （必需）工件类型.

(2) （必需）从中下载工件的 URL。

(3) （可选）SHA-512 checksum 以验证工件。

使用 TGZ 工件

TGZ 工件用于下载使用 Gzip 压缩压缩的 TAR 存档。TGZ 构件可以包含整个 Kafka Connect 连接器，即使包含多个不同的文件。构建新容器镜像时，AMQ Streams 会自动下载并解压缩 TGZ 构件。要使用 TGZ 工件，请将 type 属性设置为 tgz，并使用 url 属性指定下载位置。

另外，您可以指定工件的 SHA-512 校验和。如果指定，AMQ Streams 将先验证校验和，然后再解压缩它并构建新的容器镜像。

TGZ 构件示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins:
      - name: my-plugin
        artifacts:
          - type: tgz (1)
            url: https://my-domain.tld/my-connector-archive.jar (2)
            sha512sum: 158...jg10 (3)
  #...
```

(1) （必需）工件类型.

(2) 从中下载存档的（必需）URL。

(3) （可选）SHA-512 checksum 以验证工件。

使用 其他 工件

其他 工件表示要下载并添加到容器镜像的任何类型的文件。如果要在生成的容器镜像中使用特定名称作为工件，请使用 fileName 字段。如果未指定文件名，则基于 URL 哈希命名该文件。

另外，您可以指定工件的 SHA-512 校验和。如果指定，AMQ Streams 将在构建新容器镜像时验证工件的校验和。

其他 工件示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins:
      - name: my-plugin
        artifacts:
          - type: other (1)
            url: https://my-domain.tld/my-other-file.ext (2)
            sha512sum: 589...ab4 (3)
            fileName: name-the-file.ext (4)
  #...
```

(1) *（必需）工件类型.*

(2) *（必需）从中下载工件的 URL。*

(3) *（可选）SHA-512 checksum 以验证工件。*

(4) *（可选）文件存储在生成的容器镜像中的名称。*

13.2.74.3. 构建 架构属性

属性	描述
output **dockerOutput**, **ImageStreamOutput**	配置新构建映像的存储位置。必需。类型取决于给定对象中的 **output.type** 属性的值，它必须是 [docker, imagestream] 之一。
资源 ResourceRequirements	为构建保留的 CPU 和内存资源。如需更多信息，请参阅核心/v1 资源要求的外部文档。
plugins **插件** 数组	应添加到 Kafka Connect 中的连接器插件列表。必需。

13.2.75. dockerOutput 模式参考

使用的：** **Build

type 属性是一个分级器，它区分 DockerOutput 类型与 ImageStreamOutput 的使用。它必须具有类型 DockerOutput 的值 docker。

属性	描述
image 字符串	用于标记和推送新构建镜像的完整名称。例如 **quay.io/my-organization/my-custom-connect:latest**。必需。
pushSecret 字符串	带有推送新构建镜像的凭据的容器注册表 Secret。
additionalKanikoOptions 字符串数组	配置在构建新 Connect 镜像时将传递给 Kaniko executor 的附加选项。允许的选项有：--customPlatform, --insecure-pull, --insecure-registry, --log-format, --log-timestamp, --registry-mirror, --reproducible, --single-snapshot, --skip-tls-verify, --skip-tls-verify-pull, --skip-tls-verify-registry, --verbosity, --snapshotMode, --use-new-run.这些选项将仅在使用 Kaniko executor 的 OpenShift 中使用。OpenShift 中将忽略它们。Kaniko GitHub 仓库 中描述了这些选项。更改此字段不会触发 Kafka Connect 镜像的新构建。
type	必须是 **docker**。

属性	描述
字符串	

13.2.76. ImageStreamOutput 模式参考

使用的： Build

type 属性是一个分级器，它区分使用 ImageStreamOutput 和 DockerOutput。它必须具有类型 ImageStreamOutput 的值 镜像流。

属性	描述
image	推送新构建镜像的 ImageStream 的名称和标签。例如 **my-custom-connect:latest**。必需。
字符串	
type	必须是 **镜像流**。
字符串	

13.2.77. 插件 架构参考

使用的： Build

属性	描述
name	连接器插件的唯一名称。将用于生成存储连接器工件的路径。该名称必须在 KafkaConnect 资源中唯一。名称必须遵循以下模式： **^[a-z][-_a-z0-9]*[a-z]$**.必需。
字符串	
工件	属于此连接器插件的工件列表。必需。
JarArtifact、**TgzArtifact**、**ZipArtifact**、**OtherArtifact** 数组	

13.2.78. JarArtifact 架构参考

用于： 插件

属性	描述
url 字符串	要下载的工件的 URL。AMQ Streams 不会对下载的工件进行任何安全扫描。出于安全考虑，您应首先手动验证工件并配置 checksum 验证，以确保自动构建中使用同样的工件。必需。
sha512sum 字符串	工件的 SHA512 校验和.可选。如果指定，则构建新容器时将验证校验和。如果没有指定，则不会验证下载的工件。
type 字符串	必须是 **jar**。

13.2.79. TgzArtifact 架构参考

用于： ***插件***

属性	描述
url 字符串	要下载的工件的 URL。AMQ Streams 不会对下载的工件进行任何安全扫描。出于安全考虑，您应首先手动验证工件并配置 checksum 验证，以确保自动构建中使用同样的工件。必需。
sha512sum 字符串	工件的 SHA512 校验和.可选。如果指定，则构建新容器时将验证校验和。如果没有指定，则不会验证下载的工件。
type 字符串	必须为 **tgz**。

13.2.80. ZipArtifact 架构参考

用于： ***插件***

属性	描述

属性	描述
url 字符串	要下载的工件的 URL。AMQ Streams 不会对下载的工件进行任何安全扫描。出于安全考虑，您应首先手动验证工件并配置 checksum 验证，以确保自动构建中使用同样的工件。必需。
sha512sum 字符串	工件的 SHA512 校验和.可选。如果指定，则构建新容器时将验证校验和。如果没有指定，则不会验证下载的工件。
type 字符串	必须是 **zip**。

13.2.81. OtherArtifact 架构参考

用于： ***插件***

属性	描述
url 字符串	要下载的工件的 URL。AMQ Streams 不会对下载的工件进行任何安全扫描。出于安全考虑，您应首先手动验证工件并配置 checksum 验证，以确保自动构建中使用同样的工件。必需。
sha512sum 字符串	工件的 SHA512 校验和.可选。如果指定，则构建新容器时将验证校验和。如果没有指定，则不会验证下载的工件。
fileName 字符串	工件存储在下的名称。
type 字符串	必须是 **其他**.

13.2.82. KafkaConnectStatus 模式参考

用于： ***KafkaConnect***

属性	描述
conditions	状态条件列表。
条件 数组	
observedGeneration	生成 Operator 最后协调的 CRD。
整数	
url	用于管理和监控 Kafka Connect 连接器的 REST API 端点 URL。
字符串	
connectorPlugins	此 Kafka Connect 部署中可用的连接器插件列表。
ConnectorPlugin 数组	
labelSelector	提供此资源的 pod 的标签选择器。
字符串	
replicas	用于提供此资源的当前容器集数量。
整数	

13.2.83. ConnectorPlugin 架构参考

用于： ***KafkaConnectS2IStatus***、***KafkaConnectStatus***、***KafkaMirrorMaker2Status***

属性	描述
type	连接器插件的类型。可用的类型是 **sink** 和 **source**。
字符串	
version	连接器插件的版本。
字符串	
class	连接器插件的类。
字符串	

13.2.84. KafkaConnectS2I 模式参考

KafkaConnectS2I 类型已弃用。请改为使用 Build。

属性	描述
spec	Kafka Connect Source-to-Image(S2I)集群的规格。
KafkaConnectS2ISpec	
status	Kafka Connect Source-to-Image(S2I)集群的状态。
KafkaConnectS2IStatus	

13.2.85. KafkaConnectS2ISpec 模式参考

用于：KafkaConnectS2I

KafkaConnectS2ISpec schema 属性的完整列表

配置支持 Source-to-Image(S2I)支持的 Kafka Connect 集群。

在 OpenShift 上（仅限使用连接器插件）扩展 Kafka Connect 时，您可以使用 OpenShift 构建和 S2I 创建供 Kafka Connect 部署使用的容器镜像。

配置选项使用 Kafka ConnectSpec 模式与 Kafka Connect 配置类似。

13.2.85.1. KafkaConnectS2ISpec schema 属性

属性	描述
version	Kafka Connect 版本。默认值为 2.8.0。请参阅用户文档以了解升级或降级版本所需的流程。
字符串	
replicas	Kafka Connect 组中的 pod 数量。
整数	

属性	描述
image 字符串	容器集的 docker 镜像。
buildResources ResourceRequirements	要保留的 CPU 和内存资源。如需更多信息，请参阅核心/v1 资源要求的外部文档。
bootstrapServers 字符串	引导服务器以进行连接。这应该以逗号分隔的 *<hostname>: ><port>* 对列表指定。
tls **KafkaConnectTls**	TLS 配置。
身份验证 **KafkaClientAuthenticationTls**, **KafkaClientAuthenticationScramSha512**, **KafkaClientAuthenticationPlain**, **KafkaClientAuthenticationOAuth**	Kafka Connect 的身份验证配置.这个类型取决于给定对象中的 **authentication.type** 属性的值，必须是 [tls, scram-sha-512, plain, oauth] 中的一个。
config map	Kafka Connect 配置。无法设置带有以下前缀的属性：ssl.、sasl.、security.、listeners、plugin.path、re.、bootstrap.servers、consumer.interceptor.classes、producer.interceptor.classes（除：ssl.endpoint.identification.algorithm 除外）SSL.cipher.suites、ssl.protocol、ssl.enabled.protocols。
资源 ResourceRequirements	CPU 和内存资源以及请求的初始资源的最大限值。如需更多信息，请参阅核心/v1 资源要求的外部文档。
livenessProbe **probe**	Pod 存活度检查.
readinessprobe **probe**	Pod 就绪度检查。
jvmOptions	容器集的 JVM 选项.

属性	描述
JvmOptions	
jmxOptions	JMX 选项.
KafkaJmxOptions	
logging	Kafka Connect 的日志配置.类型取决于给定对象中的 **logging.type** 属性的值，它必须是 [inline, external] 之一。
InlineLogging, **ExternalLogging**	
tracing	Kafka Connect 中的追踪配置。类型取决于给定对象中 **tracing.type** 属性的值，它必须是 [jaeger] 之一。
JaegerTracing	
模板	Kafka Connect 和 Kafka Connect S2I 资源的模板。该模板允许用户指定如何生成 **Deployment**、**Pod** 和 **Service**。
KafkaConnectTemplate	
externalConfiguration	将数据从 Secret 或 ConfigMap 传递给 Kafka Connect Pod，并使用它们来配置连接器。
ExternalConfiguration	
build	配置连接容器镜像的构建方式。可选。
build	
clientRackInitImage	用于初始化 **client.rack** 的 init 容器的镜像。
字符串	
insecureSourceRepository	当为 true 时，这会使用 'Local' 引用策略和接受不安全源标签的导入策略来配置源存储库。
布尔值	
metricsConfig	指标配置.这个类型取决于给定对象中 **metricsConfig.type** 属性的值，必须是 [jmxPrometheusExporter] 中的一个。
JmxPrometheusExporterMetrics	
rack	配置节点标签，该标签将用作 client.rack consumer 配置。
rack	

13.2.86. KafkaConnectS2IStatus 模式参考

用于： KafkaConnectS2I

属性	描述
conditions	状态条件列表。
条件 数组	
observedGeneration	生成 Operator 最后协调的 CRD。
整数	
url	用于管理和监控 Kafka Connect 连接器的 REST API 端点 URL。
字符串	
connectorPlugins	此 Kafka Connect 部署中可用的连接器插件列表。
ConnectorPlugin 数组	
buildConfigName	构建配置的名称。
字符串	
labelSelector	提供此资源的 pod 的标签选择器。
字符串	
replicas	用于提供此资源的当前容器集数量。
整数	

13.2.87. KafkaTopic 架构参考

属性	描述
spec	主题的规范。
KafkaTopicSpec	
status	主题的状态。
KafkaTopicStatus	

13.2.88. KafkaTopicSpec 模式参考

中使用的： *KafkaTopic*

属性	描述
分区 整数	主题应具有的分区数量。这在主题创建后不能减少。可在主题创建后增加，但务必要了解其结果，特别是对于语义分区的主题。缺少时，这将默认为代理配置 for **num.partitions**。
replicas 整数	主题应具有的副本数。如果没有，这将默认为 default. **replication.factor** 的代理配置。
config map	主题配置。
topicName 字符串	主题的名称。如果没有，这将默认为主题的 metadata.name。除非主题名称不是有效的 OpenShift 资源名称，否则建议不要设置此项。

13.2.89. KafkaTopicStatus schema reference

中使用的： *KafkaTopic*

属性	描述
conditions **条件** 数组	状态条件列表。
observedGeneration 整数	生成 Operator 最后协调的 CRD。
topicName 字符串	主题名称.

13.2.90. KafkaUser 架构参考

属性	描述
spec	用户的规范。
KafkaUserSpec	
status	Kafka 用户的状态。
KafkaUserStatus	

13.2.91. KafkaUserSpec 模式参考

用于： KafkaUser

属性	描述
身份验证	为此 Kafka 用户启用身份验证机制。类型取决于给定对象中的 **authentication.type** 属性的值，它必须是 [tls, scram-sha-512] 之一。
KafkaUserTlsClientAuthentication, KafkaUserScramSha512ClientAuthentication	
授权	此 Kafka 用户的授权规则。类型取决于给定对象中的 **authorization.type** 属性的值，它必须是 [simple] 之一。
KafkaUserAuthorizationSimple	
配额	控制客户端使用的代理资源的请求配额。可以强制执行网络带宽和请求速率配额。有关 Kafka 用户配额的 Kafka 文档，请访问
KafkaUserQuotas	
模板	模板，以指定如何生成 **Kafka User Secret**。
KafkaUserTemplate	

13.2.92. KafkaUserTlsClientAuthentication schema reference

用于： KafkaUserSpec

***type** 属性是一个缺点，它区分使用 KafkaUserTlsClientAuthentication 类型与 KafkaUserScramSha512ClientAuthentication。它必须具有类型 KafkaUserTlsClientAuthentication 的值 tls。*

属性	描述
type	必须是 **tls**。
字符串	

13.2.93. KafkaUserScramSha512ClientAuthentication schema reference

用于： ***KafkaUserSpec***

***type** 属性是一个缺点，它区分使用 KafkaUserScramSha512ClientAuthentication 类型与 KafkaUserTlsClientAuthentication 类型。它必须具有类型 KafkaUserScramSha512ClientAuthentication 的值 scr am-sha-512。*

属性	描述
type	必须是 scr **am-sha-512**。
字符串	

13.2.94. KafkaUserAuthorizationSimple schema 参考

用于： ***KafkaUserSpec***

***type** 属性是一个缺点，它区分使用 KafkaUserAuthorizationSimple 类型和将来可能添加的其他子类型。它的值 对于 KafkaUserAuthorizationSimple 类型必须简单。*

属性	描述
type	必须 **简单**。
字符串	
ACL	应应用于此用户的 ACL 规则的列表。
AclRule 数组	

13.2.95. AclRule 架构参考

用于： ***KafkaUserAuthorizationSimple***

AclRule 架构属性的完整列表

在代理使用 AclAuthorizer 时，为 KafkaUser 配置访问控制规则。

授权的 KafkaUser 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  # ...
  authorization:
    type: simple
    acls:
      - resource:
          type: topic
          name: my-topic
          patternType: literal
        operation: Read
      - resource:
          type: topic
          name: my-topic
          patternType: literal
        operation: Describe
      - resource:
          type: group
          name: my-group
          patternType: prefix
        operation: Read
```

13.2.95.1. resource

使用 resource 属性指定规则应用到的资源。

简单授权支持四种资源类型，它们在 type 属性中指定：

- 主题（主题）
- 消费者组（组）
- 集群（集群）
- 事务 ID(actionId)

对于主题、组和事务 ID 资源，您可以在 name 属性中指定规则应用到的资源名称。

集群类型资源没有名称。

使用 patternType 属性将名称指定为 字面 或 前缀。

- 字面名称与在 name 字段中指定完全相同。
- 前缀名称使用 名称 中的值作为前缀，并将规则应用于名称以值开头的所有资源。

13.2.95.2. type

规则 类型，即允许 或拒绝 （目前不支持）操作。

type 字段是可选的。如果未指定 类型，则 ACL 规则被视为 允许 规则。

13.2.95.3. 操作

为规则指定允许或拒绝 的操作。

支持以下操作：

- *读*
- *写*
- *删除*
- *更改*
- *describe*
- *all*
- *IdempotentWrite*
- *ClusterAction*
- *create*
- *AlterConfigs*
- *DescribeConfigs*

只有某些操作可以与每个资源配合使用。

有关 AclAuthorizer、AclAuthorizer、ACL 和支持的资源与操作组合的详情，请参阅 授权和 ACL。

13.2.95.4. host

使用 host 属性指定允许或拒绝该规则的远程主机。

*使用星号(*)来允许或拒绝来自所有主机的操作。host 字段是可选的。如果未指定 主机，则默认使用 * 值。*

13.2.95.5. AclRule 架构属性

属性	描述
主机	ACL 规则中描述的操作被允许或拒绝的主机。
字符串	
操作	允许或拒绝的操作.支持的操作有： Read、Write、Create、Delete、Describe、ClusterAction、AlterConfigs、DescribeConfigs、IdempotentWrite 和 All。
字符串([Read, Write, Delete, Alter, Describe, All, IdempotentWrite, ClusterAction, Create, AlterConfigs, DescribeConfigs])	
resource	指明对其应用给定 ACL 规则的资源。该类型取决于给定对象中的 **resource.type** 属性的值，它必须是 [topic、group、cluster、actionId] 之一。
AclRuleTopicResource, **AclRuleGroupResource**, **AclRuleClusterResource**, **AclRuleTransactionalIdResource**	
type	规则的类型。目前唯一支持的类型是 **允许的**。使用类型为 **allow** 的 ACL 规则来允许用户执行指定的操作。默认值为 **allow**。
字符串（ [allow, deny] 之一）	

13.2.96. AclRuleTopicResource 模式参考

*用于： **AclRule***

*type 属性是一个 Red Hatcriminator，它区分了 AclRuleTopicResource 类型与 **AclRuleGroupResource**、**AclRuleClusterResource**、**AclRuleTransactionalIdResource** 的使用。它必须具有类型 AclRuleTopicResource 的值 主题。*

属性	描述
type	必须成为 **主题**。

属性	描述
字符串	
name 字符串	给定 ACL 规则所应用的资源名称。可以和 **patternType** 字段组合使用前缀模式。
patternType 字符串（[前缀、字面] 之一）	描述 resource 字段中所用的模式。支持的类型是 **字面** 和 **前缀**。对于 **字面** 模式类型，资源字段将用作完整主题名称的定义。使用 **前缀** 模式类型时，资源名称将仅用作前缀。默认值为 **文字**。

13.2.97. AclRuleGroupResource schema reference

用于： *AclRule*

***type** 属性是一个 Red Hatcriminator，它区分 AclRuleGroupResource 类型与 AclRuleTopicResource、AclRuleClusterResource、AclRuleTransactionalIdResource 的使用。它必须具有类型 AclRuleGroupResource 的值 组。*

属性	描述
type 字符串	必须是 **group**。
name 字符串	给定 ACL 规则所应用的资源名称。可以和 **patternType** 字段组合使用前缀模式。
patternType 字符串（[前缀、字面] 之一）	描述 resource 字段中所用的模式。支持的类型是 **字面** 和 **前缀**。对于 **字面** 模式类型，资源字段将用作完整主题名称的定义。使用 **前缀** 模式类型时，资源名称将仅用作前缀。默认值为 **文字**。

13.2.98. AclRuleClusterResource 模式参考

用于： *AclRule*

***type** 属性是一个 Red Hatcriminator，它区分 AclRuleClusterResource 类型与 AclRuleTopicResource、AclRuleGroupResource、AclRuleTransactionalIdResource 的使用。它必*

须具有类型 AclRuleClusterResource 的值 cluster。

属性	描述
type	必须是 **群集**.
字符串	

13.2.99. AclRuleTransactionalIdResource schema reference

用于： AclRule

type 属性是一个发行者，它区分 AclRuleTransactionalIdResource 类型与 AclRule TopicResource、AclRule GroupResource、AclRuleClusterResource 、AclRuleClusterResource 的使用。它的值必须为 AclRuleTransactionalIdResource 类型。

属性	描述
type	必须为 **actionId**。
字符串	
name	给定 ACL 规则所应用的资源名称。可以和 **patternType** 字段组合使用前缀模式。
字符串	
patternType	描述 resource 字段中所用的模式。支持的类型是 **字面** 和 **前缀**。对于 **字面** 模式类型，资源字段将用作全名的定义。使用 **前缀** 模式类型时，资源名称将仅用作前缀。默认值为 **文字**。
字符串（[前缀、字面] 之一）	

13.2.100. KafkaUserQuotas 架构参考

用于： KafkaUserSpec

KafkaUserQuotas 架构属性的完整列表

Kafka 允许用户设置 配额 来控制客户端的资源使用。

13.2.100.1. quotas

您可以将客户端配置为使用以下类型的配额：

- 网络使用量 配额为共享配额的每个客户端组指定字节率阈值。
- CPU 使用率 配额为来自客户端的代理请求指定一个窗口。窗口是客户端发出请求的时间百分比。客户端对代理的 I/O 线程和网络线程发出请求。
- 分区变异 配额限制允许每秒客户端进行的分区变异数量。

分区变异配额可防止 Kafka 集群被并发主题操作所困扰。分区变异是为响应以下类型的用户请求而发生：

- 为新主题创建分区
- 将分区添加到现有主题
- 从主题中删除分区

您可以配置分区变异配额来控制用户请求接受变异的速度。

在很多情形中，为 Kafka 客户端使用配额可能会很有用。考虑错误配置的 Kafka 制作者，它以太高的速率发送请求。这种错误配置可能会导致服务拒绝给其他客户端，因此有问题的客户端会被阻止。通过使用网络限制配额，可以防止这种情况严重影响其他客户端。

AMQ Streams 支持用户级配额，但不支持客户端级别的配额。

Kafka 用户配额配置示例

```
spec:
  quotas:
```

```
producerByteRate: 1048576
consumerByteRate: 2097152
requestPercentage: 55
controllerMutationRate: 10
```

有关 Kafka 用户配额的更多信息，请参阅 Apache Kafka 文档。

13.2.100.2. KafkaUserQuotas 架构属性

属性	描述
consumerByteRate 整数	每个客户端组可以在对组中的客户端进行节流前从代理获取的最大字节数的配额。按每个代理定义.
controllerMutationRate 数字	对创建主题请求接受变异的速率、创建分区请求和删除主题请求的配额。速率是通过创建或删除的分区数量来累计的。
producerByteRate 整数	每个客户端组可以发布至代理的最大字节数的配额，然后再对组中的客户端进行节流。按每个代理定义.
requestPercentage 整数	每个客户端组的最大 CPU 使用率配额作为网络和 I/O 线程的百分比。

13.2.101. KafkaUserTemplate 模式参考

用于： ***KafkaUserSpec***

KafkaUserTemplate 架构属性的完整列表

为 User Operator 创建的 secret 指定额外的标签和注解。

显示 KafkaUserTemplate的示例

-

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: tls
  template:
    secret:
      metadata:
        labels:
          label1: value1
        annotations:
          anno1: value1
  # ...
```

13.2.101.1. KafkaUserTemplate 架构属性

属性	描述
Secret	KafkaUser 资源的模板。该模板允许用户指定如何生成密码或 TLS 证书的 **Secret**。
ResourceTemplate	

13.2.102. KafkaUserStatus 模式参考

用于： ***KafkaUser***

属性	描述
conditions	状态条件列表。
条件 数组	
observedGeneration	生成 Operator 最后协调的 CRD。
整数	
username	用户名.
字符串	

属性	描述
Secret	存储凭证的 **Secret** 名称。
字符串	

13.2.103. KafkaMirrorMaker 模式参考

属性	描述
spec	Kafka MirrorMaker 的规格。
KafkaMirrorMakerSpec	
status	Kafka MirrorMaker 的状态。
KafkaMirrorMakerStatus	

13.2.104. KafkaMirrorMakerSpec 模式参考

用于： ***KafkaMirrorMaker***

KafkaMirrorMakerSpec 架构属性的完整列表

配置 Kafka MirrorMaker。

13.2.104.1. Include

使用 include 属性配置 Kafka MirrorMaker 镜像从源到目标 Kafka 集群的主题列表。

属性允许从最简单的情况下通过单个主题名称到复杂的模式的任何正则表达式。例如，您可以使用 "A|B" 或所有主题(*)来镜像主题 A 和 B。您还可以将用逗号分开的多个正则表达式传递给 Kafka MirrorMaker。

13.2.104.2. KafkaMirrorMakerConsumerSpec and KafkaMirrorMakerProducerSpec

使用 KafkaMirrorMakerConsumerSpec 和 KafkaMirrorMakerProducerSpec 配置源（消费者）和目标(producer)集群。

Kafka MirrorMaker 总是与两个 Kafka 集群（源和目标）一同工作。要建立连接，源的 bootstrap 服务器和目标 Kafka 集群的 bootstrap 服务器被指定为用逗号分隔的 HOSTNAME:PORT 对列表。每个以逗号分开的列表包含一个或多个 Kafka 代理，或者指向指定为 HOSTNAME:PORT 对的 Kafka 代理 的服务。

13.2.104.3. logging

Kafka MirrorMaker 拥有自己的可配置日志记录器：

- mirrormaker.root.logger

MirrorMaker 使用 Apache log4j 日志记录器实现。

使用 logging 属性来配置日志记录器和日志记录器级别。

您可以通过直接（内线）指定日志记录器和级别来设置日志级别，或使用自定义（外部）ConfigMap。如果使用 ConfigMap，则将 logging.valueFrom.configMapKeyRef.name 属性设置为包含外部日志记录配置的 ConfigMap 的名称。在 ConfigMap 中，日志配置使用 log4j.properties 进行 描述。logging.valueFrom.configMapKeyRef.name 和 logging.valueFrom.configMapKeyRef.key 属性均是必需的。在运行 Cluster Operator 时，会使用自定义资源创建使用指定准确日志配置的 ConfigMap，然后在每次协调后重新创建。如果没有指定自定义 ConfigMap，则会使用默认日志设置。如果没有设置特定的日志记录器值，则会继承该日志记录器的上一级日志记录器设置。有关日志级别的更多信息，请参阅 Apache 日志记录服务。

在这里，我们会看到 内联 和外部 日志记录示例：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker
spec:
  # ...
  logging:
    type: inline
    loggers:
      mirrormaker.root.logger: "INFO"
  # ...
```

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker
spec:
```

```
# ...
logging:
  type: external
  valueFrom:
    configMapKeyRef:
      name: customConfigMap
      key: mirror-maker-log4j.properties
# ...
```

垃圾收集器(GC)

也可以使用 jvmOptions 属性 来启用（或禁用）垃圾收集器日志记录。

13.2.104.4. KafkaMirrorMakerSpec 模式属性

属性	描述
version	Kafka MirrorMaker 版本。默认值为 2.8.0。请参阅相关文档，以了解升级或降级版本所需的流程。
字符串	
replicas	**部署** 中的容器集数量。
整数	
image	容器集的 docker 镜像。
字符串	
consumer	源集群配置.
KafkaMirrorMakerConsumerSpec	
producer	目标集群的配置.
KafkaMirrorMakerProducerSpec	
资源	要保留的 CPU 和内存资源。如需更多信息，请参阅核心/v1 资源要求的外部文档。
ResourceRequirements	
whitelist	**whitelist 属性已弃用，现在应使用 spec.include 配置。**镜像中包含的主题列表。此选项允许任何使用 Java 样式的正则表达式的正则表达式。使用表达式 **"A|B"** 实现名为 A 和 B 的两个主题的镜像。或者，作为特殊情况，您可以使用正则表达式 '*' 来镜像所有主题。您还可以指定多个用逗号分开的正则表达式。
字符串	

属性	描述
Include 字符串	镜像中包含的主题列表。此选项允许任何使用 Java 样式的正则表达式的正则表达式。使用表达式 **"A|B"** 实现名为 A 和 B 的两个主题的镜像。或者，作为特殊情况，您可以使用正则表达式 '*' 来镜像所有主题。您还可以指定多个用逗号分开的正则表达式。
jvmOptions **JvmOptions**	容器集的 JVM 选项.
logging **InlineLogging**, **ExternalLogging**	MirrorMaker 的日志记录配置.类型取决于给定对象中的 **logging.type** 属性的值，它必须是 [inline, external] 之一。
metricsConfig **JmxPrometheusExporterMetrics**	指标配置.这个类型取决于给定对象中 **metricsConfig.type** 属性的值，必须是 [jmxPrometheusExporter] 中的一个。
tracing **JaegerTracing**	Kafka MirrorMaker 中的追踪配置。类型取决于给定对象中 **tracing.type** 属性的值，它必须是 [jaeger] 之一。
模板 **KafkaMirrorMakerTemplate**	模板，以指定如何生成 Kafka MirrorMaker 资源、**Deployment 和 Pod**。
livenessProbe **probe**	Pod 存活度检查.
readinessprobe **probe**	Pod 就绪度检查。

13.2.105. KafkaMirrorMakerConsumerSpec schema reference

用于： ***KafkaMirrorMakerSpec***

KafkaMirrorMakerConsumerSpec 模式属性的完整列表

配置 MirrorMaker 使用者。

13.2.105.1. numStreams

使用 consumer.numStreams 属性配置使用者的流数。

您可以通过增加消费者线程的数量来提高镜像主题中的吞吐量。消费者线程属于为 Kafka MirrorMaker 指定的使用者组。主题分区在使用者线程之间分配，消费者线程并行使用消息。

13.2.105.2. offsetCommitInterval

使用 consumer.offsetCommitInterval 属性为使用者配置偏移自动提交间隔。

您可以指定在 Kafka MirrorMaker 消耗源 Kafka 集群的数据后提交偏移的常规时间间隔。时间间隔以毫秒为单位设置，默认值为 84。

13.2.105.3. config

使用 consumer.config 属性为消费者配置 Kafka 选项。

config 属性包含 Kafka MirrorMaker consumer 配置选项作为键，其值以以下 JSON 类型之一的形式设置：

- 字符串
- 数字
- 布尔值

对于使用特定 密码套件 作为 TLS 版本进行客户端连接，您可以配置 allowed ssl 属性。您还可以 配置 ssl.endpoint.identification.algorithm 属性来 启用或禁用主机名验证。

例外

您可以为消费者指定并配置 Apache Kafka 配置文档 中列出的选项。

但是，与以下相关的 AMQ Streams 会自动配置和管理的选项会有例外：

- Kafka 集群 bootstrap 地址
- 安全（加密、身份验证和授权）
- 消费者组标识符
- 拦截器

具体来说，所有键为等于或以以下任一字符串开头的配置选项将被禁止：

- bootstrap.servers
- group.id
- interceptor.classes
- SSL.（不包括特定例外）
- sasl.
- 安全性.

当 config 属性中存在禁止选项时，会忽略它，并把警告信息输出到 Cluster Operator 日志文件中。所有其他选项都传递给 Kafka MirrorMaker。

重要

Cluster Operator 不会验证提供的 config 对象中的密钥或值。当提供无效的配置时，Kafka MirrorMaker 可能无法启动或可能变得不稳定。在这种情况下，Kafka MirrorMaker.spec.consumer.config 对象中的配置应该被修复，Cluster Operator 将推出 Kafka MirrorMaker 的新配置。

13.2.105.4. groupId

使用 consumer.groupId 属性为使用者配置使用者组标识符。

Kafka MirrorMaker 使用 Kafka 使用者来消耗消息，与任何其他 Kafka 消费者客户端一样。从源 Kafka 集群使用的消息会镜像到目标 Kafka 集群。需要组标识符，因为使用者需要成为分配分区的消费者组的一部分。

13.2.105.5. KafkaMirrorMakerConsumerSpec schema properties

属性	描述
numStreams 整数	指定要创建的消费者流线程数量。
offsetCommitInterval 整数	以 ms 为单位指定偏移自动提交间隔。默认值为 60000。
bootstrapServers 字符串	用于建立与 Kafka 集群的初始连接的 host:port 对列表。
groupId 字符串	个唯一字符串，用于标识该消费者所属的消费者组。
身份验证 **KafkaClientAuthenticationTls**, **KafkaClientAuthenticationScramSha512**, **KafkaClientAuthenticationPlain**, **KafkaClientAuthenticationOAuth**	用于连接到集群的身份验证配置。这个类型取决于给定对象中的 **authentication.type** 属性的值，必须是 [tls, scram-sha-512, plain, oauth] 中的一个。

属性	描述
config map	MirrorMaker 消费者配置.无法设置带有以下前缀的属性：ssl.、bootstrap.servers、group.id、sasl.、security.、拦截器.classes（ssl.endpoint.identification.algorithm、ssl.cipher.suites、ssl.protocol、ssl.enabled.protocols 除外）。
tls **KafkaMirrorMakerTls**	将 MirrorMaker 连接到集群的 TLS 配置。

13.2.106. KafkaMirrorMakerTls 模式参考

用于： ***KafkaMirrorMakerConsumerSpec***、***KafkaMirrorMakerProducerSpec***

KafkaMirrorMakerTls 架构属性的完整列表

配置 TLS 可信证书，以将 MirrorMaker 连接到集群。

13.2.106.1. trustedCertificates

使用 trustedCertificates 属性提供 secret 列表。

13.2.106.2. kafkaMirrorMakerTls 架构属性

属性	描述
trustedCertificates **CertSecretSource** 数组	TLS 连接的可信证书。

13.2.107. KafkaMirrorMakerProducerSpec schema reference

用于： ***KafkaMirrorMakerSpec***

KafkaMirrorMakerProducerSpec schema 属性的完整列表

配置 MirrorMaker 制作者。

13.2.107.1. abortOnSendFailure

使用 producer.abortOnSendFailure 属性来配置如何处理来自生产者的消息发送失败。

默认情况下，如果在从 Kafka MirrorMaker 发送消息到 Kafka 集群时发生错误：

- Kafka MirrorMaker 容器在 OpenShift 中终止。
- 然后，重新创建容器。

如果 abort OnSendFailure 选项设为 false，则忽略发送错误的消息。

13.2.107.2. config

使用 producer.config 属性为制作者配置 Kafka 选项。

config 属性包含 Kafka MirrorMaker producer 配置选项作为键，其值在以下 JSON 类型之一中设置：

- 字符串
- 数字
- 布尔值

对于使用特定 密码套件 作为 TLS 版本进行客户端连接，您可以配置 allowed ssl 属性。您还可以 配置 ssl.endpoint.identification.algorithm 属性来 启用或禁用主机名验证。

例外

您可以为生产者指定并配置 Apache Kafka 配置文档 中列出的选项。

但是，与以下相关的 AMQ Streams 会自动配置和管理的选项会有例外：

- Kafka 集群 bootstrap 地址
- 安全（加密、身份验证和授权）
- 拦截器

具体来说，所有键为等于或以以下任一字符串开头的配置选项将被禁止：

- bootstrap.servers
- interceptor.classes
- SSL.（不包括特定例外）
- sasl.
- 安全性.

当 config 属性中存在禁止选项时，会忽略它，并把警告信息输出到 Cluster Operator 日志文件中。所有其他选项都传递给 Kafka MirrorMaker。

重要

Cluster Operator 不会验证提供的 config 对象中的密钥或值。当提供无效的配置时，Kafka MirrorMaker 可能无法启动或可能变得不稳定。在这种情况下，Kafka MirrorMaker.spec.producer.config 对象中的配置应该被修复，Cluster Operator 将推出 Kafka MirrorMaker 的新配置。

13.2.107.3. KafkaMirrorMakerProducerSpec schema properties

属性	描述
bootstrapServers 字符串	用于建立与 Kafka 集群的初始连接的 host:port 对列表。
abortOnSendFailure 布尔值	将 MirrorMaker 设置为在失败发送时退出的标记。默认值为 **true**。
身份验证 **KafkaClientAuthenticationTls**, **KafkaClientAuthenticationScramSha512**, **KafkaClientAuthenticationPlain**, **KafkaClientAuthenticationOAuth**	用于连接到集群的身份验证配置。这个类型取决于给定对象中的 **authentication.type** 属性的值，必须是 [tls, scram-sha-512, plain, oauth] 中的一个。
config map	MirrorMaker 制作者配置.无法设置带有以下前缀的属性：ssl.、bootstrap.servers、sasl.、security.、拦截器.classes（ssl.endpoint.identification.algorithm、ssl.cipher.suites、ssl.protocols、ssl.protocols.protocols 除外）。
tls **KafkaMirrorMakerTls**	将 MirrorMaker 连接到集群的 TLS 配置。

13.2.108. KafkaMirrorMakerTemplate 模式参考

用于： ***KafkaMirrorMakerSpec***

属性	描述
Deployment **DeploymentTemplate**	Kafka MirrorMaker **部署** 模板.

属性	描述
Pod	Kafka MirrorMaker **Pod** 模板。
PodTemplate	
podDisruptionBudget	Kafka MirrorMaker **PodDisruptionBudget** 模板。
PodDisruptionBudgetTemplate	
mirrorMakerContainer	Kafka MirrorMaker 容器模板。
ContainerTemplate	
serviceAccount	Kafka MirrorMaker 服务帐户的模板。
ResourceTemplate	

13.2.109. KafkaMirrorMakerStatus 模式参考

用于： ***KafkaMirrorMaker***

属性	描述
conditions	状态条件列表。
条件 数组	
observedGeneration	生成 Operator 最后协调的 CRD。
整数	
labelSelector	提供此资源的 pod 的标签选择器。
字符串	
replicas	用于提供此资源的当前容器集数量。
整数	

13.2.110. KafkaBridge 模式参考

属性	描述
spec	Kafka 网桥的规格。
KafkaBridgeSpec	
status	Kafka 网桥的状态。
KafkaBridgeStatus	

13.2.111. KafkaBridgeSpec 模式参考

用于： *KafkaBridge*

KafkaBridgeSpec 模式属性的完整列表

配置 Kafka Bridge 集群。

配置选项与：

- *Kafka 集群 bootstrap 地址*
- *安全（加密、身份验证和授权）*
- *使用者配置*
- *制作者配置*
- *HTTP 配置*

13.2.111.1. logging

Kafka Bridge 有自己的可配置日志记录器：

- *logger.bridge*
- *logger.<operation-id>*

您可以替换 日志记录器中的 <operation-id> 。<operation-id> logger 为特定操作设置日志级别：

- *createConsumer*
- *deleteConsumer*
- *订阅*
- *取消订阅*
- *Poll*
- *assign*
- *commit*
- *send*
- *sendToPartition*
- *seekToBeginning*
- *seekToEnd*

- 寻道
- healthy
- Ready
- OpenAPI

每个操作都按照 OpenAPI 规范定义，并且具有一个对应的 API 端点，该端点用于接收来自 HTTP 客户端的请求。您可以更改每个端点的日志级别，以创建关于传入和传出 HTTP 请求的精细日志信息。

每个日志记录器都必须配置为 http.openapi.operation.<operation-id>。例如，为发送操作日志记录器配置日志级别意味着定义以下内容：

```
logger.send.name = http.openapi.operation.send
logger.send.level = DEBUG
```

Kafka Bridge 使用 Apache log4j2 日志记录器实施。日志记录器在 log4j2.properties 文件中定义，该文件具有 健康和 就绪 端点的以下默认配置：

```
logger.healthy.name = http.openapi.operation.healthy
logger.healthy.level = WARN
logger.ready.name = http.openapi.operation.ready
logger.ready.level = WARN
```

所有其他操作的日志级别默认设置为 INFO。

使用 logging 属性来配置日志记录器和日志记录器级别。

您可以通过直接（内线）指定日志记录器和级别来设置日志级别，或使用自定义（外部）ConfigMap。如果使用 ConfigMap，则将 logging.valueFrom.configMapKeyRef.name 属性设置为包含外部日志记录配置的 ConfigMap 的名称。logging.valueFrom.configMapKeyRef.name 和 logging.valueFrom.configMapKeyRef.key 属性是必需的。如果未设置 名称或 密钥，则会使用默认日志记录。在 ConfigMap 中，日志配置使用 log4j.properties 进行 描述。有关日志级别的更多信息，请参阅 Apache 日志记录服务。

此处我们会看到 内联 和外部 记录示例。

内联日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
spec:
  # ...
  logging:
    type: inline
    loggers:
      logger.bridge.level: "INFO"
      # enabling DEBUG just for send operation
      logger.send.name: "http.openapi.operation.send"
      logger.send.level: "DEBUG"
  # ...
```

外部日志记录

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
spec:
  # ...
  logging:
    type: external
    valueFrom:
      configMapKeyRef:
        name: customConfigMap
        key: bridge-logj42.properties
  # ...
```

任何未配置可用的日志记录器将其级别设置为 OFF。

如果使用 Cluster Operator 部署 Kafka Bridge，则会动态应用 Kafka Bridge 日志级别。

如果使用外部日志记录，当日志附加程序被更改时会触发滚动更新。

垃圾收集器(GC)

也可以使用 jvmOptions 属性 来启用（或禁用）垃圾收集器日志记录。

13.2.111.2. KafkaBridgeSpec 模式属性

属性	描述
replicas	**部署** 中的容器集数量。
整数	
image	容器集的 docker 镜像。
字符串	
bootstrapServers	用于建立与 Kafka 集群的初始连接的 host:port 对列表。
字符串	
tls	将 Kafka Bridge 连接到集群的 TLS 配置。
KafkaBridgeTls	
身份验证	用于连接到集群的身份验证配置。这个类型取决于给定对象中的 **authentication.type** 属性的值，必须是 [tls, scram-sha-512, plain, oauth] 中的一个。
KafkaClientAuthenticationTls, **KafkaClientAuthenticationScramSha512**, **KafkaClientAuthenticationPlain**, **KafkaClientAuthenticationOAuth**	
http	与 HTTP 相关的配置。
KafkaBridgeHttpConfig	
adminClient	Kafka AdminClient 相关配置。
KafkaBridgeAdminClientSpec	
consumer	Kafka 消费者相关配置。
KafkaBridgeConsumerSpec	

属性	描述
producer **KafkaBridgeProducerSpec**	Kafka 制作者相关配置。
资源 ResourceRequirements	要保留的 CPU 和内存资源。如需更多信息，请参阅核心/v1 资源要求的外部文档。
jvmOptions **JvmOptions**	**目前不支持** pod 的 JVM 选项。
logging **InlineLogging**, **ExternalLogging**	日志记录 Kafka 网桥配置.类型取决于给定对象中的 **logging.type** 属性的值，它必须是 [inline, external] 之一。
enableMetrics 布尔值	为 Kafka 网桥启用指标。默认值为 false。
livenessProbe **probe**	Pod 存活度检查.
readinessprobe **probe**	Pod 就绪度检查。
模板 **KafkaBridgeTemplate**	Kafka Bridge 资源的模板。该模板允许用户指定如何生成 **Deployment** 和 **Pod**。
tracing **JaegerTracing**	在 Kafka 网桥中追踪的配置。类型取决于给定对象中 **tracing.type** 属性的值，它必须是 [jaeger] 之一。

13.2.112. KafkaBridgeTls 模式参考

用于：KafkaBridgeSpec

属性	描述
trustedCertificates	TLS 连接的可信证书。
CertSecretSource 数组	

13.2.113. KafkaBridgeHttpConfig schema reference

用于： KafkaBridgeSpec

KafkaBridgeHttpConfig 模式属性的完整列表

为 Kafka 网桥配置对 Kafka 集群的 HTTP 访问。

默认 HTTP 配置是 Kafka Bridge 侦听端口 8080。

13.2.113.1. CORS

除了启用对 Kafka 集群的 HTTP 访问外，HTTP 属性还提供通过跨操作资源共享(CORS)启用和定义 Kafka 网桥访问控制的功能。CORS 是一种 HTTP 机制，它允许浏览器从多个来源访问选定的资源。要配置 CORS，您需要定义允许的资源来源和 HTTP 访问方法列表。对于原始卷，您可以使用 URL 或 Java 正则表达式。

Kafka Bridge HTTP 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  http:
    port: 8080
    cors:
      allowedOrigins: "https://strimzi.io"
      allowedMethods: "GET,POST,PUT,DELETE,OPTIONS,PATCH"
  # ...
```

13.2.113.2. KafkaBridgeHttpConfig schema properties

属性	描述
port	是侦听的服务器的端口。
整数	
CORS	HTTP 网桥的 CORS 配置.
KafkaBridgeHttpCors	

13.2.114. KafkaBridgeHttpCors schema reference

用于： *KafkaBridgeHttpConfig*

属性	描述
allowedOrigins	允许的来源列表。可以使用 Java 正则表达式.
字符串数组	
allowedMethods	允许的 HTTP 方法列表。
字符串数组	

13.2.115. KafkaBridgeAdminClientSpec schema reference

用于： *KafkaBridgeSpec*

属性	描述
config	Kafka AdminClient 配置，用于网桥创建的 AdminClient 实例。
map	

13.2.116. KafkaBridgeConsumerSpec schema reference

用于： KafkaBridgeSpec

KafkaBridgeConsumerSpec 模式属性的完整列表

将 Kafka 网桥的使用者选项配置为密钥。

这些值可以是以下 JSON 类型之一：

- 字符串
- 数字
- 布尔值

您可以为消费者指定并配置 Apache Kafka 配置文档 中列出的选项，但那些直接由 AMQ Streams 管理的选项除外。具体来说，所有键为等于或以以下任一字符串开头的配置选项将被禁止：

- ssl.
- sasl.
- 安全性.
- bootstrap.servers
- group.id

当 config 属性中存在一个禁止选项时，它将被忽略，并会在 Cluster Operator 日志文件中输出警告信息。所有其他选项将传递给 Kafka

重要

Cluster Operator 不验证 config 对象中的键或值。如果提供了无效的配置，Kafka Bridge 集群可能不会启动，或者可能会变得不稳定。修复配置，以便 Cluster Operator 可以将新配置部署到所有 Kafka Bridge 节点。

禁止的选项有例外。对于使用特定 密码套件 作为 TLS 版本进行客户端连接，您可以配置 allowed ssl 属性。

Kafka Bridge consumer 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  consumer:
    config:
      auto.offset.reset: earliest
      enable.auto.commit: true
      ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384"
      ssl.enabled.protocols: "TLSv1.2"
      ssl.protocol: "TLSv1.2"
      ssl.endpoint.identification.algorithm: HTTPS
    # ...
```

13.2.116.1. KafkaBridgeConsumerSpec schema properties

属性	描述
config	Kafka 使用者配置，用于网桥创建的消费者实例。无法设置带有以下前缀的属性： ssl.、bootstrap.servers、group.id、sasl.、security.（ssl.endpoint.identification.algorithm、ssl.cipher.suites、ssl.protocols、ssl.protocols、ssl.enabled.protocols 除外）。
map	

13.2.117. KafkaBridgeProducerSpec schema reference

用于： KafkaBridgeSpec

KafkaBridgeProducerSpec schema 属性的完整列表

将 Kafka 网桥的制作者选项配置为密钥。

这些值可以是以下 JSON 类型之一：

- 字符串
- 数字
- 布尔值

您可以为生产者指定并配置 Apache Kafka 配置文档 中列出的选项，但那些直接由 AMQ Streams 管理的选项除外。具体来说，所有键为等于或以以下任一字符串开头的配置选项将被禁止：

- ssl.
- sasl.
- 安全性.
- bootstrap.servers

当 config 属性中存在一个禁止选项时，它将被忽略，并会在 Cluster Operator 日志文件中输出警告信息。所有其他选项将传递给 Kafka

重要

Cluster Operator 不验证 config 对象中的键或值。如果提供了无效的配置，Kafka Bridge 集群可能不会启动，或者可能会变得不稳定。修复配置，以便 Cluster Operator 可以将新配置部署到所有 Kafka Bridge 节点。

禁止的选项有例外。对于使用特定 密码套件 作为 TLS 版本进行客户端连接，您可以配置 allowed ssl 属性。

Kafka Bridge producer 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  producer:
    config:
      acks: 1
      delivery.timeout.ms: 300000
      ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384"
      ssl.enabled.protocols: "TLSv1.2"
      ssl.protocol: "TLSv1.2"
      ssl.endpoint.identification.algorithm: HTTPS
    # ...
```

13.2.117.1. KafkaBridgeProducerSpec schema properties

属性	描述
config	Kafka 制作者配置，用于网桥创建的制作者实例。无法设置带有以下前缀的属性：ssl.、bootstrap.servers、sasl. and security. （除 ssl.endpoint.identification.algorithm, ssl.cipher.suites, ssl.protocol, ssl.protocols ssl.enabled.protocols 外）。
map	

13.2.118. KafkaBridgeTemplate 模式引用

用于：KafkaBridgeSpec

属性	描述
Deployment	Kafka 网桥 **部署的模板.**
DeploymentTemplate	
Pod	Kafka 网桥 **Pod** 模板.
PodTemplate	
apiService	Kafka Bridge API **服务模板**。
InternalServiceTemplate	
podDisruptionBudget	Kafka Bridge **PodDisruptionBudget** 模板。
PodDisruptionBudgetTemplate	
bridgeContainer	Kafka Bridge 容器的模板。
ContainerTemplate	
serviceAccount	Kafka Bridge 服务帐户的模板。
ResourceTemplate	

13.2.119. KafkaBridgeStatus 模式参考

用于：KafkaBridge

属性	描述
conditions	状态条件列表。
条件 数组	
observedGeneration	生成 Operator 最后协调的 CRD。
整数	
url	外部客户端应用程序可以访问 Kafka 网桥的 URL。

属性	描述
字符串	
labelSelector	提供此资源的 pod 的标签选择器。
字符串	
replicas	用于提供此资源的当前容器集数量。
整数	

13.2.120. KafkaConnector 模式参考

属性	描述
spec	Kafka Connector 的规格。
KafkaConnectorSpec	
status	Kafka Connector 的状态。
KafkaConnectorStatus	

13.2.121. KafkaConnectorSpec 模式参考

用于： ***KafkaConnector***

属性	描述
class	Kafka 连接器的类.
字符串	
tasksMax	Kafka Connector 的任务数量上限。
整数	
config	Kafka Connector 配置。无法设置以下属性：connector.class、tasks.max。
map	
pause	是否应暂停连接器。默认为false。

属性	描述
布尔值	

13.2.122. KafkaConnectorStatus 模式参考

用于： *KafkaConnector*

属性	描述
conditions	状态条件列表。
条件 数组	
observedGeneration	生成 Operator 最后协调的 CRD。
整数	
connectorStatus	连接器状态，如 Kafka Connect REST API 报告。
map	
tasksMax	Kafka Connector 的任务数量上限。
整数	
主题	Kafka Connector 使用的主题列表。
字符串数组	

13.2.123. KafkaMirrorMaker2 模式参考

属性	描述
spec	Kafka MirrorMaker 2.0 集群的规格。
KafkaMirrorMaker2Spec	
status	Kafka MirrorMaker 2.0 集群的状态。
KafkaMirrorMaker2Status	

13.2.124. KafkaMirrorMaker2Spec 模式参考

用于： ***KafkaMirrorMaker2***

属性	描述
version	Kafka Connect 版本。默认值为 2.8.0。请参阅用户文档以了解升级或降级版本所需的流程。
字符串	
replicas	Kafka Connect 组中的 pod 数量。
整数	
image	容器集的 docker 镜像。
字符串	
connectCluster	用于 Kafka Connect 的集群别名。别名必须与 **spec.clusters** 列表中的集群匹配。
字符串	
clusters	用于镜像的 Kafka 集群。
KafkaMirrorMaker2ClusterSpec 数组	
mirrors	配置 MirrorMaker 2.0 连接器.
KafkaMirrorMaker2MirrorSpec 数组	
资源	CPU 和内存资源以及请求的初始资源的最大限值。如需更多信息，请参阅核心/v1 资源要求的外部文档。
ResourceRequirements	
livenessProbe	Pod 存活度检查.
probe	
readinessprobe	Pod 就绪度检查。
probe	
jvmOptions	容器集的 JVM 选项.
JvmOptions	
jmxOptions	JMX 选项.

属性	描述
KafkaJmxOptions	
logging **InlineLogging**, **ExternalLogging**	Kafka Connect 的日志配置.类型取决于给定对象中的 **logging.type** 属性的值，它必须是 [inline, external] 之一。
tracing **JaegerTracing**	Kafka Connect 中的追踪配置。类型取决于给定对象中 **tracing.type** 属性的值，它必须是 [jaeger] 之一。
模板 **KafkaConnectTemplate**	Kafka Connect 和 Kafka Connect S2I 资源的模板。该模板允许用户指定如何生成 **Deployment**、**Pod** 和 **Service**。
externalConfiguration **ExternalConfiguration**	将数据从 Secret 或 ConfigMap 传递给 Kafka Connect Pod，并使用它们来配置连接器。
metricsConfig **JmxPrometheusExporterMetrics**	指标配置.这个类型取决于给定对象中 **metricsConfig.type** 属性的值，必须是 [jmxPrometheusExporter] 中的一个。

13.2.125. KafkaMirrorMaker2ClusterSpec 模式参考

用于： ***KafkaMirrorMaker2Spec***

KafkaMirrorMaker2ClusterSpec schema 属性的完整列表

配置 Kafka 集群以进行镜像。

13.2.125.1. config

使用 配置 属性配置 Kafka 选项。

标准 Apache Kafka 配置可能会提供，仅限于不是由 AMQ Streams 直接管理的属性。

对于使用特定 密码套件 作为 TLS 版本进行客户端连接，您可以配置 allowed ssl 属性。您还可以 配

置 ssl.endpoint.identification.algorithm 属性来 启用或禁用主机名验证。

13.2.125.2. KafkaMirrorMaker2ClusterSpec schema properties

属性	描述
Alias	用于引用 Kafka 集群的别名。
字符串	
bootstrapServers	用于建立与 Kafka 集群连接的以逗号分隔的 **host:port** 对列表。
字符串	
tls	用于将 MirrorMaker 2.0 连接器连接到集群的 TLS 配置。
KafkaMirrorMaker2Tls	
身份验证	用于连接到集群的身份验证配置。这个类型取决于给定对象中的 **authentication.type** 属性的值，必须是 [tls, scram-sha-512, plain, oauth] 中的一个。
KafkaClientAuthenticationTls, **KafkaClientAuthenticationScramSha512**, **KafkaClientAuthenticationPlain**, **KafkaClientAuthenticationOAuth**	
config	MirrorMaker 2.0 集群配置。无法设置带有以下前缀的属性：ssl.、sasl.、security.、listeners、plugin.path、re.、bootstrap.servers、consumer.interceptor.classes、producer.interceptor.classes（除：ssl.endpoint.identification.algorithm 除外）SSL.cipher.suites、ssl.protocol、ssl.enabled.protocols。
map	

13.2.126. KafkaMirrorMaker2Tls 模式参考

用于： KafkaMirrorMaker2ClusterSpec

属性	描述
trustedCertificates	TLS 连接的可信证书。
CertSecretSource 数组	

13.2.127. KafkaMirrorMaker2MirrorSpec 模式参考

用于： ***KafkaMirrorMaker2Spec***

属性	描述
sourceCluster 字符串	Kafka MirrorMaker 2.0 连接器使用的源集群的别名。别名必须与 **spec.clusters** 列表中的集群匹配。
targetCluster 字符串	Kafka MirrorMaker 2.0 连接器使用的目标集群的别名。别名必须与 **spec.clusters** 列表中的集群匹配。
sourceConnector **KafkaMirrorMaker2ConnectorSpec**	Kafka MirrorMaker 2.0 源连接器的规格。
heartbeatConnector **KafkaMirrorMaker2ConnectorSpec**	Kafka MirrorMaker 2.0 heartbeat 连接器的规格。
checkpointConnector **KafkaMirrorMaker2ConnectorSpec**	Kafka MirrorMaker 2.0 checkpoint 连接器的规格。
topicsPattern 字符串	与要镜像的主题匹配的正则表达式，如 "topic1|topic2|topic3"。也支持以逗号分隔的列表。
topicsBlacklistPattern 字符串	**topic BlacklistPattern 属性已弃用，现在应使用 .spec.mirrors.topicsExcludePattern 配置。**与要从镜像中排除的主题匹配的正则表达式。也支持以逗号分隔的列表。
topicsExcludePattern 字符串	与要从镜像中排除的主题匹配的正则表达式。也支持以逗号分隔的列表。
groupsPattern 字符串	与要镜像的使用者组匹配的正则表达式。也支持以逗号分隔的列表。
groupsBlacklistPattern 字符串	**groupsBlacklistPattern 属性已弃用，现在应使用 .spec.mirrors.groupsExcludePattern 配置。**与使用者组匹配的正则表达式，从镜像中排除。也支持以逗号分隔的列表。

属性	描述
groupsExcludePattern 字符串	与使用者组匹配的正则表达式，从镜像中排除。也支持以逗号分隔的列表。

13.2.128. KafkaMirrorMaker2ConnectorSpec schema reference

用于： *KafkaMirrorMaker2MirrorSpec*

属性	描述
tasksMax 整数	Kafka Connector 的任务数量上限。
config map	Kafka Connector 配置。无法设置以下属性：connector.class、tasks.max。
pause 布尔值	是否应暂停连接器。默认为false。

13.2.129. KafkaMirrorMaker2Status 模式参考

用于： *KafkaMirrorMaker2*

属性	描述
conditions **条件** 数组	状态条件列表。
observedGeneration 整数	生成 Operator 最后协调的 CRD。
url 字符串	用于管理和监控 Kafka Connect 连接器的 REST API 端点 URL。

属性	描述
connectorPlugins **ConnectorPlugin** 数组	此 Kafka Connect 部署中可用的连接器插件列表。
连接器 映射数组	MirrorMaker 2.0 连接器状态列表，如 Kafka Connect REST API 报告。
labelSelector 字符串	提供此资源的 pod 的标签选择器。
replicas 整数	用于提供此资源的当前容器集数量。

13.2.130. KafkaRebalance 模式参考

属性	描述
spec **KafkaRebalanceSpec**	Kafka 重新平衡的规格。
status **KafkaRebalanceStatus**	Kafka 重新平衡的状态。

13.2.131. KafkaRebalanceSpec 模式参考

中使用的： ***KafkaRebalance***

属性	描述
目标 字符串数组	通过降低优先级排序的目标列表，用于生成和执行重新平衡提议。支持的目标位于 https://github.com/linkedin/cruise-control#goals。如果提供空的目标列表，则使用 default.goals Cruise Control 配置参数中声明的目标。

属性	描述
skipHardGoalCheck 布尔值	是否允许在优化后跳过 Kafka CR 中指定的硬目标。当其中一些困难目标妨碍找到平衡解决方案时，这非常有用。默认值为 false。
excludedTopics 字符串	在优化计算中排除任何匹配主题的正则表达式。此表达式将由 java.util.regex.Pattern 类解析；有关支持的 formar 的更多信息，请查阅该类的文档。
concurrentPartitionMovementsPerBroker 整数	持续分区副本移动的上限将进入/退出每个代理。默认值为 5。
concurrentIntraBrokerPartitionMovements 整数	每个代理内磁盘之间连续分区副本移动的上限。默认值为 2。
concurrentLeaderMovements 整数	持续分区领导运动的上限。默认值为 1000。
replicationThrottle 整数	上限（以字节/秒为单位）用于移动副本的带宽上。默认情况下没有限制。
replicaMovementStrategies 字符串数组	用于确定生成优化建议中副本移动的执行顺序的策略类名称列表。默认情况下，使用 BaseReplicaMovementStrategy，它将按照生成副本移动的顺序执行副本移动。

13.2.132. KafkaRebalanceStatus 模式参考

***中使用的：** KafkaRebalance*

属性	描述
conditions 条件 数组	状态条件列表。
observedGeneration 整数	生成 Operator 最后协调的 CRD。

属性	描述
sessionId	与这个 KafkaRebalance 资源相关的 Cruise Control 的会话标识符。Kafka Rebalance operator 使用它来跟踪正在进行的重新平衡操作的状态。
字符串	
optimizationResult	描述优化结果的 JSON 对象。
map	

附录 A. 使用您的订阅

AMQ Streams 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，请使用客户门户查找要下载的相关文件。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. 查找 INTEGRATION 类别 中的 Red Hat AMQ Streams 条目。
3. 选择所需的 AMQ Streams 产品。此时会打开 Software Downloads 页面。
4. 单击组件的 Download 链接。

2021-12-18 14:40:02 +1000 修订