Red Hat Streams for Apache Kafka 2.7

在 OpenShift 中部署和管理 Apache Kafka 的流

在 OpenShift Container Platform 上部署和管理 Apache Kafka 2.7 的流

Last Updated: 2024-06-28

Red Hat Streams for Apache Kafka 2.7 在 OpenShift 中部署和管理 Apache Kafka 的流

在 OpenShift Container Platform 上部署和管理 Apache Kafka 2.7 的流

法律通告

摘要

使用 Apache Kafka operator 的 Streams 来部署 Kafka 组件。配置 Kafka 组件以构建大规模消息传递网络。设置对 Kafka 集群的安全客户端访问，以及指标和距离追踪等功能。升级以使用新功能，包括最新支持的 Kafka 版本。

目录

前言

对红帽文档提供反馈

我们感谢您对我们文档的反馈。

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

前提条件

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

流程

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

感谢您花时间来提供反馈。

第 1 章 部署概述

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

本指南提供有关部署和管理 Apache Kafka Streams 的说明。部署选项和步骤使用 Apache Kafka Streams 中包含的示例安装文件进行。虽然指南突出显示了重要的配置注意事项，但它并不涵盖所有可用选项。要深入了解 Kafka 组件配置选项，请参阅 Apache Kafka 自定义资源 API 参考 。

除了部署说明外，指南还提供了部署前和部署后指导。它涵盖了设置并保护对 Kafka 集群的客户端访问。另外，它探索额外的部署选项，如指标集成、分布式追踪和集群管理工具，如 Cruise Control，以及 Apache Kafka Drain Cleaner 的 Streams。您还将发现有关管理 Apache Kafka 的流的建议，并微调 Kafka 配置以获得最佳性能。

为 Apache Kafka 和 Kafka 的 Streams 提供了升级说明，以帮助保持部署最新。

Apache Kafka 的 Streams 旨在与所有类型的 OpenShift 集群兼容，无论它们的发布是什么。无论您的部署涉及公共云或私有云，还是要设置本地开发环境，本指南中的说明适用于所有情况。

1.1. APACHE KAFKA 自定义资源流

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

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

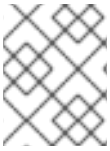
CRD 还允许 Apache Kafka 资源的 Streams 从原生 OpenShift 功能中受益，如 CLI 访问和配置验证。

1.1.1. Apache Kafka 自定义资源流示例

CRD 需要在集群中一次性安装，以定义用于实例化和管理 Apache Kafka 特定资源的 Streams 的 schema。

在安装 CRD 中添加新的自定义资源类型后，您可以根据规格创建资源实例。

根据集群设置，安装通常需要集群管理员特权。



注意

访问自定义资源仅限于 Apache Kafka 管理员的 Streams。如需更多信息，请参阅 第 4.6 节 “为 Apache Kafka 管理员设计流”。

在 OpenShift 集群中，CRD 定义了一个新的资源 **kind**，如 **kind:Kafka**。

Kubernetes API 服务器允许根据类型创建自定义资源，并通过 CRD 了解在添加到 OpenShift 时如何验证和存储自定义资源。

每个特定于 Apache Kafka 的自定义资源流都符合为资源类型定义的 CRD 定义的 schema。**Apache Kafka 组件的 Streams 自定义资源具有通用配置属性，这些属性在 spec 下定义。**

要了解 CRD 和自定义资源之间的关系，请参阅 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、名称和标签来标识 CRD。

2

此 CRD 的规格，包括组（域）名称、复数名称和受支持的模式版本，它们用于 URL 用于访问主题的 API。其他名称用于识别 CLI 中的实例资源。例如，oc get kafkatopic my-topic 或 oc get kafkatopics。

3

CLI 命令可以使用短名称。例如，oc get kt 是 oc get kafkatopic 的缩写形式。

4

对自定义资源使用 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 验证提供了创建主题自定义资源的验证。例如，主题至少需要一个分区和一个副本。

注意

您可以识别由 Apache Kafka 安装文件流提供的 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 标识自定义资源为实例的 CRD。

2 标签，仅适用于 KafkaTopic 和 KafkaUser 资源，用于定义主题或用户所属的 Kafka 集群的名称（与 Kafka 资源的名称相同）。

3 spec 显示主题的分区和副本数，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，并且指定了日志的分段文件大小。

4 KafkaTopic 资源的状态条件。在 lastTransitionTime 时 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，它使用与 Kubernetes API 内置资源相同的验证。

创建 KafkaTopic 自定义资源后，主题 Operator 会收到通知，并在 Apache Kafka 的 Streams 中创建对应的 Kafka 主题。

其他资源

- 使用 CustomResourceDefinitions 扩展 Kubernetes API
- 为 Apache Kafka 提供的流配置文件示例

1.1.2. 对自定义资源执行 oc 操作

您可以使用 oc 命令检索信息，并在 Apache Kafka 自定义资源的 Streams 上执行其他操作。使用 oc

命令，如 get, describe, edit, 或 delete, 对资源类型执行操作。例如，oc get kafkatopics 检索所有 Kafka 主题和 oc get kafkas 列表，检索所有部署的 Kafka 集群。

当引用资源类型时，您可以使用单数和复数名称： oc get kafkas 获取与 oc get kafka 相同的结果。

您还可以使用资源 *的短名称*。学习短名称可在为 Apache Kafka 管理流时节省时间。Kafka 的短名称为 k，因此您也可以运行 oc get k 来列出所有 Kafka 集群。

列出 Kafka 集群

```
oc get k

NAME       DESIRED KAFKA REPLICAS   DESIRED ZK REPLICAS
my-cluster   3                        3
```

表 1.1. 每个 Apache Kafka 资源流的长和短名称

Apache Kafka 资源流	长名称	短名称
Kafka	kafka	k
Kafka 节点池	kafkanodepool	knp
Kafka 主题	kafkatopic	kt
Kafka 用户	kafkauser	ku
Kafka Connect	kafkaconnect	kc
Kafka Connector	kafkaconnector	kctr
Kafka Mirror Maker	kafkamirrormaker	kmm
Kafka Mirror Maker 2	kafkamirrormaker2	kmm2
Kafka Bridge	kafkabridge	kb
Kafka Rebalance	KafkaRebalance	kr

1.1.2.1. 资源类别

自定义资源的类别也可以在 oc 命令中使用。

Apache Kafka 自定义资源的所有流属于类别 strimzi，因此您可以使用 strimzi 获取 Apache Kafka 资源的所有流。

例如，运行 oc get strimzi 列出给定命名空间中 Apache Kafka 自定义资源的所有流。

列出所有自定义资源

```
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

列出所有资源类型和名称

```
oc get strimzi -o name

kafka.kafka.strimzi.io/my-cluster
kafkatopic.kafka.strimzi.io/kafka-apps
kafkauser.kafka.strimzi.io/my-user
```

您可以将这个 strimzi 命令与其他命令合并。例如，您可以将其传给 oc delete 命令，以删除单个命令中的所有资源。

删除所有自定义资源

```
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
```

删除单个操作中的所有资源可能很有用，例如，当您为 Apache Kafka 功能测试新流时。

1.1.2.2. 查询子资源的状态

您可以使用其他值传递给 -o 选项。例如，通过使用 -o yaml，您可以以 YAML 格式获取输出。使用 -o json 将返回 JSON。

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[? (@.name=="tls")].bootstrapServers} 从 Kafka 自定义资源的状态中获取 bootstrap 地址，并在 Kafka 客户端中使用它。

在这里，命令检索名为 tls 的监听程序的 bootstrapServers 值：

检索 bootstrap 地址

```
oc get kafka my-cluster -o=jsonpath='{.status.listeners[?(@.name=="tls")].bootstrapServers}{"\n"}'
```

```
my-cluster-kafka-bootstrap.myproject.svc:9093
```

通过更改 name 条件，您还可以获取其他 Kafka 侦听程序的地址。

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。

1.1.3. Apache Kafka 自定义资源状态信息的流

status 属性提供某些自定义资源的状态信息。

下表列出了提供状态信息（部署时）以及定义 status 属性的 schema 的自定义资源。

如需有关 schema 的更多信息，请参阅 Apache Kafka 自定义资源 API 参考流。

表 1.2. 提供状态信息的自定义资源

Apache Kafka 资源流	模式参考	在... 中发布状态信息
Kafka	**KafkaStatus** 模式参考	Kafka 集群、其监听程序和节点池
KafkaNodePool	**KafkaNodePoolStatus** 模式参考	节点池中的节点、其角色和关联的 Kafka 集群
KafkaTopic	**KafkaTopicStatus** 模式参考	Kafka 集群中的 Kafka 主题
KafkaUser	**KafkaUserStatus** 模式参考	Kafka 集群中的 Kafka 用户
KafkaConnect	**KafkaConnectStatus** schema 参考	Kafka Connect 集群和连接器插件
KafkaConnector	**KafkaConnectorStatus** 模式参考	**KafkaConnector** 资源
KafkaMirrorMaker2	**KafkaMirrorMaker2Status** 模式参考	Kafka MirrorMaker 2 集群和内部连接器
KafkaMirrorMaker	**KafkaMirrorMakerStatus** 模式参考	Kafka MirrorMaker 集群

Apache Kafka 资源流	模式参考	在... 中发布状态信息
KafkaBridge	**KafkaBridgeStatus** schema reference	Apache Kafka Bridge 的流
KafkaRebalance	**KafkaRebalance** 模式参考	重新平衡的状态和结果
StrimziPodSet	**StrimziPodSetStatus** schema reference	受管、使用当前版本和就绪状态的 pod 数量

资源的 **status** 属性提供有关资源状态的信息。**status.conditions** 和 **status.observedGeneration** 属性适用于所有资源。

status.conditions

状态条件描述了资源的*当前状态*。状态条件属性可用于跟踪与资源相关的进度，实现 ***其所需状态***，如 **spec** 中指定的配置所定义。**Status** 条件属性提供资源更改的时间和原因，以及防止或延迟 **Operator** 正常状态的事件详情。

status.observedGeneration

最后观察到的生成表示 **Cluster Operator** 资源的最新协调。如果 **observedGeneration** 的值与 **metadata.generation** 的值（部署的当前版本）不同，**Operator** 尚未处理对资源的最新更新。如果这些值相同，状态信息反映了资源的最新更改。

status 属性还提供特定于资源的信息。例如，**KafkaStatus** 提供有关侦听器地址的信息，以及 **Kafka** 集群的 **ID**。

KafkaStatus 还提供有关使用的 **Apache Kafka** 版本的 **Kafka** 和 **Streams** 的信息。您可以检查 **operatorLastSuccessfulVersion** 和 **kafkaVersion** 的值，以确定是否完成了对 **Apache Kafka** 的 **Streams** 升级

Apache Kafka 的 **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** 在自定义资源上执行更新时，则其状态不可编辑。另外，更改 状态 不会影响 **Kafka** 集群的配置。

在这里，我们看到 **Kafka** 自定义资源的状态 属性。

Kafka 自定义资源状态

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
spec:
  # ...
status:
  clusterId: XP9FP2P-RByvEy0W4cOEUA (1)
  conditions: (2)
    - lastTransitionTime: '2023-01-20T17:56:29.396588Z'
      status: 'True'
      type: Ready (3)
  kafkaMetadataState: KRaft (4)
  kafkaVersion: 3.7.0 (5)
  kafkaNodePools: (6)
    - name: broker
    - name: controller
  listeners: (7)
    - addresses:
        - host: my-cluster-kafka-bootstrap.prm-project.svc
          port: 9092
      bootstrapServers: 'my-cluster-kafka-bootstrap.prm-project.svc:9092'
      name: plain
    - addresses:
        - host: my-cluster-kafka-bootstrap.prm-project.svc
          port: 9093
      bootstrapServers: 'my-cluster-kafka-bootstrap.prm-project.svc:9093'
      certificates:
        - |
          -----BEGIN CERTIFICATE-----

          -----END CERTIFICATE-----
      name: tls
    - addresses:
        - host: >-
            2054284155.us-east-2.elb.amazonaws.com
          port: 9095
      bootstrapServers: >-
        2054284155.us-east-2.elb.amazonaws.com:9095
      certificates:
        - |
          -----BEGIN CERTIFICATE-----

          -----END CERTIFICATE-----
      name: external3
    - addresses:
        - host: ip-10-0-172-202.us-east-2.compute.internal
          port: 31644
      bootstrapServers: 'ip-10-0-172-202.us-east-2.compute.internal:31644'
      certificates:
        - |
          -----BEGIN CERTIFICATE-----

          -----END CERTIFICATE-----
```

```
    name: external4
observedGeneration: 3 8
operatorLastSuccessfulVersion: 2.7 9
```

1

Kafka 集群 ID。

2

状态条件 描述了 Kafka 集群的当前状态。

3

Ready 条件表示 Cluster Operator 认为 Kafka 集群可以处理流量。

4

显示使用(KRaft 或 ZooKeeper)的机制来管理 Kafka 元数据和协调操作的 Kafka 元数据状态。

5

Kafka 集群使用的 Kafka 版本。

6

属于 Kafka 集群的节点池。

7

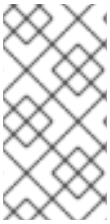
监听器根据类型 描述 Kafka bootstrap 地址。

8

observedGeneration 值表示 Cluster Operator 的 Kafka 自定义资源的最后协调。

9

成功完成最后一次协调的 Operator 版本。



注意

状态中列出的 Kafka bootstrap 地址没有表示这些端点或 Kafka 集群处于 Ready 状态。

1.1.4. 查找自定义资源的状态

将 oc 与自定义资源 的状态 子资源搭配使用，以检索资源的相关信息。

先决条件

- 一个 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}' | jq
```

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 ）来微调您要查看的状态信息。

使用 jq 命令行 JSON 解析器工具 可以更轻松地读取输出。

其他资源

- 有关使用 JSONPath 的更多信息，请参阅 JSONPath 支持。

1.2. APACHE KAFKA OPERATOR 的流

Apache Kafka operator 的流专门构建有专家操作知识，以便在 OpenShift 上有效地管理 Kafka。每个操作器都执行不同的功能。

Cluster Operator

Cluster Operator 在 OpenShift 上处理 Apache Kafka 集群的部署和管理。它自动设置 Kafka 代理和其他 Kafka 组件和资源。

Topic Operator

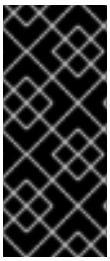
主题 Operator 管理 Kafka 集群中的创建、配置和删除主题。

User Operator

User Operator 管理需要访问 Kafka 代理的 Kafka 用户。

当您为 Apache Kafka 部署流时，您首先部署 Cluster Operator。然后，Cluster Operator 已准备好处理 Kafka 的部署。您还可以使用 Cluster Operator （推荐）或独立 Operator 部署 Topic Operator 和 User Operator。您可以将独立 Operator 与不是由 Cluster Operator 管理的 Kafka 集群一起使用。

主题 Operator 和 User Operator 是实体 Operator 的一部分。Cluster Operator 可以基于 Entity Operator 配置部署一个或多个 Operator。



重要

要部署独立 Operator，您需要设置环境变量以连接到 Kafka 集群。如果您使用 Cluster Operator 部署 Operator，则不需要设置这些环境变量，因为它们将由 Cluster Operator 设置。

1.2.1. 在 OpenShift 命名空间中监视 Apache Kafka 资源的流

Operator 会监视和管理 OpenShift 命名空间中的 Apache Kafka 资源的流。Cluster Operator 可以监控 OpenShift 集群中的单个命名空间、多个命名空间或所有命名空间。主题 Operator 和用户 Operator 可以监视单个命名空间。

- Cluster Operator 监视 Kafka 资源
- 主题 Operator 监视 KafkaTopic 资源
- User Operator 监视 KafkaUser 资源

主题 Operator 和 User Operator 只能监视命名空间中的单个 Kafka 集群。它们只能连接到单个 Kafka 集群。

如果多个主题 Operator 监控同一命名空间，则可能会出现名称冲突和主题删除。这是因为每个 Kafka 集群都使用名称相同的 Kafka 主题（如 __consumer_offsets）。请确定只有一个主题 Operator 会监视给定的命名空间。

当将多个用户 Operator 与单个命名空间一起使用时，带有给定用户名的用户可在多个 Kafka 集群中存在。

如果您使用 Cluster Operator 部署 Topic Operator 和 User Operator，它们会默认监控 Cluster Operator 部署的 Kafka 集群。您还可以使用 operator 配置中的 watchedNamespace 指定命名空间。

对于每个 Operator 的独立部署，您可以指定一个命名空间和与 Kafka 集群的连接，以便在配置中监视。

1.2.2. 管理 RBAC 资源

Cluster Operator 为需要访问 OpenShift 资源的 Apache Kafka 组件创建和管理基于角色的访问控制 (RBAC)资源。

要使 Cluster Operator 正常工作，OpenShift 集群中需要权限与 Kafka 资源交互，如 Kafka 和 KafkaConnect，以及 ConfigMap、Pod、Deployment 和 Service 等受管资源。

通过以下 OpenShift RBAC 资源指定权限：

- ServiceAccount
- Role 和 ClusterRole
- RoleBinding 和 ClusterRoleBinding

1.2.2.1. 为 Apache Kafka 组件委派权限到流

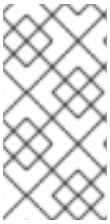
Cluster Operator 在名为 strimzi-cluster-operator 的服务帐户下运行。它被分配集群角色，授予为 Apache Kafka 组件创建流的 RBAC 资源的权限。角色绑定将集群角色与服务帐户关联。

OpenShift 会阻止一个 ServiceAccount 下运行组件授予授予 ServiceAccount 没有的另一个 ServiceAccount 权限。因为 Cluster Operator 会创建其管理的资源所需的 RoleBinding 和 ClusterRoleBinding RBAC 资源，所以需要一个赋予同一特权的角色。

以下小节描述了 Cluster Operator 所需的 RBAC 资源。

1.2.2.2. ClusterRole 资源

Cluster Operator 使用 ClusterRole 资源来提供资源所需的访问权限。根据 OpenShift 集群设置，可能需要集群管理员来创建集群角色。



注意

只有创建 ClusterRole 资源时才需要集群管理员权限。Cluster Operator 不会在集群管理员帐户下运行。

RBAC 资源遵循 *最小特权原则*，仅包含 Cluster Operator 运行 Kafka 组件集群所需的权限。

Cluster Operator 需要所有集群角色才能委派权限。

表 1.3. ClusterRole 资源

Name	描述
strimzi-cluster-operator-namespaced	Cluster Operator 用来部署和管理操作对象的命名空间范围的资源的访问权限。
strimzi-cluster-operator-global	Cluster Operator 用来部署和管理操作对象的资源的访问权限。
strimzi-cluster-operator-leader-election	Cluster Operator 用来进行领导选举机制的访问权限。
strimzi-cluster-operator-watched	Cluster Operator 用来监控和管理 Apache Kafka 自定义资源的 Streams 的访问权限。

Name	描述
strimzi-kafka-broker	在使用机架感知时，访问允许 Kafka 代理从 OpenShift worker 节点获取拓扑标签的权限。
strimzi-entity-operator	主题和用户 Operator 使用的访问权限来管理 Kafka 用户和主题。
strimzi-kafka-client	在使用机架感知时，访问允许 Kafka Connect、MirrorMaker (1 和 2)和 Kafka Bridge 的权限，以从 OpenShift worker 节点获取拓扑标签。

1.2.2.3. ClusterRoleBinding 资源

Cluster Operator 使用 ClusterRoleBinding 和 RoleBinding 资源将其 ClusterRole 与 ServiceAccount 关联。包含集群范围资源的集群角色需要集群角色绑定。

表 1.4. ClusterRoleBinding 资源

Name	描述
strimzi-cluster-operator	为 Cluster Operator 授予来自 **strimzi-cluster-operator-global** 集群角色的权限。
strimzi-cluster-operator-kafka-broker-delegation	为 Cluster Operator 授予 **strimzi-entity-operator** 集群角色的权限。
strimzi-cluster-operator-kafka-client-delegation	授予 Cluster Operator 来自 **strimzi-kafka-client** 集群角色的权限。

表 1.5. RoleBinding 资源

Name	描述
strimzi-cluster-operator	为 Cluster Operator 授予来自 **strimzi-cluster-operator-namespaced** 集群角色的权限。
strimzi-cluster-operator-leader-election	为 Cluster Operator 授予来自 **strimzi-cluster-operator-leader-election** 集群角色的权限。
strimzi-cluster-operator-watched	为 Cluster Operator 授予 **strimzi-cluster-operator-watched** 集群角色的权限。
strimzi-cluster-operator-entity-operator-delegation	为 Cluster Operator 授予 **strimzi-cluster-operator-entity-operator-delegation** 集群角色的权限。

1.2.2.4. ServiceAccount 资源

Cluster Operator 使用 strimzi-cluster-operator ServiceAccount 运行。此服务帐户授予其管理操作对象所需的权限。Cluster Operator 创建额外的 ClusterRoleBinding 和 RoleBinding 资源，以将其中一些 RBAC 权限委派给操作对象。

每个操作对象都使用 Cluster Operator 创建自己的服务帐户。这允许 Cluster Operator 遵循最小特权原则，并只授予操作对象只包括实际需要的访问权限。

表 1.6. ServiceAccount 资源

名称	使用者
<cluster_name>-zookeeper	ZooKeeper pod
<cluster_name>-kafka	Kafka 代理 pod
<cluster_name>-entity-operator	Entity Operator
<cluster_name>-cruise-control	Cruise Control pod
<cluster_name>-kafka-exporter	Kafka Exporter pod
<cluster_name>-connect	Kafka Connect pod
<cluster_name>-mirror-maker	MirrorMaker Pod
<cluster_name>-mirrormaker2	MirrorMaker 2 pod
<cluster_name>-bridge	Kafka Bridge pod

1.2.3. 管理 pod 资源

StrimziPodSet 自定义资源供 Apache Kafka 用于创建和管理 Kafka、Kafka Connect 和 MirrorMaker 2 pod。如果使用 ZooKeeper，则也会使用 StrimziPodSet 资源创建和管理 ZooKeeper pod。

您不能创建、更新或删除 StrimziPodSet 资源。StrimziPodSet 自定义资源在内部使用，资源由 Cluster Operator 单独管理。因此，Cluster Operator 必须正确运行，以避免 pod 无法启动，并且 Kafka 集群不可用。

注意

OpenShift Deployment 资源用于创建和管理其他组件的 pod：Kafka Bridge、Kafka Exporter、Cruise Control，（已弃用） MirrorMaker 1、User Operator 和 Topic Operator。

1.3. 使用 KAFKA BRIDGE 与 KAFKA 集群连接

您可以使用 Apache Kafka Bridge API 的 Streams 来创建和管理消费者，并通过 HTTP 而不是原生 Kafka 协议发送和接收记录。

设置 Kafka Bridge 时，您可以配置对 Kafka 集群的 HTTP 访问。然后，您可以使用 Kafka Bridge 来生成和消费来自集群的消息，以及通过其 REST 接口执行其他操作。

其他资源

- 有关安装和使用 Kafka Bridge 的详情，请参考使用 Apache Kafka Bridge 的 Streams。

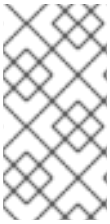
1.4. 无缝 FIPS 支持

联邦信息处理标准(FIPS)是计算机安全和互操作性的标准。当在启用了 FIPS 的 OpenShift 集群上运行 Apache Kafka 的 Streams 时，Apache Kafka 容器镜像流中使用的 OpenJDK 会自动切换到 FIPS 模式。从 2.3 版本，Apache Kafka 的 Streams 可以在启用了 FIPS 的 OpenShift 集群上运行，而无需任何更改或特殊配置。它只使用 OpenJDK 中的 FIPS 兼容安全库。

重要

如果使用启用了 FIPS 的 OpenShift 集群，与常规 OpenShift 集群相比，您可能会遇到更高的内存消耗。为了避免任何问题，我们建议将内存请求增加到至少 512Mi。

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



注意

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

1.4.1. 最小密码长度

在 FIPS 模式下运行时，SCRAM-SHA-512 密码至少需要 32 个字符。在 Apache Kafka 2.3 的 Streams 中，Apache Kafka User Operator 的 Streams 中的默认密码长度也被设置为 32 个字符。如果您的 Kafka 集群带有使用小于 32 个字符的密码长度的自定义配置，则需要更新您的配置。如果您有任何密码少于 32 个字符的用户，则需要重新生成具有所需长度的密码。例如，您可以删除用户 secret 并等待 User Operator 创建具有适当长度的新密码。

其他资源

- 使用 Cluster Operator 配置禁用 FIPS 模式
- 什么是联邦信息处理标准(FIPS)

1.5. 文档约定

user-replaced 值

用户替换的值（也称为 *可替换值* ）以尖括号(< >)一同显示。下划线(_)用于多词语值。如果值引用代码或命令，也使用 monospace。

例如，以下代码显示 &lt ;*my_namespace* > 必须替换为正确的命名空间名称：

```
sed -i 's/namespace: .*/namespace: <my_namespace>/' install/cluster-operator/*RoleBinding*.yaml
```

1.6. 其他资源

- Apache Kafka 概述
- Apache Kafka 自定义资源 API 参考流
- 使用 Apache Kafka Bridge 的流

第 2 章 APACHE KAFKA 安装方法流

您可以通过两种方式将 OpenShift 4.12 到 4.15 上的 Apache Kafka 安装流。

安装方法	Description
安装工件 (YAML 文件)	从 Streams for Apache Kafka 软件下载页面 下载 *Red Hat Streams for Apache Kafka 2.7 OpenShift Installation and Example Files*。使用 **oc** 将 YAML 安装工件部署到 OpenShift 集群。首先，将 Cluster Operator 从 **install/cluster-operator** 部署到单个命名空间、多个命名空间或所有命名空间。 您还可以使用 **install/** 工件来部署以下内容： • Apache Kafka 管理员角色流(**strimzi-admin**) • 独立主题 Operator (**topic-operator**) • 独立用户 Operator (**user-operator**) • Apache Kafka Drain Cleaner 的流(**drain-cleaner**)
OperatorHub	使用 OperatorHub 中的 **Apache Kafka** operator 的 Streams 将 Apache Kafka 的 Streams 部署到单个命名空间或所有命名空间。

要获得最大的灵活性，请选择安装工件方法。OperatorHub 方法提供标准配置，并允许您利用自动更新。

注意

不支持使用 Helm 安装 Apache Kafka 的 Streams。

第 3 章 使用 APACHE KAFKA 的 STREAMS 部署的内容

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

使用 Kafka 组件的典型部署可能包括：

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

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

3.1. 部署顺序

部署到 OpenShift 集群所需的顺序如下：

1. 部署 Cluster Operator 以管理 Kafka 集群

2. 使用 ZooKeeper 集群部署 Kafka 集群，并在部署中包含主题 Operator 和 User Operator

3. （可选）部署：

   - 如果没有使用 Kafka 集群部署主题 Operator 和用户 Operator 独立
   - Kafka Connect
   - Kafka MirrorMaker
   - Kafka Bridge
   - 用于监控指标的组件

Cluster Operator 为组件创建 OpenShift 资源，如 Deployment、Service 和 Pod 资源。OpenShift 资源的名称附加在部署时为组件指定的名称。例如，名为 my-kafka-cluster 的 Kafka 集群有一个名为 my-kafka-cluster-kafka 的服务。

3.2. （预览）为 APACHE KAFKA 代理部署流

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

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

注意

Apache Kafka 代理的流当前作为技术预览提供。

3.3. （预览）为 APACHE KAFKA 控制台部署流

部署由 Streams for Apache Kafka 管理的 Kafka 集群后，您可以部署 Apache Kafka 控制台的 Streams 并连接集群。Apache Kafka 控制台的 Streams 有助于管理 Kafka 集群，通过其用户界面为监控、管理和优化每个集群提供实时见解。

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

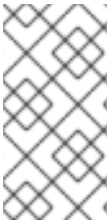
注意

Apache Kafka 控制台的流当前作为技术预览提供。

第 4 章 为 APACHE KAFKA 部署准备您的流

通过完成任何必要的部署前任务，准备 Apache Kafka 部署流。根据您的具体要求执行必要的准备步骤，如下所示：

- 在为 Apache Kafka 部署流前，请确保您有必要的先决条件
- 下载 Apache Kafka 发行版本工件的 Streams，以便于您的部署
- 将 Apache Kafka 容器镜像的流推送到您自己的 registry 中（如果需要）
- 设置 admin 角色以启用部署中使用的自定义资源的配置



注意

要在本指南中运行命令，您的集群用户必须具有管理基于角色的访问控制(RBAC)和 CRD 的权限。

4.1. 部署先决条件

要为 Apache Kafka 部署流，您需要以下内容：

- OpenShift 4.12 到 4.15 集群。

  Apache Kafka 的流基于 Strimzi 0.40.x。
- oc 命令行工具已安装并配置为连接到正在运行的集群。

4.2. OPERATOR 部署最佳实践

同一 OpenShift 集群中安装多个流 for Apache Kafka operator 可能会出现潜在的问题，特别是在使用不同的版本时。每个 Apache Kafka operator Streams 都会管理 OpenShift 集群中的一组资源。当您为 Apache Kafka operator 安装多个流时，它们可能会尝试同时管理同一资源。这会导致集群中的冲突和无法预计的行为。即使在同一 OpenShift 集群中不同命名空间中为 Apache Kafka operator 部署流，

也会发生冲突。虽然命名空间提供一定程度的资源隔离，但由 Apache Kafka operator 的 Streams 管理的某些资源（如自定义资源定义(CRD)和角色）具有集群范围的范围。

另外，安装具有不同版本的多个 Operator 可能会导致 Operator 和它们管理的 Kafka 集群间的兼容性问题。Apache Kafka operator 的不同流版本可能会引入不向后兼容的更改、错误修复或改进。

为了避免在 OpenShift 集群中为 Apache Kafka Operator 安装多个流相关的问题，建议遵循以下准则：

- 在与 Kafka 集群和其他 Kafka 组件独立的命名空间中安装 Apache Kafka operator 的 Streams，以确保明确隔离资源和配置。
- 使用单个 Streams for Apache Kafka operator 管理 OpenShift 集群中的所有 Kafka 实例。
- 更新 Apache Kafka operator 的 Streams 以及支持的 Kafka 版本，以反映最新的功能和增强。

通过遵循这些最佳实践并确保对 Apache Kafka operator 的单一流的一致更新，您可以增强在 OpenShift 集群中管理 Kafka 实例的稳定性。这个方法还允许您充分利用 Apache Kafka 的最新功能和功能。

注意

随着 Apache Kafka 的 Streams 基于 Strimzi，在将 Apache Kafka operator 与 OpenShift 集群中的 Strimzi operator 合并时，可能会出现相同的问题。

4.3. 下载 APACHE KAFKA 发行版本工件的流

要使用部署文件为 Apache Kafka 安装流，请从 Apache Kafka 软件下载页面 的 Streams 下载文件。

Apache Kafka 发行工件的流包括示例 YAML 文件，可帮助您将 Apache Kafka 的 Streams 组件部署到 OpenShift，执行常见操作并配置 Kafka 集群。

使用 oc 从下载的 ZIP 文件的 install/cluster-operator 文件夹部署 Cluster Operator。有关部署和配置 Cluster Operator 的更多信息，请参阅 第 6.2 节 “部署 Cluster Operator”。

另外，如果要使用由 Apache Kafka Cluster Operator 的 Streams 管理的主题和用户 Operator 的独立安装，您可以从 install/topic-operator 和 install/user-operator 文件夹部署它们。

注意

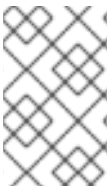
Apache Kafka 容器镜像流也可以通过 红帽生态系统目录 获得。但是，我们建议您使用为 Apache Kafka 部署流提供的 YAML 文件。

4.4. 将容器镜像推送到您自己的 REGISTRY 中

Red Hat Ecosystem Catalog 包括了用于 Apache Kafka 的流的容器镜像。由 Apache Kafka 的 Streams 提供的安装 YAML 文件将直接 从红帽生态系统目录 拉取镜像。

如果您无法访问 红帽生态系统目录 或想要使用您自己的容器存储库，请执行以下操作：

1. 拉取此处列出的 所有容器镜像
2. 将它们推送到您自己的 registry 中
3. 更新安装 YAML 文件中的镜像名称



注意

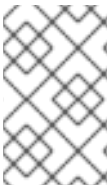
发行版本支持的每个 Kafka 版本均有单独的镜像。

容器镜像	namespace/Repository	Description

容器镜像	namespace/Repository	Description
Kafka	• registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0 • registry.redhat.io/amq-streams/kafka-36-rhel9:2.7.0	运行 Kafka 的 Apache Kafka 镜像流，包括： • Kafka Broker • Kafka Connect • Kafka MirrorMaker • ZooKeeper • TLS Sidecars • Sything Control
Operator	• registry.redhat.io/amq-streams/strimzi-rhel9-operator:2.7.0	运行 Operator 的 Apache Kafka 镜像流： • Cluster Operator • Topic Operator • User Operator • Kafka Initializer
Kafka Bridge	• registry.redhat.io/amq-streams/bridge-rhel9:2.7.0	用于运行 Apache Kafka Bridge 的 Streams 的 Apache Kafka 镜像流
Apache Kafka Drain Cleaner 的流	• registry.redhat.io/amq-streams/drain-cleaner-rhel9:2.7.0	用于运行 Apache Kafka Drain Cleaner 的 Streams 的 Apache Kafka 镜像的流
Apache Kafka 代理流	• registry.redhat.io/amq-streams/proxy-rhel9-operator:2.7.0	用于运行 Apache Kafka 代理的 Streams 的 Apache Kafka 镜像
Apache Kafka 控制台流	• registry.redhat.io/amq-streams/console-rhel9-operator:2.7.0	运行 Apache Kafka 控制台的流的 Apache Kafka 镜像

4.5. 创建用于向容器镜像 REGISTRY 进行身份验证的 PULL SECRET

由 Apache Kafka 的 Streams 提供的安装 YAML 文件直接 从红帽生态系统目录拉取容器镜像。如果 Apache Kafka 部署的流需要身份验证，请在 secret 中配置身份验证凭据并将其添加到安装 YAML 中。



注意

通常不需要身份验证，但可能会在某些平台上请求。

先决条件

- 您需要红帽用户名和密码，或者来自 Red Hat registry 服务帐户的登录详情。

注意

您可以使用您的红帽订阅 从红帽客户门户网站 创建 registry 服务帐户。

流程

1. 创建包含登录详情和从其中拉取 Apache Kafka 镜像的容器 registry 的 pull secret：

```
oc create secret docker-registry <pull_secret_name> \
  --docker-server=registry.redhat.io \
  --docker-username=<user_name> \
  --docker-password=<password> \
  --docker-email=<email>
```

添加您的用户名和密码。电子邮件地址是可选的。

2. 编辑 install/cluster-operator/060-Deployment-strimzi-cluster-operator.yaml 部署文件，以使用 STRIMZI_IMAGE_PULL_SECRETS 环境变量指定 pull secret：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-cluster-operator
spec:
  # ...
  template:
    spec:
      serviceAccountName: strimzi-cluster-operator
      containers:
        # ...
        env:
```

```
        - name: STRIMZI_IMAGE_PULL_SECRETS
          value: "<pull_secret_name>"
# ...
```

secret 适用于 Cluster Operator 创建的所有 pod。

4.6. 为 APACHE KAFKA 管理员设计流

Apache Kafka 的流提供用于配置部署的自定义资源。默认情况下，查看、创建、编辑和删除这些资源的权限仅限于 OpenShift 集群管理员。Apache Kafka 的流提供了两个集群角色，您可以使用它们为其他用户分配这些权限：

- strimzi-view 允许用户查看和列出 Apache Kafka 资源的流。
- strimzi-admin 允许用户为 Apache Kafka 资源创建、编辑或删除流。

安装这些角色时，它们将自动将这些权限聚合（添加）到默认的 OpenShift 集群角色。strimzi-view 聚合到 view 角色，strimzi-admin 聚合到 edit 和 admin 角色。由于聚合，您可能不需要将这些角色分配给已具有类似权利的用户。

以下流程演示了如何分配 strimzi-admin 角色，允许非集群管理员管理 Apache Kafka 资源的 Streams。

系统管理员可在部署 Cluster Operator 后为 Apache Kafka 管理员指定流。

先决条件

- 使用 Cluster Operator 部署 Apache Kafka 自定义资源定义(CRD)和基于角色的访问控制(RBAC)资源的 流。

流程

1. 在 OpenShift 中创建 strimzi-view 和 strimzi-admin 集群角色。

```
oc create -f install/strimzi-admin
```

2. 如果需要，为需要它们的用户分配提供访问权限的角色。

```
oc create clusterrolebinding strimzi-admin --clusterrole=strimzi-admin --user=user1 --
user=user2
```

第 5 章 使用 WEB 控制台从 OPERATORHUB 安装 APACHE KAFKA 的流

在 OpenShift Container Platform Web 控制台中，从 OperatorHub 安装 Streams for Apache Kafka operator。

本节中的步骤演示了如何：

- 从 OperatorHub 安装 Apache Kafka operator 的 Streams
- 使用 Apache Kafka operator 的 Streams 部署 Kafka 组件

5.1. 从 OPERATORHUB 安装 APACHE KAFKA OPERATOR 的 STREAMS

您可以使用 OpenShift Container Platform Web 控制台中的 OperatorHub 安装并订阅 Apache Kafka operator 的 Streams。

此流程描述了如何创建项目，并将 Apache Kafka operator 的 Streams 安装到该项目中。项目是命名空间的表示。对于可管理性，最好使用命名空间来分隔功能。

警告

确保使用正确的更新频道。如果您在受支持的 OpenShift 版本中，从默认 stable 频道安装 Apache Kafka 的 Streams 通常是安全的。但是，我们不推荐在 stable 频道中启用自动更新。自动升级将在升级前跳过所有必要的步骤。仅在特定于版本的频道中使用自动升级。

先决条件

- 使用具有 cluster-admin 或 strimzi-admin 权限的账户访问 OpenShift Container Platform Web 控制台。

流程

1.

在 OpenShift Web 控制台中进入到 Home > Projects 页面，再创建一个用于安装的项目（命名空间）。

在这个示例中，我们使用名为 amq-streams-kafka 的项目。

2. 进入 Operators > OperatorHub 页面。

3. 在 Filter by keyword 框中滚动或输入关键字以查找 Apache Kafka operator 的 Streams。

operator 位于 Streaming 和 Messaging 目录中。

4. 点 Streams for Apache Kafka 来显示 Operator 信息。

5. 阅读有关 Operator 的信息，再点 Install。

6. 在 Install Operator 页面中，从以下安装和更新选项中选择：

- 更新频道 ：选择 Operator 的更新频道。
  - （默认） stable 频道包含所有最新的更新和发行版本，包括主版本、次版本和微版本，这些版本被认为经过充分测试和稳定。
  - amq-streams-*X*.x 频道包含主发行版本的次要和微版本更新，其中 *X* 是主版本的版本号。
  - amq-streams-*X*.*Y*.x 频道包含次要发行本版本的微版本更新，其中 *X* 是主版本的版本号，*Y* 是次版本号。
- Installation Mode ：选择您创建的项目，以便在特定命名空间中安装 Operator。

您可以将 Apache Kafka operator 的 Streams 安装到集群中的所有命名空间（默认选项）或特定命名空间中。我们建议您将特定命名空间专用于 Kafka 集群和其他 Streams for Apache Kafka 组件。

- 更新批准 ：默认情况下，Apache Kafka operator 的 Streams 由 Operator Lifecycle Manager (OLM)自动升级到 Apache Kafka 版本的最新流。另外，如果您希望手动批准将来的升级，请选择 Manual。如需有关 operator 的更多信息，请参阅 OpenShift 文档。

7. 点 Install 将 Operator 安装到所选命名空间中。

Apache Kafka operator 的 Streams 将 Cluster Operator、CRD 和基于角色的访问控制 (RBAC)资源部署到所选命名空间中。

8. Operator 就绪可用后，进入 Operators > Installed Operators 来验证 Operator 是否已安装到所选命名空间中。

状态将显示为 Succeeded。

现在，您可以使用 Apache Kafka operator 的 Streams 来部署 Kafka 组件，从 Kafka 集群开始。

注意

如果您进入到 Workloads > Deployments，您可以查看 Cluster Operator 和 Entity Operator 的部署详情。Cluster Operator 的名称包含一个版本号：amq-streams-cluster-operator-<version>。当使用 Streams for Apache Kafka 安装工件部署 Cluster Operator 时，名称会有所不同。在本例中，名称是 strimzi-cluster-operator。

5.2. 使用 APACHE KAFKA OPERATOR 的 STREAMS 部署 KAFKA 组件

在 Openshift 上安装时，Apache Kafka operator 的 Streams 使 Kafka 组件可从用户界面安装。

以下 Kafka 组件可用于安装：

- Kafka
- Kafka Connect

- Kafka MirrorMaker
- Kafka MirrorMaker 2
- Kafka 主题
- Kafka 用户
- Kafka Bridge
- Kafka Connector
- Kafka Rebalance

您可以选择组件并创建实例。您至少创建一个 Kafka 实例。这个步骤描述了如何使用默认设置创建 Kafka 实例。您可以在执行安装前配置默认安装规格。

创建其他 Kafka 组件实例的过程相同。

先决条件

- Apache Kafka operator 的 Streams 安装在 OpenShift 集群中。

流程

1. 在 Web 控制台中进入 Operators &gt ; Installed Operators 页面，点 Streams for Apache Kafka 来显示 Operator 详情。

   在 Provided APIs 中，您可以创建 Kafka 组件的实例。

2. 点 Kafka 下的 Create instance 创建 Kafka 实例。

默认情况下，您将创建一个名为 my-cluster 的 Kafka 集群，它有三个 Kafka 代理节点和三个 ZooKeeper 节点。集群使用临时存储。

3. 点 Create 开始安装 Kafka。

等待状态变为 Ready。

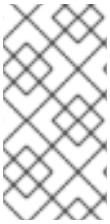
第 6 章 使用安装工件为 APACHE KAFKA 部署流

为 Apache Kafka 部署流准备了环境，您可以将 Apache Kafka 的流部署到 OpenShift 集群中。使用由发行工件提供的安装文件。

Apache Kafka 的流基于 Strimzi 0.40.x。您可以将 Apache Kafka 2.7 上的流部署到 OpenShift 4.12 到 4.15。

使用安装文件为 Apache Kafka 部署流的步骤如下：

1. 部署 Cluster Operator
2. 使用 Cluster Operator 部署以下内容：
    a. Kafka 集群
    b. Topic Operator
    c. User Operator
3. 另外，还可根据要求部署以下 Kafka 组件：
    - Kafka Connect
    - Kafka MirrorMaker
    - Kafka Bridge



注意

若要在本指南中运行命令，OpenShift 用户必须具有管理基于角色的访问控制(RBAC)和 CRD 的权限。

6.1. 基本部署路径

您可以设置一个部署，其中 Apache Kafka 的 Streams 管理同一命名空间中的单个 Kafka 集群。您可以使用此配置进行开发或测试。或者在生产环境中，您可以使用 Apache Kafka 的 Streams 来管理不同命名空间中的多个 Kafka 集群。

任何为 Apache Kafka 部署流的第一步是使用 install/cluster-operator 文件安装 Cluster Operator。

单个命令应用 cluster-operator 文件夹中的所有安装文件： oc apply -f ./install/cluster-operator。

该命令设置您可以创建和管理 Kafka 部署所需的所有内容，包括：

- Cluster Operator (部署、ConfigMap)
- Apache Kafka CRD 的流(CustomResourceDefinition)
- RBAC 资源(ClusterRole、ClusterRoleBinding、RoleBinding)
- 服务帐户(ServiceAccount)

基本部署路径如下：

1. 下载发行工件
2. 创建用于部署 Cluster Operator 的 OpenShift 命名空间

3. 部署 Cluster Operator

   a. 更新 install/cluster-operator 文件，以使用为 Cluster Operator 创建的命名空间

   b. 安装 Cluster Operator 以监视一个或多个命名空间

4. 创建 Kafka 集群

之后，您可以部署其他 Kafka 组件并设置部署的监控。

6.2. 部署 CLUSTER OPERATOR

Cluster Operator 负责在 OpenShift 集群中部署和管理 Kafka 集群。

当 Cluster Operator 运行时，它会开始监视 Kafka 资源的更新。

默认情况下，会部署 Cluster Operator 的单个副本。您可以使用领导选举机制添加副本，以便在出现问题时有其他 Cluster Operator 处于待机状态。如需更多信息，请参阅 第 9.5.4 节 “使用领导选举机制运行多个 Cluster Operator 副本”。

6.2.1. 指定 Cluster Operator 监视的命名空间

Cluster Operator 监视部署了 Kafka 资源的命名空间中是否有更新。部署 Cluster Operator 时，您可以指定要在 OpenShift 集群中监视的命名空间。您可以指定以下命名空间：

- 单个所选命名空间 （包含 Cluster Operator 的同一命名空间）
- 多个所选命名空间
- 集群中的所有命名空间

监视多个所选命名空间会因为增加处理开销而对性能有影响。要优化命名空间监控的性能，通常建议监视单个命名空间或监控整个集群。监控单个命名空间允许集中监控特定于命名空间的资源，而监控所有命名空间则提供所有命名空间中集群资源的全面视图。

Cluster Operator 会监视以下资源的更改：

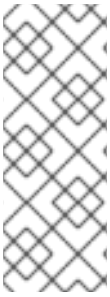
- Kafka 集群的 Kafka。
- Kafka Connect 集群的 KafkaConnect。
- 在 Kafka Connect 集群中创建和管理连接器的 KafkaConnector。
- Kafka MirrorMaker 实例的 KafkaMirrorMaker。
- Kafka MirrorMaker 2 实例的 KafkaMirrorMaker2。
- KafkaBridge 用于 Kafka Bridge 实例。
- Cruise Control 优化请求的 KafkaRebalance。

当在 OpenShift 集群中创建其中一个资源时，Operator 会从资源获取集群描述，并通过创建必要的 OpenShift 资源（如 Deployments、Pod、服务和 ConfigMap）开始为资源创建新集群。

每次更新 Kafka 资源时，Operator 会在为资源组成集群的 OpenShift 资源上执行对应的更新。

资源可以被修补或删除，然后重新创建资源以便使集群反映集群的所需状态。此操作可能会导致滚动更新导致服务中断。

删除资源时，操作器会取消部署集群并删除所有相关 OpenShift 资源。



注意

虽然 Cluster Operator 可以监控 OpenShift 集群中的一个、多个或所有命名空间，但主题 Operator 和 User Operator 会监视单个命名空间中的 KafkaTopic 和 KafkaUser 资源。如需更多信息，请参阅 第 1.2.1 节 “在 OpenShift 命名空间中监视 Apache Kafka 资源的流”。

6.2.2. 部署 Cluster Operator 以观察单个命名空间

此流程演示了如何部署 Cluster Operator，以观察 OpenShift 集群中单个命名空间中的 Apache Kafka 资源的 Streams。

先决条件

- 您需要具有创建和管理 CustomResourceDefinition 和 RBAC (ClusterRole、RoleBinding和 RoleBinding)资源的权限的帐户。

流程

1. 编辑 Apache Kafka 安装文件的 Streams，以使用 Cluster Operator 将安装到的命名空间。

   例如，在这个流程中，Cluster Operator 已安装到命名空间 my-cluster-operator-namespace 中。

   在 Linux 中，使用：

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

   对于 MacOS，使用：

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

2. 部署 Cluster Operator：

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

3. 检查部署的状态：

```
oc get deployments -n my-cluster-operator-namespace
```

输出显示部署名称和就绪

```
NAME                      READY  UP-TO-DATE  AVAILABLE
strimzi-cluster-operator  1/1    1           1
```

READY 显示就绪/预期的副本数。当 AVAILABLE 输出显示为 1 时，部署成功。

6.2.3. 部署 Cluster Operator 以观察多个命名空间

此流程演示了如何部署 Cluster Operator，以观察 OpenShift 集群中多个命名空间中的 Apache Kafka 资源的流。

先决条件

- 您需要具有创建和管理 CustomResourceDefinition 和 RBAC (ClusterRole、RoleBinding和 RoleBinding)资源的权限的帐户。

流程

1. 编辑 Apache Kafka 安装文件的 Streams，以使用 Cluster Operator 将安装到的命名空间。

   例如，在这个流程中，Cluster Operator 已安装到命名空间 my-cluster-operator-namespace 中。

   在 Linux 中，使用：

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

对于 MacOS，使用：

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

2. 编辑 install/cluster-operator/060-Deployment-strimzi-cluster-operator.yaml 文件，以添加 Cluster Operator 将监视到 STRIMZI_NAMESPACE 环境变量的所有命名空间的列表。

例如，在这个流程中，Cluster Operator 将监视 watched-namespace-1,watched-namespace-2, watched-namespace-3。

```
apiVersion: apps/v1
kind: Deployment
spec:
  # ...
  template:
    spec:
      serviceAccountName: strimzi-cluster-operator
      containers:
      - name: strimzi-cluster-operator
        image: registry.redhat.io/amq-streams/strimzi-rhel9-operator:2.7.0
        imagePullPolicy: IfNotPresent
        env:
        - name: STRIMZI_NAMESPACE
          value: watched-namespace-1,watched-namespace-2,watched-namespace-3
```

3. 对于列出的每个命名空间，安装 RoleBindings。

在这个示例中，将这些命令的 *watched-namespace* 替换为在前一步中列出的命名空间，为 watched-namespace-1, watched-namespace-2, watched-namespace-3 重复这个操作：

```
oc create -f install/cluster-operator/020-RoleBinding-strimzi-cluster-operator.yaml -n <watched_namespace>
oc create -f install/cluster-operator/023-RoleBinding-strimzi-cluster-operator.yaml -n <watched_namespace>
oc create -f install/cluster-operator/031-RoleBinding-strimzi-cluster-operator-entity-operator-delegation.yaml -n <watched_namespace>
```

4. 部署 Cluster Operator：

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

-

5. 检查部署的状态：

```
oc get deployments -n my-cluster-operator-namespace
```

输出显示部署名称和就绪

```
NAME                      READY  UP-TO-DATE  AVAILABLE
strimzi-cluster-operator  1/1    1           1
```

READY 显示就绪/预期的副本数。当 AVAILABLE 输出显示为 1 时，部署成功。

6.2.4. 部署 Cluster Operator 以监视所有命名空间

此流程演示了如何部署 Cluster Operator，以观察 OpenShift 集群中的所有命名空间中的 Apache Kafka 资源的 Streams。

在这种模式下运行时，Cluster Operator 会在创建的任何新命名空间中自动管理集群。

先决条件

- 您需要具有创建和管理 CustomResourceDefinition 和 RBAC (ClusterRole、RoleBinding和 RoleBinding)资源的权限的帐户。

流程

1. 编辑 Apache Kafka 安装文件的 Streams，以使用 Cluster Operator 将安装到的命名空间。

   例如，在这个流程中，Cluster Operator 已安装到命名空间 my-cluster-operator-namespace 中。

   在 Linux 中，使用：

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

对于 **MacOS，使用：**

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

2. 编辑 **install/cluster-operator/060-Deployment-strimzi-cluster-operator.yaml 文件，将 STRIMZI_NAMESPACE** 环境变量的值设置为 ***。**

```
apiVersion: apps/v1
kind: Deployment
spec:
  # ...
  template:
    spec:
      # ...
      serviceAccountName: strimzi-cluster-operator
      containers:
      - name: strimzi-cluster-operator
        image: registry.redhat.io/amq-streams/strimzi-rhel9-operator:2.7.0
        imagePullPolicy: IfNotPresent
        env:
        - name: STRIMZI_NAMESPACE
          value: "*"
        # ...
```

3. 创建 **ClusterRoleBindings，将所有命名空间的集群范围访问权限授予 Cluster Operator。**

```
oc create clusterrolebinding strimzi-cluster-operator-namespaced --clusterrole=strimzi-cluster-operator-namespaced --serviceaccount my-cluster-operator-namespace:strimzi-cluster-operator
oc create clusterrolebinding strimzi-cluster-operator-watched --clusterrole=strimzi-cluster-operator-watched --serviceaccount my-cluster-operator-namespace:strimzi-cluster-operator
oc create clusterrolebinding strimzi-cluster-operator-entity-operator-delegation --clusterrole=strimzi-entity-operator --serviceaccount my-cluster-operator-namespace:strimzi-cluster-operator
```

4. **将 Cluster Operator 部署到 OpenShift 集群。**

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

-

5. 检查部署的状态：

```
oc get deployments -n my-cluster-operator-namespace
```

输出显示部署名称和就绪

```
NAME                       READY  UP-TO-DATE  AVAILABLE
strimzi-cluster-operator  1/1    1           1
```

READY 显示就绪/预期的副本数。当 AVAILABLE 输出显示为 1 时，部署成功。

6.3. 部署 KAFKA

要使用 Cluster Operator 管理 Kafka 集群，您必须将它部署为 Kafka 资源。Apache Kafka 的流提供了示例部署文件来执行此操作。您可以使用这些文件同时部署主题 Operator 和 User Operator。

部署 Cluster Operator 后，使用 Kafka 资源来部署以下组件：

- 使用 KRaft 或 ZooKeeper 的 Kafka 集群：
  - 带有节点池的 KRaft 或基于 ZooKeeper 的 Kafka 集群
  - 在没有节点池的情况下基于 ZooKeeper 的 Kafka 集群
- Topic Operator
- User Operator

节点池为一组 Kafka 节点提供配置。通过使用节点池，节点可以在同一 Kafka 集群中有不同的配置。

如果您没有将 Kafka 集群部署为 Kafka 资源，则无法使用 Cluster Operator 管理它。例如，这适用于在 OpenShift 外部运行的 Kafka 集群。但是，您可以将主题 Operator 和 User Operator 与 不是由 Apache Kafka 管理的 Kafka 集群一起使用，方法是将 它们部署为独立组件。您还可以在不是由 Apache Kafka 的 Streams 管理的 Kafka 集群中部署和使用其他 Kafka 组件。

6.3.1. 使用节点池部署 Kafka 集群

此流程演示了如何使用 Cluster Operator 将带有节点池的 Kafka 部署到 OpenShift 集群。节点池代表 Kafka 集群中共享相同配置的 Kafka 节点组。对于节点池中的每个 Kafka 节点，节点池中没有定义的任何配置都会继承 kafka 资源中的集群配置。

部署使用 YAML 文件来提供规格来创建 KafkaNodePool 资源。您可以将节点池与 Kafka 集群一起使用，该集群使用 KRaft (Kafka Raft metadata)模式或 ZooKeeper 进行集群管理。要以 KRaft 模式部署 Kafka 集群，您必须使用 KafkaNodePool 资源。

Apache Kafka 的 Streams 提供了以下 示例文件，您可以使用它来创建使用节点池的 Kafka 集群：

kafka-with-dual-role-kraft-nodes.yaml

使用共享代理和控制器角色的 KRaft 节点池部署 Kafka 集群。

kafka-with-kraft.yaml

部署具有一个控制器节点池的持久 Kafka 集群，以及一个代理节点池。

kafka-with-kraft-ephemeral.yaml

部署一个临时 Kafka 集群，它具有一个控制器节点池和一个代理节点池。

kafka.yaml

使用 3 个节点和 2 个不同的 Kafka 代理池部署 ZooKeeper。每个池都有 3 个代理。示例中的池使用不同的存储配置。

注意

您可以执行此处概述的步骤，以使用 KafkaNodePool 资源部署新的 Kafka 集群，或迁移现有的 Kafka 集群。

先决条件

- 必须部署 Cluster Operator。

流程

1. 部署基于 KRaft 的 Kafka 集群。

   - 使用使用 dual-role 节点的单一节点池，以 KRaft 模式部署 Kafka 集群：

     ```
     oc apply -f examples/kafka/kraft/kafka-with-dual-role-nodes.yaml
     ```

   - 以 KRaft 模式为代理和控制器节点部署持久的 Kafka 集群：

     ```
     oc apply -f examples/kafka/kraft/kafka.yaml
     ```

   - 以 KRaft 模式为代理和控制器节点部署临时 Kafka 集群：

     ```
     oc apply -f examples/kafka/kraft/kafka-ephemeral.yaml
     ```

   - 使用三个代理的两个节点池部署 Kafka 集群和 ZooKeeper 集群：

     ```
     oc apply -f examples/kafka/kafka-with-node-pools.yaml
     ```

2. 检查部署的状态：

   ```
   oc get pods -n <my_cluster_operator_namespace>
   ```

   输出显示节点池名称和就绪

   ```
   NAME                          READY  STATUS   RESTARTS
   my-cluster-entity-operator  3/3    Running  0
   my-cluster-pool-a-0         1/1    Running  0
   my-cluster-pool-a-1         1/1    Running  0
   my-cluster-pool-a-4         1/1    Running  0
   ```

- my-cluster 是 Kafka 集群的名称。
- pool-a 是节点池的名称。

以 0 开始的连续索引号标识每个创建的 Kafka pod。如果使用 ZooKeeper，您还会看到 ZooKeeper pod。

READY 显示就绪/预期的副本数。当 STATUS 显示为 Running 时，部署成功。

有关部署的信息也会显示在 KafkaNodePool 资源的状态中，包括池中节点的 ID 列表。

注意

节点 ID 按顺序分配自集群中所有节点池中的 0 （零）。这意味着节点 ID 可能无法在特定节点池中按顺序运行。如果集群中节点 ID 序列出现差距，则会为要添加的下一个节点分配一个填充空白的 ID。缩减时，池中具有最高节点 ID 的节点会被删除。

其他资源

节点池配置

6.3.2. 在没有节点池的情况下部署基于 ZooKeeper 的 Kafka 集群

此流程演示了如何使用 Cluster Operator 将基于 ZooKeeper 的 Kafka 集群部署到 OpenShift 集群。

部署使用 YAML 文件来提供规格来创建 Kafka 资源。

Apache Kafka 的 Streams 提供了 以下示例 文件来创建使用 ZooKeeper 进行集群管理的 Kafka 集群：

kafka-persistent.yaml

使用三个 ZooKeeper 和三个 Kafka 节点部署持久集群。

kafka-jbod.yaml

使用三个 ZooKeeper 和三个 Kafka 节点（各自使用多个持久性卷）部署持久集群。

kafka-persistent-single.yaml

使用单个 ZooKeeper 节点和单个 Kafka 节点部署持久性集群。

kafka-ephemeral.yaml

使用三个 ZooKeeper 和三个 Kafka 节点部署临时集群。

kafka-ephemeral-single.yaml

使用三个 ZooKeeper 节点和一个 Kafka 节点部署临时集群。

在此过程中，我们对 ***ephemeral（临时）*** 和 ***persistent（持久）*** Kafka 集群部署使用示例。

临时集群

通常，临时（或临时）Kafka 集群适合开发和测试目的，不适用于生产环境。此部署使用 emptyDir 卷来存储代理信息（用于 ZooKeeper）和主题或分区（用于 Kafka）。使用 emptyDir 卷意味着其内容严格与 pod 生命周期相关，并在 pod 停机时删除。

持久性集群

持久的 Kafka 集群使用持久性卷来存储 ZooKeeper 和 Kafka 数据。使用 PersistentVolumeClaim 获取 PersistentVolume，使其独立于 PersistentVolume 的实际类型。PersistentVolumeClaim 可以使用 StorageClass 触发自动卷置备。如果没有指定 StorageClass，OpenShift 将尝试使用默认的 StorageClass。

以下示例显示了一些常见的持久性卷类型：

- 如果您的 OpenShift 集群在 Amazon AWS 上运行，OpenShift 可以置备 Amazon EBS 卷
- 如果您的 OpenShift 集群在 Microsoft Azure 上运行，OpenShift 可以置备 Azure Disk Storage 卷
- 如果您的 OpenShift 集群在 Google Cloud 上运行，OpenShift 可以置备 Persistent Disk 卷

- 如果您的 OpenShift 集群在裸机上运行，OpenShift 可以置备本地持久性卷

示例 YAML 文件指定最新支持的 Kafka 版本，以及其支持的日志消息格式版本和 inter-broker 协议版本的配置。Kafka config 的 inter.broker.protocol.version 属性必须是指定的 Kafka 版本 (spec.kafka.version) 支持的版本。属性代表 Kafka 集群中使用的 Kafka 协议版本。

从 Kafka 3.0.0，当 inter.broker.protocol.version 设置为 3.0 或更高版本时，log.message.format.version 选项会被忽略，且不需要设置。

默认情况下，示例集群名为 my-cluster。集群名称由资源名称定义，在部署集群后无法更改。要在部署集群前更改集群名称，请编辑相关 YAML 文件中的 Kafka 资源的 Kafka.metadata.name 属性。

默认集群名称和指定的 Kafka 版本

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    version: 3.7.0
    #...
    config:
      #...
      log.message.format.version: "3.7"
      inter.broker.protocol.version: "3.7"
  # ...
```

先决条件

- 必须部署 Cluster Operator。

流程

1. 部署基于 ZooKeeper 的 Kafka 集群。

- 部署临时集群：

```
oc apply -f examples/kafka/kafka-ephemeral.yaml
```

- 部署持久集群：

```
oc apply -f examples/kafka/kafka-persistent.yaml
```

2. 检查部署的状态：

```
oc get pods -n <my_cluster_operator_namespace>
```

输出显示 pod 名称和就绪状态

```
NAME                          READY   STATUS    RESTARTS
my-cluster-entity-operator  3/3     Running   0
my-cluster-kafka-0          1/1     Running   0
my-cluster-kafka-1          1/1     Running   0
my-cluster-kafka-2          1/1     Running   0
my-cluster-zookeeper-0      1/1     Running   0
my-cluster-zookeeper-1      1/1     Running   0
my-cluster-zookeeper-2      1/1     Running   0
```

my-cluster 是 Kafka 集群的名称。

从 0 开始的连续索引号会标识每个 Kafka 和 ZooKeeper pod 创建。

使用默认部署，您可以创建一个 Entity Operator 集群、3 Kafka pod 和 3 ZooKeeper pod。

READY 显示就绪/预期的副本数。当 STATUS 显示为 Running 时，部署成功。

其他资源

Kafka 集群配置

6.3.3. 使用 Cluster Operator 部署主题 Operator

此流程描述了如何使用 Cluster Operator 部署主题 Operator。可以部署主题 Operator 以在双向模式或单向模式中使用。要了解更多有关双向和单向主题管理的信息，请参阅 第 10.1 节 “主题管理模式”。

您可以配置 Kafka 资源的 entityOperator 属性，使其包含 topicOperator。默认情况下，Topic Operator 会监视 Cluster Operator 部署的 Kafka 集群命名空间中的 KafkaTopic 资源。您还可以使用 Topic Operator spec 中的 watchedNamespace 指定一个命名空间。单个主题 Operator 可以监视单个命名空间。一个命名空间应该只由一个 Topic Operator 监视。

如果您使用 Streams for Apache Kafka 将多个 Kafka 集群部署到同一命名空间中，请只为一个 Kafka 集群启用 Topic Operator，或使用 watchedNamespace 属性配置主题 Operator 来监视其他命名空间。

如果要将主题 Operator 与不是由 Apache Kafka 的 Streams 管理的 Kafka 集群搭配使用，您必须将 Topic Operator 部署为独立组件。

有关配置 entityOperator 和 topicOperator 属性的更多信息，请参阅配置 Entity Operator。

先决条件

- 必须部署 Cluster Operator。

流程

1. 编辑 Kafka 资源的 entityOperator 属性，使其包含 topicOperator ：

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

2. 使用 EntityTopicOperatorSpec schema reference 中所述的属性配置 Topic Operator spec。

   如果您希望所有属性都使用其默认值，请使用空对象({})。

3. 创建或更新资源：

```
oc apply -f <kafka_configuration_file>
```

4. 检查部署的状态：

```
oc get pods -n <my_cluster_operator_namespace>
```

输出显示 pod 名称和就绪状态

```
NAME                          READY   STATUS    RESTARTS
my-cluster-entity-operator  3/3     Running   0
# ...
```

   my-cluster 是 Kafka 集群的名称。

   READY 显示就绪/预期的副本数。当 STATUS 显示为 Running 时，部署成功。

6.3.4. 使用 Cluster Operator 部署 User Operator

此流程描述了如何使用 Cluster Operator 部署 User Operator。

您可以配置 Kafka 资源的 entityOperator 属性，使其包含 userOperator。默认情况下，User Operator 会监视 Kafka 集群部署命名空间中的 KafkaUser 资源。您还可以使用 User Operator spec 中的 watchedNamespace 指定命名空间。单个 User Operator 可以监视单个命名空间。一个命名空间应该只由一个 User Operator 监视。

如果要将 User Operator 与不是由 Apache Kafka 的 Streams 管理的 Kafka 集群搭配使用，您必须将 User Operator 部署为独立组件。

有关配置 entityOperator 和 userOperator 属性的更多信息，请参阅配置 Entity Operator。

先决条件

- 必须部署 Cluster Operator。

流程

1. 编辑 Kafka 资源的 entityOperator 属性，使其包含 userOperator ：

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

2. 使用 EntityUserOperatorSpec schema reference 中所述的属性配置 User Operator spec。

   如果您希望所有属性都使用其默认值，请使用空对象({})。

3. 创建或更新资源：

```
oc apply -f <kafka_configuration_file>
```

4. 检查部署的状态：

```
oc get pods -n <my_cluster_operator_namespace>
```

输出显示 pod 名称和就绪状态

-

```
NAME                        READY   STATUS    RESTARTS
my-cluster-entity-operator  3/3     Running   0
# ...
```

my-cluster 是 Kafka 集群的名称。

READY 显示就绪/预期的副本数。当 STATUS 显示为 Running 时，部署成功。

6.3.5. 从一个终端连接到 ZooKeeper

ZooKeeper 服务使用加密和身份验证进行保护，不应被不是 Apache Kafka Streams 的一部分的外部应用程序使用。

但是，如果要使用需要连接到 ZooKeeper 的 CLI 工具，您可以使用 ZooKeeper pod 中的终端，并连接到 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 控制台打开一个终端，或者从 CLI 运行 exec 命令。

   例如：

```
oc exec -ti my-cluster-zookeeper-0 -- bin/zookeeper-shell.sh localhost:12181 ls /
```

务必使用 localhost:12181。

6.3.6. 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

<kafka_cluster_name>-cluster-ca

带有用于加密集群通信的 Cluster CA 私钥的 secret。

<kafka_cluster_name>-cluster-ca-cert

带有集群 CA 公钥的 secret。此密钥可用于验证 Kafka 代理的身份。

<kafka_cluster_name>-clients-ca

带有用于签署用户证书的 Clients CA 私钥的 secret

<kafka_cluster_name>-clients-ca-cert

带有客户端 CA 公钥的 secret。此密钥可用于验证 Kafka 用户的身份。

<kafka_cluster_name>-cluster-operator-certs

带有 Cluster operator 密钥的 secret，用于与 Kafka 和 ZooKeeper 通信。

Zookeeper 节点

<kafka_cluster_name>-zookeeper

提供给以下 ZooKeeper 资源的名称：

- 用于管理 ZooKeeper 节点 pod 的 StrimziPodSet。
- ZooKeeper 节点使用的服务帐户。
- 为 ZooKeeper 节点配置 PodDisruptionBudget。

<kafka_cluster_name>-zookeeper-<pod_id>

StrimziPodSet 创建的 Pod。

<kafka_cluster_name>-zookeeper-nodes

无头服务需要使 DNS 直接解析 ZooKeeper pod IP 地址。

<kafka_cluster_name>-zookeeper-client

Kafka 代理使用的服务作为客户端连接到 ZooKeeper 节点。

<kafka_cluster_name>-zookeeper-config

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

<kafka_cluster_name>-zookeeper-nodes

使用 ZooKeeper 节点密钥的 secret。

<kafka_cluster_name>-network-policy-zookeeper

网络策略管理对 ZooKeeper 服务的访问。

data-<kafka_cluster_name>-zookeeper-<pod_id>

用于为特定 ZooKeeper 节点存储数据的卷的持久性卷声明。只有在选择了持久性存储来存储数据时，才会创建此资源。

Kafka 代理

<kafka_cluster_name>-kafka

提供给以下 Kafka 资源的名称：

- 用于管理 Kafka 代理 pod 的 StrimziPodSet。
- Kafka pod 使用的服务帐户。
- 为 Kafka 代理配置 PodDisruptionBudget。

<kafka_cluster_name>-kafka-<pod_id>

提供给以下 Kafka 资源的名称：

- StrimziPodSet 创建的 Pod。
- 带有 Kafka 代理配置的 ConfigMap。

<kafka_cluster_name>-kafka-brokers

服务需要 DNS 解析 Kafka 代理 pod IP 地址。

<kafka_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。

<kafka_cluster_name>-kafka-external-bootstrap

从 OpenShift 集群外部连接的客户端的 bootstrap 服务。只有在启用外部监听程序时，才会创建此资源。当监听器名称为 external 且端口为 9094 时，旧的服务名称将用于向后兼容。

<kafka_cluster_name>-kafka-<pod_id>

用于将流量从 OpenShift 集群外部路由到各个容器集的服务。只有在启用外部监听程序时，才会创建此资源。当监听器名称为 external 且端口为 9094 时，旧的服务名称将用于向后兼容。

<kafka_cluster_name>-kafka-external-bootstrap

从 OpenShift 集群外部连接的客户端的 bootstrap 路由。只有在启用了外部监听程序并设置为 type 路由 时，才会创建此资源。当监听器名称为 external 且端口为 9094 时，旧的路由名称将用于向后兼容。

<kafka_cluster_name>-kafka-<pod_id>

将来自 OpenShift 集群外的流量路由到各个容器集。只有在启用了外部监听程序并设置为 type 路由 时，才会创建此资源。当监听器名称为 external 且端口为 9094 时，旧的路由名称将用于向后兼容。

<kafka_cluster_name>-kafka-<listener_name>-bootstrap

从 OpenShift 集群外部连接的客户端的 bootstrap 服务。只有在启用外部监听程序时，才会创建此资源。新的服务名称将用于所有其他外部监听程序。

<kafka_cluster_name>-kafka-<listener_name>-<pod_id>

用于将流量从 OpenShift 集群外部路由到各个容器集的服务。只有在启用外部监听程序时，才会创建此资源。新的服务名称将用于所有其他外部监听程序。

<kafka_cluster_name>-kafka-<listener_name>-bootstrap

从 OpenShift 集群外部连接的客户端的 bootstrap 路由。只有在启用了外部监听程序并设置为 type 路由 时，才会创建此资源。新路由名称将用于所有其他外部监听程序。

<kafka_cluster_name>-kafka-<listener_name>-<pod_id>

将来自 OpenShift 集群外的流量路由到各个容器集。只有在启用了外部监听程序并设置为 type 路由 时，才会创建此资源。新路由名称将用于所有其他外部监听程序。

<kafka_cluster_name>-kafka-config

包含 Kafka 辅助配置的 ConfigMap，当禁用 UseStrimziPodSets 功能门时，代理 pod 会作为卷挂载。

<kafka_cluster_name>-kafka-brokers

带有 Kafka 代理密钥的 secret。

<kafka_cluster_name>-network-policy-kafka

网络策略管理对 Kafka 服务的访问。

strimzi-*namespace-name*-<kafka_cluster_name>-kafka-init

Kafka 代理使用的集群角色绑定。

<kafka_cluster_name>-jmx

带有 JMX 用户名和密码的 secret，用于保护 Kafka 代理端口。只有在 Kafka 中启用 JMX 时，才会创建此资源。

data-<kafka_cluster_name>-kafka-<pod_id>

用于存储特定 Kafka 代理数据的卷的持久性卷声明。只有选择了持久性存储来存储数据时，才会创建此资源。

data-<id>-<kafka_cluster_name>-kafka-<pod_id>

卷 id 的持久性卷声明用于存储特定 Kafka 代理的数据。只有在置备持久性卷来存储数据时，才会为 JBOD 卷选择持久性存储来创建此资源。

Kafka 节点池

如果您使用 Kafka 节点池，则创建的资源适用于节点池中管理的节点，无论它们是否作为代理、控制器或两者运行。命名惯例包括 Kafka 集群名称和节点池： <kafka_cluster_name>-<pool_name>。

<kafka_cluster_name>-<pool_name>

提供给 StrimziPodSet 的名称，用于管理 Kafka 节点池。

<kafka_cluster_name>-<pool_name>-<pod_id>

提供给以下 Kafka 节点池资源的名称：

- StrimziPodSet 创建的 Pod。
- 带有 Kafka 节点的 ConfigMap。

data-<kafka_cluster_name>-<pool_name>-<pod_id>

用于为特定节点存储数据的卷的持久性卷声明。只有选择了持久性存储来存储数据时，才会创建此资源。

data-<id>-<kafka_cluster_name>-<pool_name>-<pod_id>

卷 id 的持久性卷声明用于存储特定节点的数据。只有在置备持久性卷来存储数据时，才会为 JBOD 卷选择持久性存储来创建此资源。

Entity Operator

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

<kafka_cluster_name>-entity-operator

提供给以下实体 Operator 资源的名称：

- 使用主题和用户 Operator 部署。
- Entity Operator 使用的服务帐户。
- 网络策略管理对实体 Operator 指标的访问。

<kafka_cluster_name>-entity-operator-<random_string>

由实体 Operator 部署创建的 Pod。

<kafka_cluster_name>-entity-topic-operator-config

带有主题 Operator 的辅助配置的 ConfigMap。

<kafka_cluster_name>-entity-user-operator-config

带有用户 Operator 的辅助配置的 ConfigMap。

<kafka_cluster_name>-entity-topic-operator-certs

带有主题 Operator 密钥的 secret，用于与 Kafka 和 ZooKeeper 通信。

<kafka_cluster_name>-entity-user-operator-certs

带有用户 Operator 密钥的 secret，用于与 Kafka 和 ZooKeeper 通信。

strimzi-<kafka_cluster_name>-entity-topic-operator

Entity Topic Operator 使用的角色绑定。

strimzi-<kafka_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。

<kafka_cluster_name>-kafka-exporter

提供给以下 Kafka 导出器资源的名称：

- 使用 Kafka 导出器进行部署。
- 用于收集消费者滞后指标的服务。
- Kafka Exporter 使用的服务帐户。

- 网络策略用于管理对 Kafka 导出器指标的访问。

<kafka_cluster_name>-kafka-exporter-<random_string>

由 Kafka Exporter 部署创建的 Pod。

Sything Control

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

<kafka_cluster_name>-cruise-control

给定以下 Cruise 控制资源的名称：

- 使用 Cruise Control 部署。
- 用于与 Cruise Control 进行通信的服务。
- Cruise Control 使用的服务帐户。

<kafka_cluster_name>-cruise-control-<random_string>

由 Cruise Control 部署创建的 Pod。

<kafka_cluster_name>-cruise-control-config

包含 Cruise Control 辅助配置的 ConfigMap，并由 Cruise Control pod 挂载为卷。

<kafka_cluster_name>-cruise-control-certs

带有 Cruise Control 密钥的 secret，用于与 Kafka 和 ZooKeeper 通信。

<kafka_cluster_name>-network-policy-cruise-control

网络策略管理对 Cruise Control 服务的访问。

6.4. 部署 KAFKA CONNECT

Kafka Connect 是一个使用连接器插件在 Kafka 代理和其他系统间流传输数据的集成工具包。Kafka

Connect 提供了一个框架，用于将 Kafka 与外部数据源或目标（如数据库或消息传递系统）集成，用于使用连接器导入或导出数据。连接器是提供所需的连接配置的插件。

在 Apache Kafka 的 Streams 中，Kafka Connect 以分布式模式部署。Kafka Connect 也可以以独立模式工作，但 Apache Kafka Streams 不支持它。

使用 *连接器* 的概念，Kafka Connect 提供了将大量数据移至和移出 Kafka 集群的框架，同时保持可扩展性和可靠性。

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

要使用 Kafka Connect，您需要执行以下操作。

- 部署 Kafka Connect 集群
- 添加连接器以与其他系统集成

注意

术语 *连接器（connector）* 可以代表在 Kafka Connect 中运行的一个连接器实例，也可以代表一个连接器类。在本指南中，当从上下文中清除含义时，会使用 *连接器*。

6.4.1. 部署 Kafka 连接到 OpenShift 集群

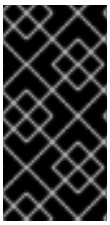
此流程演示了如何使用 Cluster Operator 将 Kafka Connect 集群部署到 OpenShift 集群。

Kafka Connect 集群部署使用可配置的节点（也称为 ***worker***）实现，它将连接器的工作负载分发为 *任务*，以便消息流具有高度可扩展且可靠的。

部署使用 YAML 文件来提供规格来创建 KafkaConnect 资源。

Apache Kafka 的流提供了 示例配置文件。在此过程中，我们使用以下示例文件：

- examples/connect/kafka-connect.yaml



重要

如果部署 Kafka Connect 集群以并行运行，则每个实例都必须为内部 Kafka Connect 主题使用唯一名称。要做到这一点，配置每个 Kafka Connect 实例来替换默认值。

先决条件

- 必须部署 Cluster Operator。
- 运行 Kafka 集群。

流程

1. 部署 Kafka 连接到您的 OpenShift 集群。使用 examples/connect/kafka-connect.yaml 文件来部署 Kafka Connect。

```
oc apply -f examples/connect/kafka-connect.yaml
```

2. 检查部署的状态：

```
oc get pods -n <my_cluster_operator_namespace>
```

输出显示部署名称和就绪

```
NAME                          READY  STATUS   RESTARTS
my-connect-cluster-connect-<pod_id>  1/1    Running  0
```

my-connect-cluster 是 Kafka Connect 集群的名称。

标识创建的每个 pod 的 pod ID。

在默认部署中，您可以创建一个单个的 Kafka Connect pod。

READY 显示就绪/预期的副本数。当 STATUS 显示为 Running 时，部署成功。

其他资源

Kafka Connect 集群配置

6.4.2. 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

提供给以下 Kafka Connect 资源的名称：

- 创建 Kafka Connect worker 节点 pod 的 StrimziPodSet。
- 无头服务，为 Kafka Connect pod 提供稳定的 DNS 名称。
- Kafka Connect pod 使用的服务帐户。
- 为 Kafka Connect worker 节点配置的 Pod 中断预算。
- 网络策略管理对 Kafka Connect REST API 的访问。

<connect_cluster_name>-connect-<pod_id>

由 Kafka Connect StrimziPodSet 创建的 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>-connect-config

包含 Kafka Connect 辅助配置的 ConfigMap，并由 Kafka Connect pod 挂载为卷。

strimzi-<namespace-name>-<connect_cluster_name>-connect-init

Kafka Connect 集群使用的集群角色绑定。

<connect_cluster_name>-connect-build

用于构建带有额外连接器插件的新容器镜像的 Pod （仅在使用 Kafka Connect Build 功能时）。

<connect_cluster_name>-connect-dockerfile

带有生成的 Dockerfile 的 ConfigMap，以使用额外的连接器插件构建新容器镜像（仅在使用 Kafka Connect 构建功能时）。

6.5. 添加 KAFKA CONNECT 连接器

Kafka Connect 使用连接器来与其他系统集成以流传输数据。连接器是 Kafka Connector 类的实例，可以是以下类型之一：

源连接器

源连接器是一个运行时实体，它从外部系统获取数据并将其作为信息提供给 Kafka。

sink 连接器

sink 连接器是一个运行时实体，它从 Kafka 主题获取信息并将其传送到外部系统。

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

使用以下方法之一在 Kafka Connect 中添加连接器插件：

- 配置 Kafka Connect 以使用插件构建新容器镜像

- 从基础 Kafka Connect 镜像创建 Docker 镜像（手动或使用持续集成）

将插件添加到容器镜像后，您可以使用以下方法启动、停止和管理连接器实例：

- 使用 Apache Kafka 的 KafkaConnector 自定义资源的 Streams
- 使用 Kafka Connect API

您还可以使用这些选项创建新的连接器实例。

6.5.1. 自动使用连接器插件构建新容器镜像

配置 Kafka Connect，以便 Apache Kafka 的 Streams 会自动使用其他连接器构建新容器镜像。您可以使用 KafkaConnect 自定义资源的 .spec.build.plugins 属性定义连接器插件。Apache Kafka 的流将自动下载，并将连接器插件添加到新容器镜像中。容器被推送到 .spec.build.output 中指定的容器存储库，并在 Kafka Connect 部署中自动使用。

先决条件

- 必须部署 Cluster Operator。
- 容器 registry。

您需要提供自己的容器 registry，从中可以推送镜像、存储和拉取镜像。Apache Kafka 的流支持私有容器 registry 以及 Quay 或 Docker Hub 等公共 registry。

流程

1. 通过在 .spec.build.output 中指定容器 registry 来配置 KafkaConnect 自定义资源，并在 .spec.build.plugins 中指定其他连接器：

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

```
spec: 1
  #...
  build:
    output: 2
      type: docker
      image: my-registry.io/my-org/my-connect-cluster:latest
      pushSecret: my-registry-credentials
    plugins: 3
      - name: connector-1
        artifacts:
          - type: tgz
            url: <url_to_download_connector_1_artifact>
            sha512sum: <SHA-512_checksum_of_connector_1_artifact>
      - name: connector-2
        artifacts:
          - type: jar
            url: <url_to_download_connector_2_artifact>
            sha512sum: <SHA-512_checksum_of_connector_2_artifact>
  #...
```

1 Kafka Connect 集群的规格。

2 （必需）推送新镜像的容器 registry 的配置。

3 （必需）连接器插件及其工件列表，以添加到新容器镜像中。每个插件必须配置至少一个 工件。

2. 创建或更新资源：

```
$ oc apply -f <kafka_connect_configuration_file>
```

3. 等待新容器镜像构建，并且部署 Kafka Connect 集群。

4. 使用 Kafka Connect REST API 或 KafkaConnector 自定义资源使用您添加的连接器插件。

其他资源

- Kafka Connect Build 模式参考

6.5.2. 使用 Kafka Connect 基础镜像中的连接器插件构建新容器镜像

使用 Kafka Connect 基础镜像中的连接器插件创建自定义 Docker 镜像。将自定义镜像添加到 /opt/kafka/plugins 目录中。

您可以使用 Red Hat Ecosystem Catalog 上的 Kafka 容器镜像作为基础镜像，以使用额外的连接器插件创建自己的自定义镜像。

在启动时，Kafka Connect 的 Apache Kafka 版本的 Streams 会加载 /opt/kafka/plugins 目录中包含的任何第三方连接器插件。

先决条件

- 必须部署 Cluster Operator。

流程

1. 使用 registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0 作为基础镜像创建一个新 Dockerfile ：

```
FROM registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0
USER root:root
COPY ./my-plugins/ /opt/kafka/plugins/
USER 1001
```

插件文件示例

```
$ tree ./my-plugins/
./my-plugins/
├── debezium-connector-mongodb
│   ├── bson-<version>.jar
│   ├── CHANGELOG.md
│   ├── CONTRIBUTE.md
│   ├── COPYRIGHT.txt
│   ├── debezium-connector-mongodb-<version>.jar
│   ├── debezium-core-<version>.jar
│   ├── LICENSE.txt
│   ├── mongodb-driver-core-<version>.jar
```

```
│   ├── README.md
│   └── # ...
├── debezium-connector-mysql
│   ├── CHANGELOG.md
│   ├── CONTRIBUTE.md
│   ├── COPYRIGHT.txt
│   ├── debezium-connector-mysql-<version>.jar
│   ├── debezium-core-<version>.jar
│   ├── LICENSE.txt
│   ├── mysql-binlog-connector-java-<version>.jar
│   ├── mysql-connector-java-<version>.jar
│   ├── README.md
│   └── # ...
└── debezium-connector-postgres
    ├── CHANGELOG.md
    ├── CONTRIBUTE.md
    ├── COPYRIGHT.txt
    ├── debezium-connector-postgres-<version>.jar
    ├── debezium-core-<version>.jar
    ├── LICENSE.txt
    ├── postgresql-<version>.jar
    ├── protobuf-java-<version>.jar
    ├── README.md
    └── # ...
```

COPY 命令指向要复制到容器镜像的插件文件。

本例为 Debezium 连接器(MongoDB、MySQL 和 PostgreSQL)添加了插件，但并不为所有文件都列出。在 Kafka Connect 中运行 Debezium 会查找与其他 Kafka Connect 任务相同的。

2. **构建容器镜像。**

3. **将自定义镜像推送到容器 registry。**

4. **指向新容器镜像。**

您可以使用以下方法之一指向镜像：

- 编辑 **KafkaConnect 自定义资源的 KafkaConnect.spec.image 属性。**

如果设置，此属性覆盖 Cluster Operator 中的 STRIMZI_KAFKA_CONNECT_IMAGES 环境变量。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec: 1
  #...
  image: my-new-container-image 2
  config: 3
    #...
```

1 Kafka Connect 集群的规格。

2 Kafka Connect pod 的 docker 镜像。

3 配置 Kafka Connect *worker*（而不是连接器）。

- 编辑 install/cluster-operator/060-Deployment-strimzi-cluster-operator.yaml 文件中的 STRIMZI_KAFKA_CONNECT_IMAGES 环境变量以指向新的容器镜像，然后重新安装 Cluster Operator。

其他资源

- 容器镜像配置和 KafkaConnect.spec.image 属性
- Cluster Operator 配置和 STRIMZI_KAFKA_CONNECT_IMAGES 变量

6.5.3. 部署 KafkaConnector 资源

部署 KafkaConnector 资源以管理连接器。KafkaConnector 自定义资源提供了由 Cluster Operator 管理连接器的 OpenShift 原生方法。您不需要发送 HTTP 请求来管理连接器，如 Kafka Connect REST API 一样。您可以通过更新其对应的 KafkaConnector 资源来管理正在运行的连接器实例，然后应用更

新。Cluster Operator 更新正在运行的连接器实例的配置。您可以通过删除其对应的 KafkaConnector 来删除连接器。

KafkaConnector 资源必须部署到它们所链接的 Kafka Connect 集群相同的命名空间中。

在此显示的配置中，autoRestart 功能被启用 (enabled: true)，用于自动重启失败的连接器和任务。您还可以注解 KafkaConnector 资源来 重启连接器 或手动 重启连接器任务。

连接器示例

您可以使用自己的连接器，或尝试 Streams for Apache Kafka 提供的示例。直到 Apache Kafka 3.1.0，Apache Kafka 中包含示例文件连接器插件。从 Apache Kafka 的 3.1.1 和 3.2.0 版本开始，需要将示例 作为任何其他连接器添加到插件路径中。

Apache Kafka 的流为 示例 KafkaConnector 配置文件(example /connect/source-connector.yaml)为示例文件连接器插件提供了一个示例，它将以下连接器实例创建为 KafkaConnector 资源：

- FileStreamSourceConnector 实例从 Kafka 许可证文件（源）读取每行，并将数据作为消息写入单个 Kafka 主题。
- 一个 FileStreamSinkConnector 实例，从 Kafka 主题读取信息，并将信息写入临时文件（接收器）。

我们使用示例文件在此流程中创建连接器。

注意

示例连接器不应在生产环境中使用。

先决条件

- Kafka Connect 部署
- Cluster Operator 正在运行

流程

1. 使用以下方法之一将 FileStreamSourceConnector 和 FileStreamSinkConnector 插件添加到 Kafka Connect：

   - 配置 Kafka Connect 以使用插件构建新容器镜像
   - 从基础 Kafka Connect 镜像创建 Docker 镜像（手动或使用持续集成）

2. 在 Kafka Connect 配置中将 strimzi.io/use-connector-resources 注解设置为 true。

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

   启用 KafkaConnector 资源后，Cluster Operator 会监视它们。

3. 编辑 examples/connect/source-connector.yaml 文件：

KafkaConnector 源连接器配置示例

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

```
    file: "/opt/kafka/LICENSE" 7
    topic: my-topic 8
    # ...
```

1

KafkaConnector 资源的名称，用作连接器的名称。使用对 OpenShift 资源有效的任何名称。

2

在其中创建连接器实例的 Kafka Connect 集群的名称。连接器必须部署到它们所链接的 Kafka Connect 集群相同的命名空间中。

3

连接器类的全名。这应该存在于 Kafka Connect 集群使用的镜像中。

4

连接器可创建的最大 Kafka Connect 任务数量。

5

启用自动重启失败的连接器和任务。默认情况下，重启数量是无限的，但您可以使用 maxRestarts 属性设置自动重启次数的最大值。

6

连接器配置 作为键值对。

7

外部数据文件的位置。在本例中，我们将 FileStreamSourceConnector 配置为从 /opt/kafka/LICENSE 文件中读取。

8

将源数据发布到的 Kafka 主题。

4. 在 OpenShift 集群中创建源 KafkaConnector ：

-

```
oc apply -f examples/connect/source-connector.yaml
```

5. 创建一个 examples/connect/sink-connector.yaml 文件：

```
touch examples/connect/sink-connector.yaml
```

6. 将以下 YAML 粘贴到 sink-connector.yaml 文件中：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-sink-connector
  labels:
    strimzi.io/cluster: my-connect
spec:
  class: org.apache.kafka.connect.file.FileStreamSinkConnector (1)
  tasksMax: 2
  config: (2)
    file: "/tmp/my-file" (3)
    topics: my-topic (4)
```

(1) 连接器类的完整名称或别名。这应该存在于 Kafka Connect 集群使用的镜像中。

(2) 连接器配置 作为键值对。

(3) 将源数据发布到的临时文件。

(4) 从中读取源数据的 Kafka 主题。

7. 在 OpenShift 集群中创建 sink KafkaConnector ：

```
oc apply -f examples/connect/sink-connector.yaml
```

8.

o.

检查是否已创建连接器资源：

```
oc get kctr --selector strimzi.io/cluster=<my_connect_cluster> -o name

my-source-connector
my-sink-connector
```

将 <my_connect_cluster> 替换为 Kafka Connect 集群的名称。

9.

在容器中，执行 kafka-console-consumer.sh 以读取源连接器写入该主题的消息：

```
oc exec <my_kafka_cluster>-kafka-0 -i -t -- bin/kafka-console-consumer.sh --
bootstrap-server <my_kafka_cluster>-kafka-bootstrap.NAMESPACE.svc:9092 --topic
my-topic --from-beginning
```

将 <my_kafka_cluster> 替换为 Kafka 集群的名称。

源和接收器连接器配置选项

连接器配置在 KafkaConnector 资源的 spec.config 属性中定义。

FileStreamSourceConnector 和 FileStreamSinkConnector 类支持与 Kafka Connect REST API 相同的配置选项。其他连接器支持不同的配置选项。

表 6.1. FileStreamSource 连接器类的配置选项

名称	类型	默认值	Description
file	字符串	null	将消息写入的源文件。如果没有指定，则使用标准输入。
topic	list	null	将数据发布到的 Kafka 主题。

表 6.2. FileStreamSinkConnector 类的配置选项

名称	类型	默认值	Description
file	字符串	null	将消息写入的目标文件。如果没有指定，则使用标准输出。

名称	类型	默认值	Description
topics	list	null	从一个或多个 Kafka 主题读取数据。
topics.regex	字符串	null	与一个或多个 Kafka 主题匹配的正则表达式，用于读取数据。

6.5.4. 公开 Kafka Connect API

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

Kafka Connect REST API 支持的操作信息包括在 Apache Kafka Connect API 文档。

注意

strimzi.io/use-connector-resources 注解启用 KafkaConnectors。如果您将注解应用到 KafkaConnect 资源配置，则需要将其删除以使用 Kafka Connect API。否则，Cluster Operator 会恢复使用 Kafka Connect REST API 进行的手动更改。

您可以将连接器配置添加为 JSON 对象。

添加连接器配置的 curl 请求示例

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

API 只能在 OpenShift 集群中访问。如果要使 Kafka Connect API 可以被 OpenShift 集群外部运行的应用程序访问，您可以通过创建以下功能之一来手动公开它：

- LoadBalancer 或 NodePort 类型服务
- Ingress 资源（仅限 Kubernetes）
- OpenShift 路由（仅限 OpenShift）

注意

连接是不安全的，因此建议从外部访问。

如果您决定创建服务，请使用来自 *<connect_cluster_name>*-connect-api 服务的 selector 配置服务将流量路由到的 pod：

服务的选择器配置

```
# ...
selector:
  strimzi.io/cluster: my-connect-cluster 1
  strimzi.io/kind: KafkaConnect
  strimzi.io/name: my-connect-cluster-connect 2
#...
```

1

OpenShift 集群中的 Kafka Connect 自定义资源的名称。

2

Cluster Operator 创建的 Kafka Connect 部署的名称。

您还必须创建一个允许来自外部客户端的 HTTP 请求的 NetworkPolicy。

允许对 Kafka Connect API 的请求的 NetworkPolicy 示例

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-custom-connect-network-policy
spec:
  ingress:
  - from:
    - podSelector: 1
        matchLabels:
          app: my-connector-manager
    ports:
    - port: 8083
      protocol: TCP
  podSelector:
    matchLabels:
      strimzi.io/cluster: my-connect-cluster
      strimzi.io/kind: KafkaConnect
      strimzi.io/name: my-connect-cluster-connect
  policyTypes:
  - Ingress
```

1

允许连接到 API 的 pod 标签。

要在集群外添加连接器配置，请使用 curl 命令中公开 API 的资源 URL。

6.5.5. 限制对 Kafka Connect API 的访问

仅将对 Kafka Connect API 的访问限制为可信用户，以防止未经授权的操作和潜在的安全问题。Kafka Connect API 提供了大量更改连接器配置的功能，这有助于采取安全措施。有权访问 Kafka Connect API 的人员可能会获得管理员可能假设的敏感信息是安全的。

Kafka Connect REST API 可以被经过身份验证访问 OpenShift 集群的任何人访问，并知道端点 URL，其中包括主机名/IP 地址和端口号。

例如，假设机构使用 Kafka Connect 集群和连接器将客户数据库的敏感数据流传输到中央数据库。管理员使用配置供应商插件存储与连接到客户数据库和中央数据库相关的敏感信息，如数据库连接详情和身份验证凭据。配置提供程序保护此敏感信息无法向未授权用户公开。但是，有权访问 Kafka Connect API 的用户仍然可以获得对客户数据库的访问权限，而无需经过管理员的批准。他们可以通过设置一个假的数据库并将连接器配置为连接它。然后，他们可以修改连接器配置以指向客户数据库，但不是将数据发送到中央数据库，而是将其发送到假的数据库。通过将连接器配置为连接到假的数据库，可以截获连接到客户端数据库的登录详情和凭证，即使它们被安全地存储在配置提供程序中。

如果您使用 KafkaConnector 自定义资源，默认情况下，OpenShift RBAC 规则只允许 OpenShift 集群管理员更改连接器。您还可以指定 非集群管理员来管理 Apache Kafka 资源的 Streams。在 Kafka Connect 配置中启用 KafkaConnector 资源后，Cluster Operator 会恢复使用 Kafka Connect REST API 所做的更改。如果您不使用 KafkaConnector 资源，默认的 RBAC 规则不会限制对 Kafka Connect API 的访问。如果要使用 OpenShift RBAC 限制对 Kafka Connect REST API 的直接访问，则需要启用和使用 KafkaConnector 资源。

为提高安全性，我们建议为 Kafka Connect API 配置以下属性：

org.apache.kafka.disallowed.login.modules

(Kafka 3.4 或更高版本)设置 org.apache.kafka.disallowed.login.modules Java 系统属性，以防止使用不安全的登录模块。例如，指定 com.sun.security.auth.module.JndiLoginModule 会阻止使用 Kafka JndiLoginModule。

禁止登录模块配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  # ...
  jvmOptions:
    javaSystemProperties:
      - name: org.apache.kafka.disallowed.login.modules
        value: com.sun.security.auth.module.JndiLoginModule,
org.apache.kafka.common.security.kerberos.KerberosLoginModule
  # ...
```

只允许可信登录模块，并按照您使用的版本的 Kafka 的最新建议进行操作。作为最佳实践，您应该使用 org.apache.kafka.disallowed.login.modules 系统属性在 Kafka Connect 配置中明确禁止不安全的登录模块。

connector.client.config.override.policy

将 connector.client.config.override.policy 属性设置为 None，以防止连接器配置覆盖 Kafka Connect 配置及其使用的使用者和制作者。

指定连接器覆盖策略的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  # ...
  config:
    connector.client.config.override.policy: None
# ...
```

6.5.6. 使用 Kafka Connect API 切换到使用 KafkaConnector 自定义资源

您可以从使用 Kafka Connect API 切换到使用 KafkaConnector 自定义资源来管理连接器。要进行交换机，请按照以下顺序执行以下操作：

1. 使用配置部署 KafkaConnector 资源，以创建您的连接器实例。
2. 通过将 strimzi.io/use-connector-resources 注解设置为 true 来启用 Kafka Connect 配置中的 KafkaConnector 资源。

警告

如果在创建它们前启用 KafkaConnector 资源，请删除所有连接器。

要从使用 KafkaConnector 资源切换到使用 Kafka Connect API，首先从 Kafka Connect 配置中删除启用 KafkaConnector 资源的注解。否则，Cluster Operator 会恢复使用 Kafka Connect REST API 进行的手动更改。

在进行交换机时，检查 KafkaConnect 资源的状态。metadata.generation （部署的当前版本）的值必须与 status.observedGeneration （资源的最新协调）匹配。当 Kafka Connect 集群为 Ready 时，您可以删除 KafkaConnector 资源。

6.6. 部署 KAFKA MIRRORMAKER

Kafka MirrorMaker 在两个或多个 Kafka 集群之间复制数据，并在数据中心之间复制数据。此过程称为镜像(mirror)，以避免与 Kafka 分区复制概念混淆。MirrorMaker 使用来自源集群的消息，并将这些消息重新发布到目标集群。

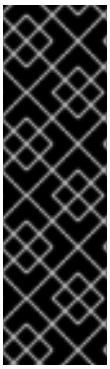
集群间的数据复制支持以下场景：

- 在系统失败时恢复数据
- 从多个源集群整合数据以进行集中分析
- 对特定集群的数据访问限制
- 在特定位置置备数据以提高延迟

6.6.1. 将 Kafka MirrorMaker 部署到 OpenShift 集群

此流程演示了如何使用 Cluster Operator 将 Kafka MirrorMaker 集群部署到 OpenShift 集群。

部署使用 YAML 文件来提供规格，根据部署的 MirrorMaker 版本创建 KafkaMirrorMaker 或 KafkaMirrorMaker2 资源。MirrorMaker 2 基于 Kafka Connect，并使用其配置属性。



重要

Kafka MirrorMaker 1 （称为文档中的 *MirrorMaker*）已在 Apache Kafka 3.0.0 中弃用，并将在 Apache Kafka 4.0.0 中删除。因此，用于部署 Kafka MirrorMaker 1 的 KafkaMirrorMaker 自定义资源也已在 Apache Kafka 的 Streams 中弃用。当使用 Apache Kafka 4.0.0 时，KafkaMirrorMaker 资源将从 Apache Kafka 的 Streams 中删除。作为替代方法，在 IdentityReplicationPolicy 中使用 KafkaMirrorMaker2 自定义资源。

Apache Kafka 的流提供了 示例配置文件。在此过程中，我们使用以下示例文件：

- examples/mirror-maker/kafka-mirror-maker.yaml
- examples/mirror-maker/kafka-mirror-maker-2.yaml

重要

如果部署 MirrorMaker 2 集群以并行运行，使用相同的目标 Kafka 集群，每个实例都必须对内部 Kafka Connect 主题使用唯一的名称。为此，请配置每个 MirrorMaker 2 实例来替换默认值。

先决条件

- 必须部署 Cluster Operator。

流程

1. 将 Kafka MirrorMaker 部署到 OpenShift 集群：

   对于 MirrorMaker：

   -

```
oc apply -f examples/mirror-maker/kafka-mirror-maker.yaml
```

对于 MirrorMaker 2：

```
oc apply -f examples/mirror-maker/kafka-mirror-maker-2.yaml
```

2. 检查部署的状态：

```
oc get pods -n <my_cluster_operator_namespace>
```

输出显示部署名称和就绪

```
NAME                                READY  STATUS   RESTARTS
my-mirror-maker-mirror-maker-<pod_id>   1/1    Running  1
my-mm2-cluster-mirrormaker2-<pod_id>    1/1    Running  1
```

my-mirror-maker 是 Kafka MirrorMaker 集群的名称。my-mm2-cluster 是 Kafka MirrorMaker 2 集群的名称。

标识创建的每个 pod 的 pod ID。

使用默认部署，您要安装单个 MirrorMaker 或 MirrorMaker 2 pod。

READY 显示就绪/预期的副本数。当 STATUS 显示为 Running 时，部署成功。

其他资源

- Kafka MirrorMaker 集群配置

6.6.2. Kafka MirrorMaker 2 集群资源列表

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

<mirrormaker2_cluster_name>-mirrormaker2

提供给以下 MirrorMaker 2 资源的名称：

- 创建 MirrorMaker 2 worker 节点 pod 的 StrimziPodSet。
- 无头服务，为 MirrorMaker 2 pod 提供稳定的 DNS 名称。
- MirrorMaker 2 pod 使用的服务帐户。
- 为 MirrorMaker 2 worker 节点配置的 Pod 中断预算。
- 网络策略管理对 MirrorMaker 2 REST API 的访问。

<mirrormaker2_cluster_name>-mirrormaker2-<pod_id>

由 MirrorMaker 2 StrimziPodSet 创建的 Pod。

<mirrormaker2_cluster_name>-mirrormaker2-api

公开用于管理 MirrorMaker 2 集群的 REST 接口的服务。

<mirrormaker2_cluster_name>-mirrormaker2-config

包含 MirrorMaker 2 辅助配置的 ConfigMap，并由 MirrorMaker 2 Pod 挂载为卷。

strimzi-<namespace-name>-<mirrormaker2_cluster_name>-mirrormaker2-init

MirrorMaker 2 集群使用的集群角色绑定。

6.6.3. Kafka MirrorMaker 集群资源列表

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

<mirrormaker_cluster_name>-mirror-maker

提供给以下 MirrorMaker 资源的名称：

- 负责创建 MirrorMaker Pod 的部署。
- MirrorMaker 节点使用的服务帐户。
- 为 MirrorMaker worker 节点配置的 Pod Disruption Budget。

<mirrormaker_cluster_name>-mirror-maker-config

包含 MirrorMaker 的辅助配置的 ConfigMap，由 MirrorMaker Pod 挂载为卷。

6.7. 部署 KAFKA BRIDGE

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

6.7.1. 在 OpenShift 集群中部署 Kafka Bridge

此流程演示了如何使用 Cluster Operator 将 Kafka Bridge 集群部署到 OpenShift 集群。

部署使用 YAML 文件来提供规格来创建 KafkaBridge 资源。

Apache Kafka 的流提供了 示例配置文件。在此过程中，我们使用以下示例文件：

- examples/bridge/kafka-bridge.yaml

先决条件

- 必须部署 Cluster Operator。

流程

1. 将 Kafka Bridge 部署到 OpenShift 集群：

```
oc apply -f examples/bridge/kafka-bridge.yaml
```

2. 检查部署的状态：

```
oc get pods -n <my_cluster_operator_namespace>
```

输出显示部署名称和就绪

```
NAME                    READY  STATUS   RESTARTS
my-bridge-bridge-<pod_id>  1/1    Running  0
```

my-bridge 是 Kafka Bridge 集群的名称。

标识创建的每个 pod 的 pod ID。

使用默认部署，您会安装单个 Kafka Bridge pod。

READY 显示就绪/预期的副本数。当 STATUS 显示为 Running 时，部署成功。

其他资源

- Kafka Bridge 集群配置
- 使用 Apache Kafka Bridge 的流

6.7.2. 将 Kafka Bridge 服务公开给本地机器

使用端口转发将 Apache Kafka Bridge 服务的流公开给您的 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/my-bridge-bridge-<pod_id>
```

2. 连接到端口 8080 上的 Kafka Bridge pod：

```
oc port-forward pod/my-bridge-bridge-<pod_id> 8080:8080 &
```

注意

如果本地计算机上的端口 8080 已经在使用，请使用备用 HTTP 端口，如 8008。

API 请求现在从本地机器上的端口 8080 转发到 Kafka Bridge pod 中的端口 8080。

6.7.3. 在 OpenShift 之外访问 Kafka Bridge

部署后，Apache Kafka Bridge 的流只能被在同一 OpenShift 集群中运行的应用程序访问。这些应用程序使用 &lt ;*kafka_bridge_name*> -bridge-service 服务来访问 API。

如果要使 Kafka Bridge 可以被 OpenShift 集群外部运行的应用程序访问，您可以通过创建以下功能之一来手动公开它：

- LoadBalancer 或 NodePort 类型服务
- Ingress 资源（仅限 Kubernetes）

- OpenShift 路由（仅限 OpenShift）

如果您决定创建服务，请使用 < *kafka_bridge_name*>-bridge-service 服务 的选择器 中的标签来配置服务将流量路由到的 pod：

```
# ...
selector:
  strimzi.io/cluster: kafka-bridge-name 1
  strimzi.io/kind: KafkaBridge
#...
```

1 OpenShift 集群中的 Kafka Bridge 自定义资源的名称。

6.7.4. 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

用于创建 Kafka Bridge worker 节点 pod 的部署。

<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 acillary 配置的 ConfigMap，并由 Kafka 代理 pod 挂载为卷。

<bridge_cluster_name>-bridge

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

6.8. APACHE KAFKA OPERATOR 的 STREAMS 的替代独立部署选项

您可以对主题 Operator 和 User Operator 执行独立部署。如果您使用不是由 Cluster Operator 管理的 Kafka 集群，请考虑这些 Operator 的独立部署。

您将操作器部署到 OpenShift。Kafka 可以在 OpenShift 外部运行。例如，您可以使用 Kafka 作为受管服务。您可以调整独立 Operator 的部署配置，以匹配 Kafka 集群的地址。

6.8.1. 部署独立主题 Operator

此流程演示了如何以单向模式部署主题 Operator，作为主题管理的独立组件。您可以将独立主题 Operator 与不是由 Cluster Operator 管理的 Kafka 集群一起使用。单向主题管理仅通过 KafkaTopic 资源维护主题。有关单向主题管理的详情，请参考 第 10.1 节 “主题管理模式”。另外，还显示了以双向模式部署主题 Operator 的备用配置。

独立部署文件随 Apache Kafka 的 Streams 提供。使用 05-Deployment-strimzi-topic-operator.yaml 部署文件来部署 Topic Operator。添加或设置与 Kafka 集群建立连接所需的环境变量。

主题 Operator 监视单个命名空间中的 KafkaTopic 资源。您可以在 Topic Operator 配置中指定要监视的命名空间以及与 Kafka 集群的连接。单个主题 Operator 可以监视单个命名空间。一个命名空间应该只由一个 Topic Operator 监视。如果要使用多个主题 Operator，请将每个主题配置为监视不同的命名空间。这样，您可以将主题 Operator 与多个 Kafka 集群一起使用。

先决条件

- 您正在运行一个 Kafka 集群，供 Topic Operator 连接到。

  只要为连接正确配置了独立主题 Operator，Kafka 集群就可以在裸机环境、虚拟机或受管云应用程序服务上运行。

流程

1. 编辑 install/topic-operator/05-Deployment-strimzi-topic-operator.yaml 独立部署文件中的 env 属性。

   独立主题 Operator 部署配置示例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-topic-operator
  labels:
    app: strimzi
spec:
  # ...
```

```
template:
 # ...
 spec:
  # ...
  containers:
   - name: strimzi-topic-operator
     # ...
     env:
      - name: STRIMZI_NAMESPACE (1)
        valueFrom:
         fieldRef:
          fieldPath: metadata.namespace
      - name: STRIMZI_KAFKA_BOOTSTRAP_SERVERS (2)
        value: my-kafka-bootstrap-address:9092
      - name: STRIMZI_RESOURCE_LABELS (3)
        value: "strimzi.io/cluster=my-cluster"
      - name: STRIMZI_FULL_RECONCILIATION_INTERVAL_MS (4)
        value: "120000"
      - name: STRIMZI_LOG_LEVEL (5)
        value: INFO
      - name: STRIMZI_TLS_ENABLED (6)
        value: "false"
      - name: STRIMZI_JAVA_OPTS (7)
        value: "-Xmx=512M -Xms=256M"
      - name: STRIMZI_JAVA_SYSTEM_PROPERTIES (8)
        value: "-Djavax.net.debug=verbose -DpropertyName=value"
      - name: STRIMZI_PUBLIC_CA (9)
        value: "false"
      - name: STRIMZI_TLS_AUTH_ENABLED (10)
        value: "false"
      - name: STRIMZI_SASL_ENABLED (11)
        value: "false"
      - name: STRIMZI_SASL_USERNAME (12)
        value: "admin"
      - name: STRIMZI_SASL_PASSWORD (13)
        value: "password"
      - name: STRIMZI_SASL_MECHANISM (14)
        value: "scram-sha-512"
      - name: STRIMZI_SECURITY_PROTOCOL (15)
        value: "SSL"
      - name: STRIMZI_USE_FINALIZERS
        value: "false" (16)
```

(1)

主题 Operator 的 OpenShift 命名空间，以监视 KafkaTopic 资源。指定 Kafka 集群的命名空间。

(2)

3

用于标识 Topic Operator 管理的 KafkaTopic 资源的标签。这不一定是 Kafka 集群的名称。它可以是分配给 KafkaTopic 资源的标签。如果部署多个主题 Operator，则标签必须为每个主题都是唯一的。也就是说，操作员无法管理同一资源。

4

定期协调之间的间隔，以毫秒为单位。默认值为 120000 (2 分钟)。

5

打印日志信息的级别。您可以将级别设置为 ERROR、WARNING、INFO、DEBUG 或 TRACE。

6

启用 TLS 支持与 Kafka 代理的加密通信。

7

（可选）运行主题 Operator 的 JVM 使用的 Java 选项。

8

（可选）为主题 Operator 设置的调试(-D)选项。

9

（可选）如果通过 STRIMZI_TLS_ENABLED 启用 TLS，则跳过生成信任存储证书。如果启用了此环境变量，代理必须为其 TLS 证书使用公共可信证书颁发机构。默认值为 false。

10

（可选）为 mTLS 身份验证生成密钥存储证书。把它设置为 false 可禁用使用 mTLS 到 Kafka 代理的客户端身份验证。默认值是 true。

11

（可选）在连接到 Kafka 代理时为客户端身份验证启用 SASL 支持。默认值为 false。

12

13

（可选）客户端身份验证的 SASL 密码。仅在通过 STRIMZI_SASL_ENABLED 启用 SASL 时才需要。

14

（可选）客户端身份验证的 SASL 机制。仅在通过 STRIMZI_SASL_ENABLED 启用 SASL 时才需要。您可以将值设为 plain、scram-sha-256 或 scram-sha-512。

15

（可选）用于与 Kafka 代理通信的安全协议。默认值为 "PLAINTEXT"。您可以将值设为 PLAINTEXT、SSL、SSL、SASL_PLAINTEXT 或 SASL_SSL。

16

如果您不想使用终结器来控制 主题删除，请将 STRIMZI_USE_FINALIZERS 设置为 false。

2. 如果要连接到使用公共证书颁发机构的 Kafka 代理，请将 STRIMZI_PUBLIC_CA 设置为 true。将此属性设置为 true，例如，如果您使用 Amazon AWS MSK 服务。

3. 如果您使用 STRIMZI_TLS_ENABLED 环境变量启用 mTLS，请指定用于验证与 Kafka 集群连接的密钥存储和信任存储。

mTLS 配置示例

```
# ....
env:
  - name: STRIMZI_TRUSTSTORE_LOCATION 1
    value: "/path/to/truststore.p12"
  - name: STRIMZI_TRUSTSTORE_PASSWORD 2
    value: "TRUSTSTORE-PASSWORD"
  - name: STRIMZI_KEYSTORE_LOCATION 3
    value: "/path/to/keystore.p12"
  - name: STRIMZI_KEYSTORE_PASSWORD 4
    value: "KEYSTORE-PASSWORD"
# ...
```

1 truststore 包含用于为 Kafka 和 ZooKeeper 服务器证书签名的证书颁发机构的公钥。

2 用于访问 truststore 的密码。

3 密钥存储包含 mTLS 身份验证的私钥。

4 用于访问密钥存储的密码。

4. 应用对部署配置的 更改 来部署主题 Operator。

5. 检查部署的状态：

```
oc get deployments
```

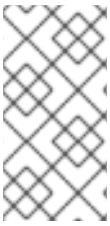
输出显示部署名称和就绪

```
NAME                      READY  UP-TO-DATE  AVAILABLE
strimzi-topic-operator  1/1    1           1
```

READY 显示就绪/预期的副本数。当 AVAILABLE 输出显示为 1 时，部署成功。

6.8.1.1. 为双向主题管理部署独立主题 Operator

双向主题管理需要 ZooKeeper 用于集群管理，并通过 KafkaTopic 资源并在 Kafka 集群中维护主题。如果要在此模式中切换到使用主题 Operator，请按照以下步骤部署独立主题 Operator。



注意

当功能门使主题 Operator 在单向模式下运行时，双向模式将分阶段化。这个转换旨在增强用户体验，特别是在 KRaft 模式中支持 Kafka。

1. **取消部署当前独立主题 Operator。**

   保留 KafkaTopic 资源，这些资源由主题 Operator 再次部署时获取。

2. **编辑独立主题 Operator 的 Deployment 配置，使其包含与 ZooKeeper 相关的环境变量：**

   - **STRIMZI_ZOOKEEPER_CONNECT**
   - **STRIMZI_ZOOKEEPER_SESSION_TIMEOUT_MS**
   - **TC_ZK_CONNECTION_TIMEOUT_MS**
   - **STRIMZI_USE_ZOOKEEPER_TOPIC_STORE**

   它是定义是否使用了双向主题 Operator 的 ZooKeeper 变量是否存在。单向主题管理不使用 ZooKeeper。如果没有 ZooKeeper 环境变量，则使用 unidirectional Topic Operator。否则会使用双向主题 Operator。

   如果需要，可以添加在单向模式中使用的其他环境变量：

   - **STRIMZI_REASSIGN_THROTTLE**
   - **STRIMZI_REASSIGN_VERIFY_INTERVAL_MS**
   - **STRIMZI_TOPIC_METADATA_MAX_ATTEMPTS**

- STRIMZI_TOPICS_PATH
- STRIMZI_STORE_TOPIC
- STRIMZI_STORE_NAME
- STRIMZI_APPLICATION_ID
- STRIMZI_STALE_RESULT_TIMEOUT_MS

双向主题管理的独立主题 Operator 部署配置示例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-topic-operator
  labels:
    app: strimzi
spec:
  # ...
  template:
    # ...
    spec:
      # ...
      containers:
        - name: strimzi-topic-operator
          # ...
          env:
            - name: STRIMZI_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: STRIMZI_KAFKA_BOOTSTRAP_SERVERS
              value: my-kafka-bootstrap-address:9092
            - name: STRIMZI_RESOURCE_LABELS
              value: "strimzi.io/cluster=my-cluster"
            - name: STRIMZI_ZOOKEEPER_CONNECT (1)
              value: my-cluster-zookeeper-client:2181
            - name: STRIMZI_ZOOKEEPER_SESSION_TIMEOUT_MS (2)
              value: "18000"
            - name: STRIMZI_TOPIC_METADATA_MAX_ATTEMPTS (3)
              value: "6"
            - name: STRIMZI_FULL_RECONCILIATION_INTERVAL_MS
```

```
      value: "120000"
    - name: STRIMZI_LOG_LEVEL
      value: INFO
    - name: STRIMZI_TLS_ENABLED
      value: "false"
    - name: STRIMZI_JAVA_OPTS
      value: "-Xmx=512M -Xms=256M"
    - name: STRIMZI_JAVA_SYSTEM_PROPERTIES
      value: "-Djavax.net.debug=verbose -DpropertyName=value"
    - name: STRIMZI_PUBLIC_CA
      value: "false"
    - name: STRIMZI_TLS_AUTH_ENABLED
      value: "false"
    - name: STRIMZI_SASL_ENABLED
      value: "false"
    - name: STRIMZI_SASL_USERNAME
      value: "admin"
    - name: STRIMZI_SASL_PASSWORD
      value: "password"
    - name: STRIMZI_SASL_MECHANISM
      value: "scram-sha-512"
    - name: STRIMZI_SECURITY_PROTOCOL
      value: "SSL"
```

1

(ZooKeeper)用于连接到 ZooKeeper 集群的地址的主机和端口对。这必须是 Kafka 集群使用的同一 ZooKeeper 集群。

2

(ZooKeeper) ZooKeeper 会话超时，以毫秒为单位。默认值为 18000 (18 秒)。

3

从 Kafka 获取主题元数据的尝试次数。每次尝试之间的时间都定义为 exponential backoff。当主题创建因为分区或副本数增加时，请考虑增加这个值。默认值为 6 次。

3. 应用对部署配置的 更改 来部署主题 Operator。

6.8.2. 部署独立用户 Operator

此流程演示了如何将 User Operator 部署为用户管理的独立组件。您可以将独立用户 Operator 与不是由 Cluster Operator 管理的 Kafka 集群一起使用。

独立部署可以处理任何 Kafka 集群。

独立部署文件随 Apache Kafka 的 Streams 提供。使用 05-Deployment-strimzi-user-operator.yaml 部署文件来部署 User Operator。添加或设置与 Kafka 集群建立连接所需的环境变量。

User Operator 监视单个命名空间中的 KafkaUser 资源。您可以在 User Operator 配置中指定要监视的命名空间以及与 Kafka 集群的连接。单个 User Operator 可以监视单个命名空间。一个命名空间应该只由一个 User Operator 监视。如果要使用多个 User Operator，请将每个 Operator 配置为监视不同的命名空间。这样，您可以将 User Operator 与多个 Kafka 集群一起使用。

先决条件

- 您正在运行一个 Kafka 集群，供 User Operator 连接。

  只要为连接正确配置了独立用户 Operator，Kafka 集群就可以在裸机环境、虚拟机或受管云应用程序服务上运行。

流程

1. 编辑 install/user-operator/05-Deployment-strimzi-user-operator.yaml 独立部署文件中的以下 env 属性。

   独立用户 Operator 部署配置示例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-user-operator
  labels:
    app: strimzi
spec:
  # ...
  template:
    # ...
    spec:
      # ...
      containers:
        - name: strimzi-user-operator
          # ...
          env:
```

```
- name: STRIMZI_NAMESPACE (1)
  valueFrom:
    fieldRef:
      fieldPath: metadata.namespace
- name: STRIMZI_KAFKA_BOOTSTRAP_SERVERS (2)
  value: my-kafka-bootstrap-address:9092
- name: STRIMZI_CA_CERT_NAME (3)
  value: my-cluster-clients-ca-cert
- name: STRIMZI_CA_KEY_NAME (4)
  value: my-cluster-clients-ca
- name: STRIMZI_LABELS (5)
  value: "strimzi.io/cluster=my-cluster"
- name: STRIMZI_FULL_RECONCILIATION_INTERVAL_MS (6)
  value: "120000"
- name: STRIMZI_WORK_QUEUE_SIZE (7)
  value: 10000
- name: STRIMZI_CONTROLLER_THREAD_POOL_SIZE (8)
  value: 10
- name: STRIMZI_USER_OPERATIONS_THREAD_POOL_SIZE (9)
  value: 4
- name: STRIMZI_LOG_LEVEL (10)
  value: INFO
- name: STRIMZI_GC_LOG_ENABLED (11)
  value: "true"
- name: STRIMZI_CA_VALIDITY (12)
  value: "365"
- name: STRIMZI_CA_RENEWAL (13)
  value: "30"
- name: STRIMZI_JAVA_OPTS (14)
  value: "-Xmx=512M -Xms=256M"
- name: STRIMZI_JAVA_SYSTEM_PROPERTIES (15)
  value: "-Djavax.net.debug=verbose -DpropertyName=value"
- name: STRIMZI_SECRET_PREFIX (16)
  value: "kafka-"
- name: STRIMZI_ACLS_ADMIN_API_SUPPORTED (17)
  value: "true"
- name: STRIMZI_MAINTENANCE_TIME_WINDOWS (18)
  value: '* * 8-10 * * ?;* * 14-15 * * ?'
- name: STRIMZI_KAFKA_ADMIN_CLIENT_CONFIGURATION (19)
  value: |
    default.api.timeout.ms=120000
    request.timeout.ms=60000
```

(1)

User Operator 的 OpenShift 命名空间以监视 KafkaUser 资源。只能指定一个命名空间。

2

bootstrap 代理地址的主机和端口对，用于发现并连接到 Kafka 集群中的所有代理。使用逗号分隔的列表，在服务器停机时指定两个或三个代理地址。

3

包含用于 mTLS 身份验证的新用户证书的 CA （证书颁发机构）值的 OpenShift Secret (ca.crt)值。

4

包含 CA 的私钥(ca.key)值的 OpenShift Secret，它签署 mTLS 身份验证的新用户证书。

5

标识由 User Operator 管理的 KafkaUser 资源的标签。这不一定是 Kafka 集群的名称。它可以是分配给 KafkaUser 资源的标签。如果部署多个 User Operator，则每个标签都必须是唯一的。也就是说，操作员无法管理同一资源。

6

定期协调之间的间隔，以毫秒为单位。默认值为 120000 (2 分钟)。

7

控制器事件队列的大小。队列的大小应至少与预期 User Operator 操作用户的最大数量相同。默认值为 1024。

8

用于协调用户的 worker 池的大小。较大的池可能需要更多资源，但它也会处理更多 KafkaUser 资源。默认值为 50。

9

Kafka Admin API 和 OpenShift 操作的 worker 池的大小。较大的池可能需要更多资源，但它也会处理更多的 KafkaUser 资源。默认为 4。

10

打印日志信息的级别。您可以将级别设置为 ERROR、WARNING、INFO、DEBUG 或 TRACE。

11

启用垃圾回收(GC)日志记录。默认值是 true。

12

CA 的有效性周期。默认值为 365 天。

13

CA 的续订周期。续订周期从当前证书的到期日期开始向后测量。默认值为 30 天，在旧证书过期前启动证书续订。

14

（可选）运行 User Operator 的 JVM 使用的 Java 选项

15

（可选）为 User Operator 设置的调试(-D)选项

16

（可选）由 User Operator 创建的 OpenShift secret 名称的前缀。

17

（可选） 指示 Kafka 集群是否支持使用 Kafka Admin API 管理授权 ACL 规则。当设置为 false 时，User Operator 将拒绝具有 simple 授权 ACL 规则的所有资源。这有助于避免 Kafka 集群日志中不必要的异常。默认值是 true。

18

（可选） Semi-colon separated list of Cron Expressions 定义过期用户证书的维护时间窗。

19

（可选）配置用户 Operator 以属性格式使用的 Kafka Admin 客户端的配置选项。

2. 如果您使用 mTLS 连接到 Kafka 集群，请指定用于验证连接的 secret。否则，请转到下一步。

mTLS 配置示例

```
# ....
env:
  - name: STRIMZI_CLUSTER_CA_CERT_SECRET_NAME 1
    value: my-cluster-cluster-ca-cert
  - name: STRIMZI_EO_KEY_SECRET_NAME 2
    value: my-cluster-entity-operator-certs
# ..."
```

1 包含为 Kafka 代理证书签名的 CA 的公钥(ca.crt)值的 OpenShift Secret。

2 包含证书公钥(entity-operator.crt)和私钥(entity-operator.key)的 OpenShift Secret，用于针对 Kafka 集群的 mTLS 身份验证。

3. 部署 User Operator。

```
oc create -f install/user-operator
```

4. 检查部署的状态：

```
oc get deployments
```

输出显示部署名称和就绪

```
NAME                  READY  UP-TO-DATE  AVAILABLE
strimzi-user-operator  1/1    1           1
```

READY 显示就绪/预期的副本数。当 AVAILABLE 输出显示为 1 时，部署成功。

第 7 章 功能门

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

功能门的目的是在功能完全采用前进行试验和测试。功能门的状态（启用或禁用）默认可能会有所不同，具体取决于其成熟度等级。

作为功能门获取并达到正式发行(GA)，它默认转换为启用的状态，并成为 Apache Kafka 部署的 Streams 的永久部分。在 GA 阶段无法禁用功能门。

7.1. GRADUATED 功能门(GA)

graduated 功能门已正式发布(GA)，并被永久启用的功能。

7.1.1. ControlPlaneListener 功能门

ControlPlaneListener 功能门为数据复制和协调分离监听程序：

- Kafka 控制器和代理之间的连接在端口 9090 上使用内部 *control plane 侦听器*。
- 在代理间复制数据，以及从 Apache Kafka operator、Cruise Control 或 Kafka Exporter 的 Streams 的内部连接在端口 9091 上使用 *复制监听程序*。

重要

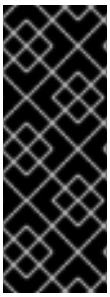
永久启用 ControlPlaneListener 功能门后，无法直接升级或降级 Apache Kafka 1.7 及更早的流(Apache Kafka 2.3 及更新版本)。您必须首先通过 Apache Kafka 版本的一个流升级或降级，禁用 ControlPlaneListener 功能门，然后降级或升级（启用了功能门）到目标版本。

7.1.2. ServiceAccountPatching 功能门

ServiceAccountPatching 功能门可确保 Cluster Operator 始终协调服务帐户并根据需要更新它们。例如，当您使用自定义资源的 template 属性更改服务帐户标签或注解时，Operator 会在现有服务帐户资源中自动更新它们。

7.1.3. UseStrimziPodSets 功能门

UseStrimziPodSets 功能门引入了 StrimziPodSet 自定义资源来管理 Kafka 和 ZooKeeper pod，替换 OpenShift StatefulSet 资源的使用。



重要

永久启用 UseStrimziPodSets 功能门后，无法直接从 Streams for Apache Kafka 2.5 及更新版本降级到 Streams for Apache Kafka 2.0 或更早版本。您必须首先通过一个流进行降级，用于 Apache Kafka 版本 in-between，禁用 UseStrimziPodSets 功能门，然后降级到 Streams for Apache Kafka 2.0 或更早版本。

7.1.4. StableConnectIdentities 功能门

StableConnectIdentities 功能门引入了 StrimziPodSet 自定义资源来管理 Kafka Connect 和 Kafka MirrorMaker 2 pod，替换 OpenShift Deployment 资源的使用。

StrimziPodSet 资源为 pod 提供稳定名称和稳定的地址，这些地址在滚动升级过程中不会改变，替换 OpenShift Deployment 资源的使用。

重要

永久启用 StableConnectIdentities 功能门后，无法直接从 Apache Kafka 2.7 及更新版本的流降级到 Streams for Apache Kafka 2.3 或更早版本。您必须首先通过一个流进行降级，用于 Apache Kafka 版本 in-between，禁用 StableConnectIdentities 功能门，然后降级到 Streams for Apache Kafka 2.3 或更早版本。

7.2. 稳定的功能门(BETA)

稳定的功能门已达到 beta 阶段，一般默认为所有用户启用。稳定的功能门是生产环境就绪，但仍然可以禁用它们。

7.2.1. UseKRaft 功能门

UseKRaft 功能门 *启用了* 默认状态。

UseKRaft 功能门以 KRaft (Kafka Raft 元数据)模式部署 Kafka 集群，而无需 ZooKeeper。ZooKeeper 和 KRaft 是用于管理 Kafka 集群中元数据和协调操作的机制。KRaft 模式消除了对

ZooKeeper 等外部协调服务的需求。在 KRaft 模式中，Kafka 节点使用代理、控制器或两者的角色。它们一起管理元数据，该元数据在分区之间复制。控制器负责协调操作和维护集群的状态。

使用 UseKRaft 功能门需要启用 KafkaNodePools 功能门。要以 KRaft 模式部署 Kafka 集群，您必须使用 KafkaNodePool 资源。如需了解更多详细信息和示例，请参阅 第 6.3.1 节 “使用节点池部署 Kafka 集群”。使用 KRaft 模式的 Kafka 自定义资源还必须具有注解 strimzi.io/kraft="enabled"。

目前，Apache Kafka Streams 中的 KRaft 模式有以下主要限制：

- KRaft 模式 *只支持 Un* idirectional Topic Operator。不支持 *Bidirectional* Topic Operator，当 UnidirectionalTopicOperator 功能门被禁用时，spec.entityOperator.topicOperator 属性 必须从 Kafka 自定义资源中删除。
- 不支持 JBOD 存储。可以使用 type: jbod 存储，但 JBOD 数组只能包含一个磁盘。
- 不支持扩展 KRaft 只在控制器或缩减的节点。

禁用 UseKRaft 功能门

要禁用 UseKRaft 功能门，在 Cluster Operator 配置中的 STRIMZI_FEATURE_GATES 环境变量中指定 -UseKRaft。

7.2.2. KafkaNodePools 功能门

KafkaNodePools 功能门 *启用了* 默认状态。

KafkaNodePools 功能门引入了一个新的 KafkaNodePool 自定义资源，该资源启用配置不同 Apache Kafka *节点池*。

节点池指的是 Kafka 集群中不同的 Kafka 节点组。每个池都有自己的唯一的配置，其中包括强制设置，如副本数、存储配置和分配的角色列表。您可以将 *控制器* 角色、*代理* 角色或两个角色都分配给 .spec.roles 字段中池中的所有节点。当与基于 ZooKeeper 的 Apache Kafka 集群一起使用时，必须将其设置为 broker 角色。与 UseKRaft 功能门一起使用时，它可以设置为 代理、控制器 或两者。

此外，节点池可以自行配置资源请求和限值、Java JVM 选项和资源模板。没有在 KafkaNodePool 资源中设置的配置选项从 Kafka 自定义资源继承。

KafkaNodePool 资源使用 strimzi.io/cluster 标签来指示它们所属的 Kafka 集群。该标签必须设置为 Kafka 自定义资源的名称。

KafkaNodePool 资源的示例可在 Apache Kafka 的 Streams 提供 的示例配置文件 中找到。

禁用 KafkaNodePools 功能门

要禁用 KafkaNodePools 功能门，在 Cluster Operator 配置中的 STRIMZI_FEATURE_GATES 环境变量中指定 -KafkaNodePools。使用节点池的 Kafka 自定义资源还必须具有注解 strimzi.io/node-pools: enabled。

从 KafkaNodePools 降级

如果您的集群已使用 KafkaNodePool 自定义资源，并且希望降级到 Apache Kafka 的旧版本，不支持它们或禁用 KafkaNodePools 功能门，您必须首先从 KafkaNodePool 自定义资源迁移到只使用 Kafka 自定义资源管理 Kafka 节点。

7.2.3. UnidirectionalTopicOperator 功能门

UnidirectionalTopicOperator 功能门 *启用了* 默认状态。

UnidirectionalTopicOperator 功能门引入了一个单向主题管理模式，用于使用 KafkaTopic 资源创建 Kafka 主题。单向模式与使用 KRaft 进行集群管理兼容。使用单向模式，您可以使用 KafkaTopic 资源创建 Kafka 主题，然后由 Topic Operator 管理。对 KafkaTopic 资源之外的主题的任何配置更改都会被恢复。有关主题管理的详情，请参考 第 10.1 节 “主题管理模式”。

禁用 UnidirectionalTopicOperator 功能门

要禁用 UnidirectionalTopicOperator 功能门，在 Cluster Operator 配置中的 STRIMZI_FEATURE_GATES 环境变量中指定 -UnidirectionalTopicOperator。

7.3. 早期访问功能门(ALPHA)

早期访问功能门还没有达到 beta 阶段，默认是禁用的。早期访问功能门为评估提供了在将其功能永久合并到 Apache Kafka 的 Streams 之前进行评估的机会。

目前，alpha 阶段没有功能门。

7.4. 启用功能门

要修改功能门的默认状态，请在 Operator 配置中使用 STRIMZI_FEATURE_GATES 环境变量。您可以使用这个单一环境变量修改多个功能门。指定以逗号分隔的功能门名称和前缀列表。+ 前缀启用功能门和 - 前缀会禁用它。

启用 FeatureGate1 并禁用 FeatureGate2的功能门配置示例

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: +FeatureGate1,-FeatureGate2
```

7.5. 功能门版本

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)- 通常总是启用

alpha 阶段功能可能是实验性的或不稳定，受更改，或者尚未足够测试以供生产环境使用。Beta 阶段功能经过良好测试，其功能可能不会改变。GA 阶段功能是稳定的，不应在以后有所变化。如果 alpha 和 beta 阶段功能无效，则它们会被移除。

- ControlPlaneListener 功能门在 Apache Kafka 2.3 的 Streams 中移到 GA 阶段。现在，它已被永久启用且无法禁用。
- ServiceAccountPatching 功能门在 Apache Kafka 2.3 的 Streams 中移到 GA 阶段。现在，它已被永久启用且无法禁用。

- UseStrimziPodSets 功能门在 Apache Kafka 2.5 的 Streams 中移到 GA 阶段，对 StatefulSets 的支持会被完全删除。现在，它已被永久启用且无法禁用。

- StableConnectIdentities 功能门在 Apache Kafka 2.7 的 Streams 中移到 GA 阶段。现在，它已被永久启用且无法禁用。

- UseKRaft 功能门处于 beta 阶段，默认是启用的。

- KafkaNodePools 功能门处于 beta 阶段，默认是启用的。

- UnidirectionalTopicOperator 功能门处于 beta 阶段，默认是启用的。

注意

当功能门达到 GA 时，可能会删除它们。这意味着这个功能被整合到 Apache Kafka 核心功能的 Streams 中，且无法再禁用。

表 7.1. 当 Apache Kafka 版本移至 alpha、beta 或 GA 时，功能门和流

功能门	Alpha	beta	GA
ControlPlaneListener	1.8	2.0	2.3
ServiceAccountPatching	1.8	2.0	2.3
UseStrimziPodSets	2.1	2.3	2.5
UseKRaft	2.2	2.7	-
StableConnectIdentities	2.4	2.6	2.7
KafkaNodePools	2.5	2.7	-
UnidirectionalTopicOperator	2.5	2.7	-

如果启用了功能门，您可能需要在从 Apache Kafka 版本的特定流升级或降级前禁用它（或者首先将 / 降级升级到可禁用它的 Apache Kafka 的 Streams 版本）。下表显示了在升级或降级 Apache Kafka 版本的 Streams 时需要禁用哪些功能门。

表 7.2. 在升级或降级 Apache Kafka 的 Streams 时禁用的功能门

禁用功能门	从 Apache Kafka 版本的 Streams 升级	降级到 Apache Kafka 版本的 Streams
ControlPlaneListener	1.7 及更早版本	1.7 及更早版本
UseStrimziPodSets	-	2.0 及更早版本
StableConnectIdentities	-	2.3 及更早版本

第 8 章 迁移到 KRAFT 模式

如果您使用 ZooKeeper 在 Kafka 集群中进行元数据管理，您可以在 KRaft 模式下迁移到使用 Kafka。KRaft 模式替代 ZooKeeper 用于分布式协调，提供增强的可靠性、可扩展性和吞吐量。

在迁移过程中，您将控制器节点的仲裁安装为节点池，该池取代了 ZooKeeper 来管理集群。您可以通过应用 strimzi.io/kraft="migration" 注解在集群配置中启用 KRaft 迁移。迁移完成后，您可以使用 strimzi.io/kraft="enabled" 注解将代理切换为使用 KRaft 和控制器退出迁移模式。

在开始迁移前，请验证您的环境是否可以在 KRaft 模式中支持 Kafka，因为有很多 限制。另请注意，以下内容：

- 迁移只在专用控制器节点上支持，不支持将双角色作为代理和控制器的节点。
- 在整个迁移过程中，ZooKeeper 和 controller 节点在一段时间内并行运行，需要集群中有足够的计算资源。

先决条件

- 您必须在 Kafka 3.7.0 或更新版本中使用 Streams for Apache Kafka 2.7 或更新版本。如果您使用早期版本的 Apache Kafka 或 Apache Kafka，在迁移到 KRaft 模式前升级。
- 验证基于 ZooKeeper 的部署是否正常运行，因为 KRaft 模式不支持它们：
  - 以双向模式运行的主题 Operator。它应该是单向模式或禁用。
  - JBOD 存储。虽然可以使用 jbod 存储类型，但 JBOD 阵列必须仅包含一个磁盘。
- 管理 Kafka 集群的 Cluster Operator 正在运行。
- Kafka 集群部署使用 Kafka 节点池。

  如果您的基于 ZooKeeper 的集群已经使用节点池，则必须迁移。如果没有，您可以 迁移集群以使用节点池。当集群没有使用节点池时，代理必须包含在分配了 代理 角色的 KafkaNodePool

资源配置中，并具有 name kafka。使用 strimzi.io/node-pools: enabled 注解在 Kafka 资源配置中启用了节点池的支持。

在此过程中，Kafka 集群名称是 my-cluster，它位于 my-project 命名空间中。创建的控制器节点池的名称为 controller。代理的节点池称为 kafka。

流程

1. 对于 Kafka 集群，创建带有 控制器 角色的节点池。

   节点池在集群中添加控制器节点的仲裁。

   控制器节点池的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaNodePool
metadata:
  name: controller
  labels:
    strimzi.io/cluster: my-cluster
spec:
  replicas: 3
  roles:
    - controller
  storage:
    type: jbod
    volumes:
      - id: 0
        type: persistent-claim
        size: 20Gi
        deleteClaim: false
    resources:
      requests:
        memory: 64Gi
        cpu: "8"
      limits:
        memory: 64Gi
        cpu: "12"
```

注意

对于迁移，您无法使用共享代理和控制器角色的节点池。

2. 应用新的 KafkaNodePool 资源以创建控制器。

在 Cluster Operator 日志中预期在基于 ZooKeeper 的环境中使用控制器相关的错误。错误可能会阻止协调。要防止这种情况，请立即执行下一步。

3. 通过将 strimzi.io/kraft 注解设置为迁移，在 Kafka 资源中启用 KRaft 迁移 ：

```
oc annotate kafka my-cluster strimzi.io/kraft="migration" --overwrite
```

启用 KRaft 迁移

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: my-project
  annotations:
    strimzi.io/kraft="migration"
# ...
```

将注解应用到 Kafka 资源配置会开始迁移。

4. 检查控制器是否已启动，且代理已推出：

```
oc get pods -n my-project
```

输出显示代理和控制器节点池中的节点

```
NAME                 READY  STATUS   RESTARTS
my-cluster-kafka-0       1/1    Running  0
```

```
my-cluster-kafka-1       1/1    Running  0
my-cluster-kafka-2       1/1    Running  0
my-cluster-controller-3  1/1    Running  0
my-cluster-controller-4  1/1    Running  0
my-cluster-controller-5  1/1    Running  0
# ...
```

5. 检查迁移的状态：

```
oc get kafka my-cluster -n my-project -w
```

更新元数据状态

```
NAME       ...  METADATA STATE
my-cluster  ...  Zookeeper
my-cluster  ...  KRaftMigration
my-cluster  ...  KRaftDualWriting
my-cluster  ...  KRaftPostMigration
```

METADATA STATE 显示用于管理 Kafka 元数据和协调操作的机制。在迁移开始时，这是 ZooKeeper。

- 当元数据仅存储在 ZooKeeper 中时，Zooee 是初始状态。
- KRaftMigration 是迁移正在进行时的状态。启用 ZooKeeper 到 KRaft 迁移 (zookeeper.metadata.migration.enable)的标记被添加到代理中，并使用控制器注册。根据集群中的主题和分区数量，迁移可能需要一些时间。
- KRaftDualWriting 是 Kafka 集群作为 KRaft 集群工作时的状态，但元数据存储在 Kafka 和 ZooKeeper 中。代理会再次回滚，以移除该标志以启用迁移。
- KRaftPostMigration 是为代理启用 KRaft 模式时的状态。元数据仍然存储在 Kafka 和 ZooKeeper 中。

迁移状态也在 Kafka 资源的 status.kafkaMetadataState 属性中表示。

警告

您可以从此时回滚到使用 ZooKeeper。下一步是启用 KRaft。在启用 KRaft 后无法执行回滚。

6. 当元数据状态达到 KRaftPostMigration 时，通过将 strimzi.io/kraft 注解设置为启用，在 Kafka 资源配置中启用 KRaft ：

```
oc annotate kafka my-cluster strimzi.io/kraft="enabled" --overwrite
```

启用 KRaft 迁移

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: my-project
  annotations:
    strimzi.io/kraft="enabled"
# ...
```

7. 检查 move to full KRaft 模式的状态：

```
oc get kafka my-cluster -n my-project -w
```

更新元数据状态

```
NAME       ... METADATA STATE
my-cluster ... Zookeeper
my-cluster ... KRaftMigration
```

```
my-cluster ... KRaftDualWriting
my-cluster ... KRaftPostMigration
my-cluster ... PreKRaft
my-cluster ... KRaft
```

- 当所有 ZooKeeper 相关资源都自动删除时，PreKRaft 就是状态。
- KRaft 是 KRaft 迁移完成后的最终状态（控制器已滚动后）。

注意

根据为 ZooKeeper 配置 deleteClaim 的方式，其持久性卷声明(PVC)和持久性卷(PV)可能无法删除。deleteClaim 指定在卸载集群时是否删除 PVC。默认值为 false。

8. 从 Kafka 资源中删除任何与 ZooKeeper 相关的配置。

   如果存在，您可以删除以下内容：

   - log.message.format.version
   - inter.broker.protocol.version
   - spec.zookeeper prerequisites 属性

   删除 log.message.format.version 和 inter.broker.protocol.version 会导致代理和控制器再次推出。删除 ZooKeeper 属性会删除与 KRaft-operated 集群中存在的 ZooKeeper 配置相关的任何警告信息。

在迁移时执行回滚

在 Kafka 资源中启用 KRaft 完成迁移前，状态已移到 KRaft 状态，您可以执行回滚操作，如下所示：

1. 将 strimzi.io/kraft="rollback" 注解应用到 Kafka 资源以回滚代理。

```
oc annotate kafka my-cluster strimzi.io/kraft="rollback" --overwrite
```

回滚 KRaft 迁移

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: my-project
  annotations:
    strimzi.io/kraft="rollback"
# ...
```

迁移过程必须处于 KRaftPostMigration 状态才能执行此操作。代理会被回滚，以便可以重新连接到 ZooKeeper，状态返回到 KRaftDualWriting。

2. 删除 Controller 节点池：

```
oc delete KafkaNodePool controller -n my-project
```

3. 将 strimzi.io/kraft="disabled" 注解应用到 Kafka 资源，将元数据状态返回到 ZooKeeper。

```
oc annotate kafka my-cluster strimzi.io/kraft="disabled" --overwrite
```

切换回使用 ZooKeeper

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: my-project
  annotations:
    strimzi.io/kraft="disabled"
# ...
```

第 9 章 配置部署

配置和管理 Apache Kafka 部署的流，以满足您的 Apache Kafka 自定义资源的精确需求。Apache Kafka 的 Streams 提供了每个发行版本的示例自定义资源，允许您配置和创建支持的 Kafka 组件实例。通过将自定义资源配置为根据您的特定要求包含其他功能来微调部署。对于 Kafka Connect 连接器的特定配置区域，如指标、日志记录和外部配置，您还可以使用 ConfigMap 资源。通过使用 ConfigMap 资源来融合配置，您可以集中维护。您还可以使用配置供应商从外部来源加载配置，我们建议提供 Kafka Connect 连接器配置的凭证。

使用自定义资源配置并创建以下组件的实例：

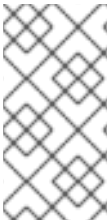
- Kafka 集群
- Kafka Connect 集群
- Kafka MirrorMaker
- Kafka Bridge
- Sything Control

您还可以使用自定义资源配置来管理实例或修改部署，以引入其他功能。这可能包括支持以下配置：

- 指定节点池
- 保护对 Kafka 代理的客户端访问
- 从集群外部访问 Kafka 代理
- 创建主题

- 创建用户（客户端）
- 控制功能门
- 更改日志频率
- 分配资源限值和请求
- 引入功能，如 Apache Kafka Drain Cleaner、Cruise Control 或 distributed tracing 等功能。

Apache Kafka 自定义资源 API Reference 的 Streams 描述了您可以在配置中使用的属性。



注意

应用到自定义资源的标签也会应用到组成集群的 OpenShift 资源。这为资源提供了一个方便的机制来根据需要进行标记。

将更改应用到自定义资源配置文件

您可以使用 spec 属性将配置添加到自定义资源中。添加配置后，您可以使用 oc 将更改应用到自定义资源配置文件：

```
oc apply -f <kafka_configuration_file>
```

9.1. 使用示例配置文件

通过纳入其他支持的配置来进一步增强部署。从 Apache Kafka 软件下载页面 的 Streams 中提供了可下载发行工件的示例。

默认情况下，示例文件仅包含自定义资源的基本属性和值。您可以使用 oc 命令行工具下载并应用示例。为部署构建您自己的 Kafka 组件配置时，示例可以作为一个起点。

注意

如果使用 Operator 安装 Apache Kafka 的 Streams，您仍然可以下载示例文件，并使用它们上传配置。

发行工件包括 一个示例 目录，其中包含配置示例。

为 Apache Kafka 提供的流配置文件示例

```
examples
├── user 1
├── topic 2
├── security 3
│   ├── tls-auth
│   ├── scram-sha-512-auth
│   └── keycloak-authorization
├── mirror-maker 4
├── metrics 5
├── kafka 6
│   └── nodepools 7
├── cruise-control 8
├── connect 9
└── bridge 10
```

1

KafkaUser 自定义资源配置，由 User Operator 管理。

2

KafkaTopic 自定义资源配置，由 Topic Operator 管理。

3

Kafka 组件的身份验证和授权配置。包括 TLS 和 SCRAM-SHA-512 身份验证的示例配置。Red Hat Single Sign-On 示例包括 Kafka 自定义资源配置和 Red Hat Single Sign-On 域规格。您可以使用示例尝试 Red Hat Single Sign-On 授权服务。另外，还有一个启用了 oauth 身份验证和 keycloak 授权指标的示例。

4

用于部署 Mirror Maker 的 Kafka 自定义资源配置。包括复制策略和同步频率配置示例。

5

指标配置，包括 Prometheus 安装和 Grafana 仪表板文件。

6

用于部署 Kafka 的 Kafka 自定义资源配置。包括临时或持久单一或多节点部署的示例配置。

7

Kafka 集群中 Kafka 节点的 KafkaNodePool 配置。包括使用 KRaft (Kafka Raft metadata)模式或 ZooKeeper 的集群中节点配置示例。

8

使用 Cruise Control 的部署配置的 Kafka 自定义资源。包含 KafkaRebalance 自定义资源，从 Cruise Control 生成优化提议，以及示例配置，以使用默认或用户优化目标。

9

用于部署 Kafka Connect 的 KafkaConnect 和 KafkaConnector 自定义资源配置。包括单节点或多节点部署的配置示例。

10

KafkaBridge 自定义资源配置用于部署 Kafka Bridge。

9.2. 配置 KAFKA

更新 Kafka 自定义资源的 spec 属性来配置 Kafka 部署。

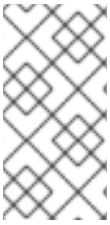
另外，您还可以为 ZooKeeper 和 Apache Kafka Operator 添加配置。各个组件都独立配置通用配置属性，如日志记录和健康检查。

特别重要的配置选项包括：

- 资源请求(CPU / Memory)

- 最多和最小内存分配的 JVM 选项
- 将客户端连接到 Kafka 代理的监听程序（以及客户端的身份验证）
- 身份验证
- 存储
- 机架感知
- 指标
- 用于集群重新平衡的 Cruise Control
- 基于 KRaft 的 Kafka 集群的元数据版本
- 基于 ZooKeeper 的 Kafka 集群的 inter-broker 协议版本

.spec.kafka.metadataVersion 属性或 config 中的 inter.broker.protocol.version 属性必须是指定的 Kafka 版本(spec.kafka.version)支持的版本。属性代表 Kafka 集群中使用的 Kafka 元数据或 inter-broker 协议版本。如果没有在配置中设置这些属性，Cluster Operator 会将版本更新为使用的 Kafka 版本的默认值。



注意

最旧的支持的元数据版本为 3.3。使用早于 Kafka 版本的元数据版本可能会导致禁用一些功能。

要深入了解 Kafka 集群配置选项，请参阅 Apache Kafka 自定义资源 API 参考。

管理 TLS 证书

在部署 Kafka 时，Cluster Operator 会自动设置和更新 TLS 证书，以便在集群中启用加密和身份验证。如果需要，您可以在续订周期开始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 手动续订 CA 证书和替换私钥。

Kafka 自定义资源配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3 (1)
    version: 3.7.0 (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: external1 (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
    default.replication.factor: 3
    min.insync.replicas: 2
    inter.broker.protocol.version: "3.7"
  storage: 19
    type: persistent-claim 20
    size: 10000Gi
  rack: 21
    topologyKey: topology.kubernetes.io/zone
  metricsConfig: 22
    type: jmxPrometheusExporter
    valueFrom:
      configMapKeyRef: 23
        name: my-config-map
        key: my-key
  # ...
zookeeper: 24
  replicas: 3 25
  logging: 26
    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: 27
  tlsSidecar: 28
    resources:
```

```
      requests:
        cpu: 200m
        memory: 64Mi
      limits:
        cpu: 500m
        memory: 128Mi
  topicOperator:
    watchedNamespace: my-topic-namespace
    reconciliationIntervalSeconds: 60
    logging: 29
      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: 30
      type: inline
      loggers:
        rootLogger.level: INFO
    resources:
      requests:
        memory: 512Mi
        cpu: "1"
      limits:
        memory: 512Mi
        cpu: "1"
kafkaExporter: 31
  # ...
cruiseControl: 32
  # ...
```

1

副本节点的数量。

2

Kafka 版本，可按照升级步骤将其改为受支持的版本。

3

Kafka 日志记录器和日志级别直接(内联)或通过 ConfigMap 间接添加(外部)。自定义 Log4j 配置必须放在 ConfigMap 中的 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

为保留支持的资源（当前 cpu 和 memory ）的请求，以及指定可消耗的最大资源的限制。

5

检查检查以了解何时重启容器（存活度）以及何时容器可以接受流量（就绪度）。

6

JVM 配置选项，用于优化运行 Kafka 的虚拟机(VM)的性能。

7

ProductShortName OPTION: 容器镜像配置，仅在特殊情况下推荐使用。

8

侦听器配置客户端如何通过 bootstrap 地址连接到 Kafka 集群。监听器被配置为 *internal* 或 *external* 监听程序，用于来自 OpenShift 集群内部或外的连接。

9

用于标识监听程序的名称。在 Kafka 集群中必须是唯一的。

10

Kafka 中监听器使用的端口号。端口号必须在给定的 Kafka 集群中唯一。允许端口号为 9092 及更高，但端口 9404 和 9999 除外，它们已经用于 Prometheus 和 JMX。根据监听器类型，端口号可能与连接 Kafka 客户端的端口号不同。

11

监听程序指定为 internal 或 cluster-ip (使用每个代理的 ClusterIP 服务公开 Kafka)，或为外部监听程序指定为 route（只限 OpenShift），loadbalancer, nodeport 或 ingress（只限 Kubernetes）。

12

启用或禁用每个监听器的 TLS 加密。对于 route 和 ingress 类型监听程序，必须通过将其设置为 true 来启用 TLS 加密。

13

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

14

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

15

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

16

由外部 CA （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。您可以在任何启用了 TLS 加密的监听程序中配置 Kafka 侦听器证书。

17

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

18

代理配置。标准 Apache Kafka 配置可能会提供，仅限于不直接由 Apache Kafka 的 Streams 管理的属性。

19

持久性卷的存储大小可能会增加，并可以添加额外的卷到 JBOD 存储中。

20

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。

21

机架感知配置，将副本分散到不同的机架、数据中心或可用性区域。topologyKey 必须与包含机架 ID 的节点标签匹配。此配置中使用的示例使用标准 topology.kubernetes.io/zone 标签指定区。

22

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

23 通过 Prometheus JMX Exporter 将指标数据导出到 Grafana 仪表板的规则，通过引用包含 Prometheus JMX exporter 配置的 ConfigMap 来启用。您可以使用对 metricsConfig.valueFrom.configMapKeyRef.key 下包含空文件的 ConfigMap 的引用来启用指标。

24 特定于 ZooKeeper 的配置，其中包含与 Kafka 配置类似的属性。

25 ZooKeeper 节点数量。ZooKeeper 集群通常有奇数个节点，一般为三个、五个或七个。大多数节点都必须可用，才能保持有效的仲裁。如果 ZooKeeper 集群丢失了其仲裁，它将停止响应客户端，并且 Kafka 代理将停止工作。拥有稳定且高度可用的 ZooKeeper 集群对于 Apache Kafka 来说至关重要。

26 ZooKeeper 日志记录器和日志级别。

27 Entity Operator 配置，用于指定主题 Operator 和 User Operator 的配置。

28 实体 Operator TLS sidecar 配置。实体 Operator 使用 TLS sidecar 与 ZooKeeper 安全通信。

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

30 指定 User Operator 日志记录器和日志级别。

31 Kafka 导出程序配置。Kafka Exporter 是一个可选组件，用于在特定消费者滞后数据中从 Kafka 代理中提取指标数据。要使 Kafka Exporter 能够正常工作，需要使用消费者组。

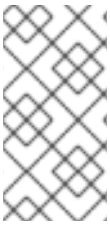
32 Cruise Control 的可选配置，用于重新平衡 Kafka 集群。

9.2.1. 使用 Kafka 静态配额插件对代理设置限制

使用 *Kafka 静态配额*插件在 Kafka 集群中的代理上设置吞吐量和存储限制。您可以通过配置 Kafka 资源来启用插件和设置限制。您可以设置字节阈值和存储配额，以在与代理交互的客户端上放置限制。

您可以为生成者和消费者带宽设置字节阈值。总限制分布在访问代理的所有客户端中。例如，您可以为制作者设置 40 MBps 的字节阈值。如果两个制作者正在运行，则它们各自限制为 20 MBps 的吞吐量。

存储配额在软限制和硬限制之间节流 Kafka 磁盘存储限制。限制适用于所有可用磁盘空间。生产者在软限制和硬限制之间逐渐减慢。限制可防止磁盘填满速度超过其容量。完整磁盘可能会导致出现问题。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，另一个磁盘将大约为空。

先决条件

- 管理 Kafka 集群的 Cluster Operator 正在运行。

流程

1. 为 Kafka 资源的 config 添加插件属性。

   本例中显示了插件属性。

   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 Static Quota 插件。
2. 设置制作者字节阈值。本示例中为 1 Mbps.
3. 设置消费者字节阈值。本示例中为 1 Mbps.
4. 为存储设置较低软限制。在本示例中为 400 GB。
5. 为存储设置更高的硬限制。本例中为 500 GB。
6. 设置存储检查间隔（以秒为单位）。本示例中为 5 秒。您可以将其设置为 0 来禁用检查。

2. 更新资源。

```
oc apply -f <kafka_configuration_file>
```

其他资源

- KafkaUserQuotas 模式参考

9.2.2. 默认 ZooKeeper 配置值

当使用 Apache Kafka 的 Streams 部署 ZooKeeper 时，Apache Kafka 的 Streams 设置的一些默认配置与标准 ZooKeeper 默认值不同。这是因为 Apache Kafka 的 Streams 设置多个 ZooKeeper 属性，其值在 OpenShift 环境中运行 ZooKeeper。

Apache Kafka Streams 中密钥 ZooKeeper 属性的默认配置如下：

表 9.1. Apache Kafka 的 Streams 中的默认 ZooKeeper 属性

属性	默认值	Description
tickTime	2000	以毫秒为单位的单空长度，它决定了会话超时的长度。
initLimit	5	在 ZooKeeper 集群中允许后续者获得的最大 tick 数量。
syncLimit	2	后续允许不与 ZooKeeper 集群中的领导不同步的最大 tick 数量。
autopurge.purgeInterval	1	启用 **autopurge** 功能，并在小时内设置清除服务器端 ZooKeeper 事务日志的时间间隔。
admin.enableServer	false	禁用 ZooKeeper admin 服务器的标记。在 Apache Kafka 中，流不使用 admin 服务器。

重要

修改这些默认值作为 Kafka 自定义资源中的 zookeeper.config 可能会影响 ZooKeeper 集群的行为和性能。

9.2.3. 使用注解删除 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 （如果集群使用持久性存储部署）。删除后，Pod 及其相关的 PersistentVolumeClaim 会自动重新创建。

警告

删除 PersistentVolumeClaim 可能会导致持久性数据丢失，且不能保证集群可用。只有在遇到存储问题时才应执行以下步骤。

先决条件

- 正在运行的 Cluster Operator

流程

1. 查找您要删除的 Pod 的名称。

   Kafka 代理 pod 名为 <cluster_name>-kafka-<index_number>，其中<index_number> 从 0 开始，以总副本数减一结束。例如，my-cluster-kafka-0。

2. 使用 oc annotate 注解 OpenShift 中的 Pod 资源：

   ```
   oc annotate pod <cluster_name>-kafka-<index_number> strimzi.io/delete-pod-and-pvc="true"
   ```

3. 等待下一个协调，当注解的 pod 带有底层持久性卷声明的 pod 将被删除，然后重新创建。

9.2.4. 使用注解删除 ZooKeeper 节点

此流程描述了如何使用 OpenShift 注解删除现有 ZooKeeper 节点。删除 ZooKeeper 节点包括删除运行 ZooKeeper 的 Pod 和相关的 PersistentVolumeClaim （如果集群使用持久性存储部署）。删除后，Pod 及其相关的 PersistentVolumeClaim 会自动重新创建。

警告

删除 PersistentVolumeClaim 可能会导致持久性数据丢失，且不能保证集群可用。只有在遇到存储问题时才应执行以下步骤。

先决条件

- 正在运行的 Cluster Operator

流程

1. 查找您要删除的 Pod 的名称。

   ZooKeeper pod 名为 <cluster_name>-zookeeper-<index_number>，其中<index_number> 从 0 开始，以总副本数减一结束。例如，my-cluster-zookeeper-0。

2. 使用 oc annotate 注解 OpenShift 中的 Pod 资源：

   ```
   oc annotate pod <cluster_name>-zookeeper-<index_number> strimzi.io/delete-pod-and-pvc="true"
   ```

3. 等待下一个协调，当注解的 pod 带有底层持久性卷声明的 pod 将被删除，然后重新创建。

9.3. 配置节点池

更新 KafkaNodePool 自定义资源的 spec 属性来配置节点池部署。

节点池指的是 Kafka 集群中不同的 Kafka 节点组。每个池都有自己的唯一的配置，其中包括副本、角色和存储分配数量的强制设置。

另外，您还可以为以下属性指定值：

- 指定内存和 cpu 请求和限值 的资源
- 模板 为 Pod 和其他 OpenShift 资源指定自定义配置
- jvmOptions，用于指定堆大小、运行时和其他选项的自定义 JVM 配置

Kafka 资源代表 Kafka 集群中所有节点的配置。KafkaNodePool 资源仅代表节点池中的节点的配置。如果没有在 KafkaNodePool 中指定配置属性，它将继承自 Kafka 资源。如果在两个资源中设置，则 KafkaNodePool 资源中指定的配置具有优先权。例如，如果节点池和 Kafka 配置都包含 jvmOptions，则使用节点池配置中指定的值。当在 KafkaNodePool.spec.jvmOptions 中设置了 -Xmx: 1024m，在 Kafka.spec.kafka.jvmOptions 中设置了 -Xms: 512m，节点会使用其节点池配置中的值。

Kafka 和 KafkaNodePool 模式的属性不会被合并。为了说明，如果 KafkaNodePool.spec.template 只包含 podSet.metadata.labels，并且 Kafka.spec.kafka.template 包含 podSet.metadata.annotations 和 pod.metadata.labels，则 Kafka 配置中的模板值会被忽略，因为节点池配置中有一个模板值。

要深入了解节点池配置选项，请参阅 Apache Kafka 自定义资源 API 参考。

使用 KRaft 模式的集群中节点池的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaNodePool
metadata:
  name: kraft-dual-role 1
  labels:
    strimzi.io/cluster: my-cluster 2
spec:
  replicas: 3 3
  roles: 4
    - controller
    - broker
  storage: 5
    type: jbod
    volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
  resources: 6
      requests:
```

```
      memory: 64Gi
      cpu: "8"
    limits:
      memory: 64Gi
      cpu: "12"
```

1

节点池的唯一名称。

2

节点池所属的 Kafka 集群。节点池只能属于单个集群。

3

节点的副本数。

4

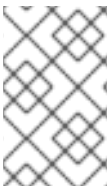
节点池中的节点的角色。在本例中，节点具有双角色作为控制器和代理。

5

节点的存储规格。

6

为保留支持的资源（当前 cpu 和 memory ）的请求，以及指定可消耗的最大资源的限制。



注意

Kafka 资源的配置必须适合 KRaft 模式。目前，KRaft 模式有很多限制。

使用 ZooKeeper 在集群中节点池的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaNodePool
metadata:
  name: pool-a
```

```
  labels:
    strimzi.io/cluster: my-cluster
spec:
  replicas: 3
  roles:
    - broker 1
  storage:
    type: jbod
    volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
  resources:
      requests:
        memory: 64Gi
        cpu: "8"
      limits:
        memory: 64Gi
        cpu: "12"
```

1

节点池中的节点的角色，只有使用带有 ZooKeeper 的 Kafka 时，才可以 代理。

9.3.1. 为节点池分配 ID 以进行扩展操作

此流程描述了如何在节点池中执行扩展操作时，Cluster Operator 对高级节点 ID 处理使用注解。您可以按顺序使用下一个 ID 来指定要使用的节点 ID，而不是 Cluster Operator。以这种方式管理节点 ID 可提供更多控制。

要添加一系列 ID，您可以为 KafkaNodePool 资源分配以下注解：

- strimzi.io/next-node-ids 来添加用于新代理的 ID 范围
- strimzi.io/remove-node-ids 来添加一系列 ID 以删除现有代理

您可以指定单个节点 ID、ID 范围或两者的组合。例如，您可以指定以下 ID 范围： [0, 1, 2, 10-20, 30] 来扩展 Kafka 节点池。通过这种格式，您可以指定单个节点 ID (0、1、2、30)以及一系列 ID (10-20)。

在典型的场景中，您可以指定一个 ID 范围来向上扩展和单一节点 ID，以便在缩减时删除特定节点。

在此过程中，我们将扩展注解添加到节点池中，如下所示：

- 为 pool-a 分配一系列 ID 进行扩展
- 为 pool-b 分配一系列 ID 用于缩减

在扩展操作过程中，使用 ID，如下所示：

- 扩展获取新节点范围内的最低可用 ID。
- 缩减会删除范围中具有最高可用 ID 的节点。

如果在节点池中分配的节点 ID 序列中有差距，则要添加的下一个节点会被分配一个填充空白的 ID。

每次扩展操作后，不需要更新注解。任何未使用的 ID 仍对下一个扩展事件有效。

Cluster Operator 允许您以升序或降序指定 ID 范围，以便您可以按照节点扩展的顺序定义它们。例如，在扩展时，您可以指定一个范围，如 [1000-1999]，并且新节点分配下一个最低 ID：1000、1001、1002、1003 等。相反，当缩减时，您可以指定一个范围，如 [1999-1000]，确保具有下一个最高 ID 的节点被删除 ：1003, 1002, 1001, 1000 等。

如果没有使用注解指定 ID 范围，Cluster Operator 遵循其在扩展操作期间处理 ID 的默认行为。节点 ID 以 0 （零）开始，并在 Kafka 集群中按顺序运行。下一个最低 ID 分配给新节点。在集群中填充节点 ID 的差距。这意味着它们可能无法在节点池中按顺序运行。扩展的默认行为是在集群中添加下一个最低可用节点 ID；在缩减时，它会删除具有最高可用节点 ID 的节点。如果分配的 ID 范围被误格式化，则扩展范围没有 ID，或者缩减范围不适用于任何正在使用的节点，则应用默认方法。

先决条件

- 必须部署 Cluster Operator。

- （可选）使用 reserved.broker-max.id 配置属性来扩展节点池中节点 ID 的允许范围。

默认情况下，Apache Kafka 将节点 ID 限制为范围从 0 到 999。要使用大于 999 的节点 ID 值，请将 reserved.broker-max.id 配置属性添加到 Kafka 自定义资源中，并指定所需的最大节点 ID 值。

在本例中，最大节点 ID 设置为 10000。然后，节点 ID 可以分配给该值。

最大节点 ID 数量配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    config:
      reserved.broker.max.id: 10000
  # ...
```

流程

1. 使用 ID 为节点池添加在扩展或缩减时要使用的 ID，如下例所示。

   用于向上扩展的 ID 会被分配给节点 池池 ：

   为扩展分配 ID

   ```
   oc annotate kafkanodepool pool-a strimzi.io/next-node-ids="[0,1,2,10-20,30]"
   ```

   将节点添加到 池 时使用此范围内的最低可用 ID。

用于缩减的 ID 分配给节点池 pool-b ：

为缩减分配 ID

```
oc annotate kafkanodepool pool-b strimzi.io/remove-node-ids="[60-50,9,8,7]"
```

缩减 pool-b 时会删除此范围内的最高可用 ID。

注意

如果要删除特定节点，您可以为缩减注解分配一个单一节点 ID： oc annotate kafkanodepool pool-b strimzi.io/remove-node-ids="[3]"。

2. 现在，您可以扩展节点池。

   如需更多信息，请参阅以下：

   - 第 9.3.3 节 “将节点添加到节点池中”
   - 第 9.3.4 节 “从节点池中删除节点”
   - 第 9.3.5 节 “在节点池之间移动节点”

   在协调时，如果注解被错误格式化，则会提供一个警告。

3. 执行扩展操作后，如果需要，可以删除注解。

   删除扩展注解

```
oc annotate kafkanodepool pool-a strimzi.io/next-node-ids-
```

删除缩减注解

```
oc annotate kafkanodepool pool-b strimzi.io/remove-node-ids-
```

9.3.2. 将节点从节点池移动时对机架的影响

如果在 Kafka 集群中启用了机架感知，副本可以分散到不同的机架、数据中心或可用区中。当将节点从节点池中移动时，请考虑对集群拓扑的影响，特别是对机架感知的影响。从节点池中删除特定的 pod，特别是没有顺序，可能会破坏集群拓扑，或者导致在机架间分布不平衡。imbalance 可能会影响节点本身和集群中的分区副本的分布。跨机架的节点和分区均匀分布可能会影响 Kafka 集群的性能和弹性。

规划以战略方式删除节点，以保持跨机架所需的平衡和弹性。使用 strimzi.io/remove-node-ids 注解来谨慎移动具有特定 ID 的节点。确保将分区副本分散到机架之间，并且客户端若要从最接近的副本使用的配置不会中断。

提示

使用 Cruise Control 和带有 RackAwareGoal 的 KafkaRebalance 资源，以确保副本在不同机架之间保持分布。

9.3.3. 将节点添加到节点池中

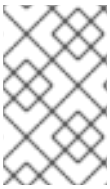
这个步骤描述了如何扩展节点池来添加新节点。目前，只有包含作为专用代理运行的节点的代理节点池才可以扩展。

在此过程中，我们从三个节点池(池)开始：

节点池中的 Kafka 节点

```
NAME                READY  STATUS   RESTARTS
my-cluster-pool-a-0  1/1    Running  0
my-cluster-pool-a-1  1/1    Running  0
my-cluster-pool-a-2  1/1    Running  0
```

节点 ID 在创建时附加到节点的名称中。我们添加节点 my-cluster-pool-a-3，节点 ID 为 3。



注意

在此过程中，保存分区副本的节点 ID 会改变。考虑引用节点 ID 的任何依赖项。

先决条件

- 必须部署 Cluster Operator。
- Cruise Control 使用 Kafka 部署。
- （可选）要扩展操作，您可以指定要在操作中使用的节点 ID。

如果您为操作分配了一系列节点 ID，正在添加的节点 ID 由给定的节点序列决定。如果您分配了单一节点 ID，则使用指定 ID 添加节点。否则，会使用集群中的最低可用节点 ID。

流程

1. 在节点池中创建新节点。

   例如，节点池 pool-a 有三个副本。我们通过增加副本数量来添加节点：

   ```
   oc scale kafkanodepool pool-a --replicas=4
   ```

2. 检查部署的状态，并等待节点池中的 pod 创建并就绪(1/1)。

```
oc get pods -n <my_cluster_operator_namespace>
```

输出显示节点池中的四个 Kafka 节点

```
NAME                 READY  STATUS   RESTARTS
my-cluster-pool-a-0  1/1    Running  0
my-cluster-pool-a-1  1/1    Running  0
my-cluster-pool-a-2  1/1    Running  0
my-cluster-pool-a-3  1/1    Running  0
```

3. 在增加节点池中的节点数后重新分配分区。

在扩展节点池后，使用 Cruise Control add-brokers 模式将分区副本从现有代理移到新添加的代理中。

使用 Cruise Control 重新分配分区副本

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
  # ...
spec:
  mode: add-brokers
  brokers: [3]
```

我们正在将分区重新分配给节点 my-cluster-pool-a-3。根据集群中的主题和分区数量，重新分配可能需要一些时间。

9.3.4. 从节点池中删除节点

这个步骤描述了如何缩减节点池来删除节点。目前，只有包含作为专用代理运行的节点的代理节点池才可以缩减。

在此过程中，我们从四个节点池(池)开始：

节点池中的 Kafka 节点

```
NAME              READY  STATUS   RESTARTS
my-cluster-pool-a-0  1/1    Running  0
my-cluster-pool-a-1  1/1    Running  0
my-cluster-pool-a-2  1/1    Running  0
my-cluster-pool-a-3  1/1    Running  0
```

节点 ID 在创建时附加到节点的名称中。我们删除节点 my-cluster-pool-a-3，节点 ID 为 3。

注意

在此过程中，保存分区副本的节点 ID 会改变。考虑引用节点 ID 的任何依赖项。

先决条件

- 必须部署 Cluster Operator。
- Cruise Control 使用 Kafka 部署。
- （可选）为了缩减操作，您可以指定要在操作中使用的节点 ID。

如果您为操作分配了一系列节点 ID，则要删除的节点的 ID 由给定的节点序列决定。如果您分配了单一节点 ID，则将删除具有指定 ID 的节点。否则，节点池中具有最高可用 ID 的节点会被删除。

流程

1. 在减少节点池中的节点数前重新分配分区。

在缩减节点池前，请使用 Cruise Control remove-brokers 模式，将分区副本从要删除的代理中移出。

使用 Cruise Control 重新分配分区副本

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
  # ...
spec:
  mode: remove-brokers
  brokers: [3]
```

我们从节点 my-cluster-pool-a-3 重新分配分区。根据集群中的主题和分区数量，重新分配可能需要一些时间。

2. 在重新分配过程完成后，删除的节点没有实时分区，从而减少节点池中的 Kafka 节点数量。

例如，节点池 pool-a 有四个副本。我们通过减少副本数来删除节点：

```
oc scale kafkanodepool pool-a --replicas=3
```

输出显示节点池中的三个 Kafka 节点

```
NAME                    READY  STATUS   RESTARTS
my-cluster-pool-b-kafka-0  1/1    Running  0
my-cluster-pool-b-kafka-1  1/1    Running  0
my-cluster-pool-b-kafka-2  1/1    Running  0
```

9.3.5. 在节点池之间移动节点

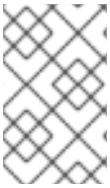
这个步骤描述了如何在不停机的情况下在源和目标 Kafka 节点池之间移动节点。您可以在目标节点池中创建新节点并重新分配分区，以便从源节点池中的旧节点移动数据。当新节点上的副本同步时，您可以删除旧节点。

在此过程中，我们从两个节点池开始：

- 具有三个副本的池 是目标节点池
- 带有四个副本的 pool-b 是源节点池

我们扩展 pool-a，并重新分配分区并缩减 pool-b，这会导致以下内容：

- pool-a 带有四个副本
- 带有三个副本的 pool-b



注意

在此过程中，保存分区副本的节点 ID 会改变。考虑引用节点 ID 的任何依赖项。

先决条件

- 必须部署 Cluster Operator。
- Cruise Control 使用 Kafka 部署。
- （可选）要扩展和缩减操作，您可以指定要使用的节点 ID 范围。

如果您为操作分配了节点 ID，正在添加或删除的节点 ID 由给定的节点序列决定。否则，在添加节点时会使用集群中可用的最低节点 ID；并删除节点池中可用 ID 的节点。

流程

1. 在目标节点池中创建新节点。

例如，节点池 pool-a 有三个副本。我们通过增加副本数量来添加节点：

```
oc scale kafkanodepool pool-a --replicas=4
```

2. 检查部署的状态，并等待节点池中的 pod 创建并就绪(1/1)。

```
oc get pods -n <my_cluster_operator_namespace>
```

输出显示源和目标节点池中的四个 Kafka 节点

```
NAME              READY  STATUS   RESTARTS
my-cluster-pool-a-0  1/1    Running  0
my-cluster-pool-a-1  1/1    Running  0
my-cluster-pool-a-4  1/1    Running  0
my-cluster-pool-a-7  1/1    Running  0
my-cluster-pool-b-2  1/1    Running  0
my-cluster-pool-b-3  1/1    Running  0
my-cluster-pool-b-5  1/1    Running  0
my-cluster-pool-b-6  1/1    Running  0
```

节点 ID 在创建时附加到节点的名称中。我们添加节点 my-cluster-pool-a-7，节点 ID 为 7。

3. 将分区从旧节点重新分配给新节点。

在缩减源节点池前，请使用 Cruise Control remove-brokers 模式，将分区副本从要删除的代理中移出。

使用 Cruise Control 重新分配分区副本

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
  # ...
spec:
  mode: remove-brokers
  brokers: [6]
```

我们从节点 my-cluster-pool-b-6 重新分配分区。根据集群中的主题和分区数量，重新分配可能需要一些时间。

4. 重新分配过程完成后，减少源节点池中的 Kafka 节点数量。

例如，节点池 pool-b 具有四个副本。我们通过减少副本数来删除节点：

```
oc scale kafkanodepool pool-b --replicas=3
```

池中具有最高 ID(6)的节点会被删除。

输出显示源节点池中的三个 Kafka 节点

```
NAME                    READY  STATUS   RESTARTS
my-cluster-pool-b-kafka-2  1/1    Running  0
my-cluster-pool-b-kafka-3  1/1    Running  0
my-cluster-pool-b-kafka-5  1/1    Running  0
```

9.3.6. 更改节点池角色

节点池可用于以 KRaft 模式（使用 Kafka Raft 元数据）操作的 Kafka 集群，或使用 ZooKeeper 进行元数据管理。如果使用 KRaft 模式，您可以为节点池中的所有节点指定角色，以作为代理、控制器或两者运行。如果使用 ZooKeeper，则节点必须设置为代理。

在某些情况下，您可能想要更改分配给节点池的角色。例如，您可能有一个节点池，其中包含执行双代理和控制器角色的节点，然后决定将角色拆分在两个节点池之间。在这种情况下，您可以使用作为代理的节点创建新节点池，然后将分区从 dual-role 节点重新分配给新代理。然后，您可以将旧节点池切换到仅限控制器的角色。

您还可以通过从带有 controller-only 和 broker-only 角色的节点池移到包含执行双代理和控制器角色的节点池，来执行反向操作。在这种情况下，您可以将 broker 角色添加到现有的 controller-only 节点池中，将仅代理节点中的分区重新分配给 dual-role 节点，然后删除仅代理节点池。

在节点池配置中删除 代理 角色时，请记住 Kafka 不会自动重新分配分区。在删除代理角色前，请确保

更改为 controller-only 角色的节点没有任何分配的分区。如果分配了分区，则阻止更改。在删除代理角色前，节点上的副本不能保留。在更改角色前重新分配分区的最佳方法是在 remove-brokers 模式中应用 Cruise Control 优化建议。如需更多信息，请参阅 第 19.6 节 “生成优化建议”。

9.3.7. 过渡到单独的代理和控制器角色

此流程描述了如何过渡到使用带有独立角色的节点池。如果您的 Kafka 集群使用带有控制器和代理角色的节点池，您可以过渡到使用具有独立角色的两个节点池。要做到这一点，请重新平衡集群，将分区副本移到具有仅代理角色的节点池中，然后将旧节点池切换到仅限控制器的角色。

在此过程中，我们从节点池 pool-a 开始，它具有 controller 和 broker 角色：

dual-role 节点池

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaNodePool
metadata:
  name: pool-a
  labels:
    strimzi.io/cluster: my-cluster
spec:
  replicas: 3
  roles:
    - controller
    - broker
  storage:
    type: jbod
    volumes:
      - id: 0
        type: persistent-claim
        size: 20Gi
        deleteClaim: false
  # ...
```

节点池有三个节点：

节点池中的 Kafka 节点

```
NAME            READY  STATUS   RESTARTS
```

```
my-cluster-pool-a-0  1/1    Running  0
my-cluster-pool-a-1  1/1    Running  0
my-cluster-pool-a-2  1/1    Running  0
```

每个节点都执行代理和控制器的组合角色。我们创建一个名为 pool-b 的第二个节点池，它有三个节点仅充当代理。

注意

在此过程中，保存分区副本的节点 ID 会改变。考虑引用节点 ID 的任何依赖项。

先决条件

- 必须部署 Cluster Operator。
- Cruise Control 使用 Kafka 部署。

流程

1. 使用 代理 角色创建节点池。

节点池配置示例

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

```
      size: 100Gi
      deleteClaim: false
# ...
```

新节点池也有三个节点。如果您已经有一个仅限代理的节点池，您可以跳过这一步。

2. 应用新的 KafkaNodePool 资源以创建代理。

3. 检查部署的状态，并等待节点池中的 pod 创建并就绪(1/1)。

```
oc get pods -n <my_cluster_operator_namespace>
```

输出显示在两个节点池中运行的 pod

```
NAME              READY  STATUS   RESTARTS
my-cluster-pool-a-0  1/1    Running  0
my-cluster-pool-a-1  1/1    Running  0
my-cluster-pool-a-2  1/1    Running  0
my-cluster-pool-b-3  1/1    Running  0
my-cluster-pool-b-4  1/1    Running  0
my-cluster-pool-b-5  1/1    Running  0
```

节点 ID 在创建时附加到节点的名称中。

4. 使用 Cruise Control remove-brokers 模式，将 dual-role 节点的分区副本重新分配给新添加的代理。

使用 Cruise Control 重新分配分区副本

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
```

```
  # ...
spec:
  mode: remove-brokers
  brokers: [0, 1, 2]
```

根据集群中的主题和分区数量，重新分配可能需要一些时间。

注意

如果更改为仅控制器角色的节点具有任何分配的分区，则阻止更改。Kafka 资源的 status.conditions 提供了防止更改的事件详情。

5. 从最初具有组合角色的节点池中删除 代理 角色。

双角色节点切换到控制器

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaNodePool
metadata:
  name: pool-a
  labels:
    strimzi.io/cluster: my-cluster
spec:
  replicas: 3
  roles:
    - controller
  storage:
    type: jbod
    volumes:
      - id: 0
        type: persistent-claim
        size: 20Gi
        deleteClaim: false
  # ...
```

6. 应用配置更改，以便节点池切换到仅限控制器的角色。

9.3.8. 过渡到双角色节点

此流程描述了如何从带有仅代理和仅控制器角色的独立节点池过渡到使用双角色节点池。如果您的 Kafka 集群使用带有专用控制器和代理节点的节点池，您可以使用带这两个角色的单一节点池过渡到。要做到这一点，将 broker 角色添加到仅 controller-only 节点池中，重新平衡集群，将分区副本移到 dual-role 节点池中，然后删除旧的仅代理节点池。

在此过程中，我们从两个节点池 pool-a 开始，它只有 controller 角色和 pool-b，它们只有 broker 角色：

单个角色节点池

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

Kafka 集群有六个节点：

节点池中的 Kafka 节点

```
NAME              READY  STATUS   RESTARTS
my-cluster-pool-a-0  1/1    Running  0
my-cluster-pool-a-1  1/1    Running  0
my-cluster-pool-a-2  1/1    Running  0
my-cluster-pool-b-3  1/1    Running  0
my-cluster-pool-b-4  1/1    Running  0
my-cluster-pool-b-5  1/1    Running  0
```

pool-a 节点执行控制器的角色。pool-b 节点执行代理角色。

注意

在此过程中，保存分区副本的节点 ID 会改变。考虑引用节点 ID 的任何依赖项。

先决条件

- 必须部署 Cluster Operator。
- Cruise Control 使用 Kafka 部署。

流程

1. 编辑节点池 pool-a 并为其添加 broker 角色。

节点池配置示例

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

2. 检查状态并等待节点池中的 pod 重启并就绪(1/1)。

```
oc get pods -n <my_cluster_operator_namespace>
```

输出显示在两个节点池中运行的 pod

```
NAME              READY  STATUS   RESTARTS
my-cluster-pool-a-0  1/1    Running  0
my-cluster-pool-a-1  1/1    Running  0
my-cluster-pool-a-2  1/1    Running  0
my-cluster-pool-b-3  1/1    Running  0
my-cluster-pool-b-4  1/1    Running  0
my-cluster-pool-b-5  1/1    Running  0
```

节点 ID 在创建时附加到节点的名称中。

3. 使用 Cruise Control remove-brokers 模式，将分区副本从 broker-only 节点重新分配给 dual-role 节点。

使用 Cruise Control 重新分配分区副本

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
  # ...
spec:
  mode: remove-brokers
  brokers: [3, 4, 5]
```

根据集群中的主题和分区数量，重新分配可能需要一些时间。

4. 删除具有旧仅限代理节点的 pool-b 节点池。

```
oc delete kafkanodepool pool-b -n <my_cluster_operator_namespace>
```

9.3.9. 使用节点池管理存储

Apache Kafka 的存储管理通常非常简单，设置时需要很少更改，但在某些情况下您可能需要修改存储配置。节点池简化了这个过程，因为您可以设置指定新存储要求的独立节点池。

在此过程中，我们为一个包含三个节点的、名为 pool-a 的节点池创建和管理存储。我们演示了如何更改定义其使用的持久性存储类型的存储类 (volumes.class)。您可以使用相同的步骤来更改存储大小 (volume.size)。

注意

我们强烈建议您使用块存储。Apache Kafka 的流只测试用于块存储。

先决条件

- 必须部署 Cluster Operator。
- Cruise Control 使用 Kafka 部署。

- 对于使用持久性卷声明进行动态卷分配的存储，存储类在与您需要的存储解决方案对应的 OpenShift 集群中定义并可用。

流程

1. 使用自己的存储设置创建节点池。

   例如，节点池 pool-a 使用带有持久性卷的 JBOD 存储：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaNodePool
metadata:
  name: pool-a
  labels:
    strimzi.io/cluster: my-cluster
spec:
  replicas: 3
  storage:
    type: jbod
    volumes:
      - id: 0
        type: persistent-claim
        size: 500Gi
        class: gp2-ebs
  # ...
```

   pool-a 中的节点配置为使用 Amazon EBS (Elastic Block Store) GP2 卷。

2. 为 pool-a 应用节点池配置。

3. 检查部署的状态，并等待 pool-a 中的 pod 创建并就绪(1/1)。

```
oc get pods -n <my_cluster_operator_namespace>
```

输出显示节点池中的三个 Kafka 节点

```
NAME                 READY  STATUS   RESTARTS
my-cluster-pool-a-0  1/1    Running  0
my-cluster-pool-a-1  1/1    Running  0
my-cluster-pool-a-2  1/1    Running  0
```

4. 要迁移到新存储类，请使用所需存储配置创建新节点池：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaNodePool
metadata:
  name: pool-b
  labels:
    strimzi.io/cluster: my-cluster
spec:
  roles:
    - broker
  replicas: 3
  storage:
    type: jbod
    volumes:
      - id: 0
        type: persistent-claim
        size: 1Ti
        class: gp3-ebs
  # ...
```

pool-b 中的节点配置为使用 Amazon EBS (Elastic Block Store) GP3 卷。

5. 为 pool-b 应用节点池配置。

6. 检查部署的状态，并等待 pool-b 中的 pod 创建并就绪。

7. 将分区从 pool-a 重新分配给 pool-b。

迁移到新的存储配置时，请使用 Cruise Control remove-brokers 模式，将分区副本从要删除的代理移出。

使用 Cruise Control 重新分配分区副本

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
  # ...
```

```
spec:
  mode: remove-brokers
  brokers: [0, 1, 2]
```

我们是从 池-a 重新分配分区。根据集群中的主题和分区数量，重新分配可能需要一些时间。

8. 重新分配过程完成后，删除旧节点池：

```
oc delete kafkanodepool pool-a
```

9.3.10. 使用节点池管理存储关联性

如果存储资源（如本地持久性卷）受到特定 worker 节点或可用区的限制，配置存储关联性有助于将 pod 调度到正确的节点。

节点池允许您独立配置关联性。在此过程中，我们为两个可用区创建和管理存储关联性： zone-1 和 zone-2。

您可以为单独的可用区配置节点池，但使用相同的存储类。我们定义了一个 all-zones 持久性存储类，代表每个区域中可用的存储资源。

我们还使用 .spec.template.pod 属性来配置节点关联性，并在 zone-1 和 zone-2 worker 节点上调度 Kafka pod。

存储类和关联性在代表每个可用区中的节点池中指定：

- pool-zone-1
- pool-zone-2.

先决条件

- 必须部署 Cluster Operator。
- 如果您不熟悉关联性概念，请参阅 Kubernetes 节点和 pod 关联性文档。

流程

1. 定义用于每个可用区的存储类：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: all-zones
provisioner: kubernetes.io/my-storage
parameters:
  type: ssd
volumeBindingMode: WaitForFirstConsumer
```

2. 创建代表两个可用区的节点池，指定每个区的 all-zones 存储类和关联性：

zone-1 的节点池配置

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaNodePool
metadata:
  name: pool-zone-1
  labels:
    strimzi.io/cluster: my-cluster
spec:
  replicas: 3
  storage:
    type: jbod
    volumes:
      - id: 0
        type: persistent-claim
        size: 500Gi
        class: all-zones
  template:
    pod:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                - key: topology.kubernetes.io/zone
                  operator: In
```

```
            values:
            - zone-1
# ...
```

zone-2 的节点池配置

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaNodePool
metadata:
  name: pool-zone-2
  labels:
    strimzi.io/cluster: my-cluster
spec:
  replicas: 4
  storage:
    type: jbod
    volumes:
      - id: 0
        type: persistent-claim
        size: 500Gi
        class: all-zones
  template:
    pod:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                - key: topology.kubernetes.io/zone
                  operator: In
                  values:
                  - zone-2
# ...
```

3. 应用节点池配置。

4. 检查部署的状态，并等待节点池中的 pod 创建并就绪(1/1)。

```
oc get pods -n <my_cluster_operator_namespace>
```

输出显示 pool-zone-1 和 4 Kafka 节点中的 3 个 Kafka 节点(pool-zone-2)

```
NAME                          READY  STATUS   RESTARTS
my-cluster-pool-zone-1-kafka-0  1/1    Running  0
my-cluster-pool-zone-1-kafka-1  1/1    Running  0
my-cluster-pool-zone-1-kafka-2  1/1    Running  0
my-cluster-pool-zone-2-kafka-3  1/1    Running  0
my-cluster-pool-zone-2-kafka-4  1/1    Running  0
my-cluster-pool-zone-2-kafka-5  1/1    Running  0
my-cluster-pool-zone-2-kafka-6  1/1    Running  0
```

9.3.11. 将现有 Kafka 集群迁移到使用 Kafka 节点池

这个步骤描述了如何迁移现有 Kafka 集群以使用 Kafka 节点池。更新 Kafka 集群后，您可以使用节点池来管理每个池中的节点配置。

注意

目前，KafkaNodePool 资源中的副本和存储配置还必须存在于 Kafka 资源中。使用节点池时会忽略配置。

先决条件

- 必须部署 Cluster Operator。

流程

1. 创建新的 KafkaNodePool 资源。
   a. 将资源命名为 kafka。
   b. 将 strimzi.io/cluster 标签指向现有的 Kafka 资源。
   c. 设置副本数和存储配置以匹配您当前的 Kafka 集群。

d.

将角色设置为 **代理**。

迁移 Kafka 集群的节点池配置示例

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

警告

要保留集群数据及其节点和资源名称，节点池名称必须是 kafka，并且 strimzi.io/cluster 标签必须与 Kafka 资源名称匹配。否则，节点和资源会使用新名称创建，包括节点使用的持久性卷存储。因此，您之前的数据可能不可用。

2.

应用 KafkaNodePool 资源：

```
oc apply -f <node_pool_configuration_file>
```

通过应用此资源，您可以将 Kafka 切换到使用节点池。

没有更改或滚动更新和资源与之前的资源相同。

3. 使用 strimzi.io/node-pools: enabled 注解启用对 Kafka 资源中的节点池的支持。

使用 ZooKeeper 在集群中节点池的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  annotations:
    strimzi.io/node-pools: enabled
spec:
  kafka:
    # ...
  zookeeper:
    # ...
```

4. 应用 Kafka 资源：

```
oc apply -f <kafka_configuration_file>
```

没有更改或滚动更新。这些资源与之前的资源相同。

5. 从 Kafka 自定义资源中删除复制的属性。当使用 KafkaNodePool 资源时，您可以删除复制到 KafkaNodePool 资源的属性，如 .spec.kafka.replicas 和 .spec.kafka.storage 属性。

翻转迁移

要恢复到只使用 Kafka 自定义资源管理 Kafka 节点：

1. 如果您有多个节点池，将其合并到名为 kafka 的单个 KafkaNodePool 中，节点 ID 从 0 到 N（其中 N 是副本数）。

2. 确保 Kafka 资源中的 .spec.kafka 配置与 Kafka NodePool 配置匹配，包括存储、资源和副

本。

3. 使用 strimzi.io/node-pools: disabled 注解禁用 Kafka 资源中节点池的支持。

4. 删除名为 kafka 的 Kafka 节点池。

9.4. 配置实体 OPERATOR

使用 Kafka.spec 中的 entityOperator 属性来配置实体 Operator。Entity Operator 用于管理在 Kafka 集群中运行的与 Kafka 相关的实体。它由以下 Operator 组成：

- 管理 Kafka 主题的主题 Operator
- 用于管理 Kafka 用户的用户 Operator

通过配置 Kafka 资源，Cluster Operator 可以部署 Entity Operator，包括一个或多个 Operator。部署后，Operator 会自动配置为处理 Kafka 集群的主题和用户。

每个 operator 只能监控单个命名空间。如需更多信息，请参阅 第 1.2.1 节 “在 OpenShift 命名空间中监视 Apache Kafka 资源的流”。

entityOperator 属性支持多个子属性：

- 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 9.16 节 “自定义 OpenShift 资源”。

topicOperator 属性包含主题 Operator 的配置。当缺少这个选项时，在没有 Topic Operator 的情况下部署 Entity Operator。

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

有关配置实体 Operator 的属性的更多信息，请参阅 EntityOperatorSpec 模式参考。

启用这两个 Operator 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，实体 Operator 不会被部署。

9.4.1. 配置主题 Operator

使用 Kafka.spec.entityOperator 中的 topicOperator 属性来配置 Topic Operator。

注意

如果您使用 unidirectional 主题管理（默认启用），则不会使用以下属性，并会被忽略： Kafka.spec.entityOperator.topicOperator.zookeeperSessionTimeoutSeconds 和 Kafka.spec.entityOperator.topicMetadataMaxAttempts。有关单向主题管理的详情，请参考 第 10.1 节 “主题管理模式”。

支持以下属性：

watchedNamespace

主题 Operator 监视 KafkaTopic 资源的 OpenShift 命名空间。default 是部署 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 120。

zookeeperSessionTimeoutSeconds

ZooKeeper 会话超时（以秒为单位）。默认 18。

topicMetadataMaxAttempts

从 Kafka 获取主题元数据的尝试次数。每次尝试之间的时间都定义为指数避退。当主题创建可能会因为分区或副本数增加时，请考虑增加这个值。默认 6。

image

image 属性可用于配置使用的容器镜像。如需更多信息，请参阅有关 配置镜像 属性 提供的信息。

资源

resources 属性配置分配给 Topic Operator 的资源数量。您可以为 内存和 cpu 资源指定请求和限值。请求应该足以确保操作器的稳定性能。

logging

logging 属性配置 Topic Operator 的日志记录。如需更多信息，请参阅 主题 Operator 日志记录 上提供的信息。

主题 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
      resources:
        requests:
          cpu: "1"
          memory: 500Mi
        limits:
          cpu: "1"
          memory: 500Mi
    # ...
```

9.4.2. 配置用户 Operator

使用 Kafka.spec.entityOperator 中的 userOperator 属性来配置 User Operator。支持以下属性：

watchedNamespace

User Operator 监视 KafkaUser 资源的 OpenShift 命名空间。default 是部署 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 120。

image

image 属性可用于配置要使用的容器镜像。如需更多信息，请参阅有关 配置镜像 属性 提供的信息。

资源

resources 属性配置分配给 User Operator 的资源数量。您可以为 内存和 cpu 资源指定请求和限值。请求应该足以确保操作器的稳定性能。

logging

logging 属性配置 User Operator 的日志记录。如需更多信息，请参阅 User Operator 日志记录上提供的信息。

secretPrefix

secretPrefix 属性在 KafkaUser 资源创建的所有 Secret 的名称中添加前缀。例如，secretPrefix: kafka- 会将所有 Secret 名称的前缀为 kafka-。因此，名为 my-user 的 KafkaUser 会创建一个名为 kafka-my-user 的 Secret。

User Operator 配置示例

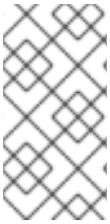
```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    userOperator:
      watchedNamespace: my-user-namespace
      reconciliationIntervalSeconds: 60
      resources:
        requests:
          cpu: "1"
          memory: 500Mi
        limits:
          cpu: "1"
          memory: 500Mi
    # ...
```

9.5. 配置 CLUSTER OPERATOR

使用环境变量配置 Cluster Operator。在 Deployment 配置文件中，指定 Cluster Operator 的容器镜像的环境变量。您可以使用以下环境变量来配置 Cluster Operator。如果您以待机模式运行 Cluster

Operator 副本，则需要额外的 环境变量来启用领导选举机制。

Kafka、Kafka Connect 和 Kafka MirrorMaker 支持多个版本。使用它们的 STRIMZI_<COMPONENT_NAME>_IMAGES 环境变量配置用于每个版本的默认容器镜像。配置提供了版本和镜像之间的映射。所需的语法是空格或用逗号分开的 < version> = <image > 对，它决定用于给定版本的镜像。例如： 3.7.0=registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0。如果在组件的配置中指定了 镜像 属性值，则这些默认镜像会被覆盖。有关组件镜像 配置的更多信息，请参阅 Apache Kafka 自定义资源 API 参考流。



注意

由 Apache Kafka 发行工件的 Streams 提供的 Deployment 配置文件是 install/cluster-operator/060-Deployment-strimzi-cluster-operator.yaml。

STRIMZI_NAMESPACE

以逗号分隔的 Operator 运行的命名空间列表。如果没有设置，则为空字符串，或设置为 *，Cluster Operator 在所有命名空间中运行。

Cluster Operator 部署可能会使用 Downward API 把它自动设置为 Cluster Operator 部署的命名空间。

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，默认的 300000 ms。内部操作的超时时间，以毫秒为单位。当在常规 OpenShift 操作需要更长的时间（例如，为容器镜像下载时间）的集群中使用 Streams for Apache Kafka 时增加这个

值。

STRIMZI_ZOOKEEPER_ADMIN_SESSION_TIMEOUT_MS

可选，默认为 10000 ms。Cluster Operator 的 ZooKeeper admin 客户端的会话超时，以毫秒为单位。如果 Cluster Operator 的 ZooKeeper 请求因为超时问题而定期失败，则增加这个值。通过 maxSessionTimeout 配置在 ZooKeeper 服务器端设置了最大允许的会话时间。默认情况下，最大会话超时值为 20 倍，默认的 tickTime （默认值为 2000）为 40000 ms。如果您需要更高的超时，请更改 maxSessionTimeout ZooKeeper 服务器配置值。

STRIMZI_OPERATIONS_THREAD_POOL_SIZE

可选，默认为 10。worker 线程池大小，用于各种异步和阻止由 Cluster Operator 运行的操作。

STRIMZI_OPERATOR_NAME

可选，默认为 pod 的主机名。在发出 OpenShift 事件时，Operator 名称标识 Apache Kafka 实例的 Streams。

STRIMZI_OPERATOR_NAMESPACE

运行 Cluster Operator 的命名空间的名称。不要手动配置此变量。使用 Downward API。

```
env:
  - name: STRIMZI_OPERATOR_NAMESPACE
    valueFrom:
      fieldRef:
        fieldPath: metadata.namespace
```

STRIMZI_OPERATOR_NAMESPACE_LABELS

可选。运行 Apache Kafka Cluster Operator 的 Streams 的命名空间标签。使用命名空间标签在网络策略 中配置命名空间选择器。网络策略允许 Apache Kafka Cluster Operator 的 Streams 访问使用这些标签的命名空间中的操作对象。如果没有设置，则网络策略中的命名空间选择器会被配置为允许从 OpenShift 集群中任何命名空间访问 Cluster Operator。

```
env:
  - name: STRIMZI_OPERATOR_NAMESPACE_LABELS
    value: label1=value1,label2=value2
```

STRIMZI_LABELS_EXCLUSION_PATTERN

可选，默认的正则表达式为 ^app.kubernetes.io/(?!part-of).*。用于过滤从主自定义资源传播到其子资源的正则表达式排除模式。标签排除过滤器不适用于 template 部分中的标签，如 spec.kafka.template.pod.metadata.labels。

```
env:
  - name: STRIMZI_LABELS_EXCLUSION_PATTERN
    value: "^key1.*"
```

STRIMZI_CUSTOM_<COMPONENT_NAME>_LABELS

可选。一个或多个自定义标签，应用到组件自定义资源创建的所有 pod。当创建自定义资源或下一个协调时，Cluster Operator 会标记 pod。

标签可应用到以下组件：

- KAFKA
- KAFKA_CONNECT
- KAFKA_CONNECT_BUILD
- ZOOKEEPER
- ENTITY_OPERATOR
- KAFKA_MIRROR_MAKER2
- KAFKA_MIRROR_MAKER
- CRUISE_CONTROL
- KAFKA_BRIDGE
- KAFKA_EXPORTER

STRIMZI_CUSTOM_RESOURCE_SELECTOR

可选。过滤 Cluster Operator 处理的自定义资源的标签选择器。Operator 仅在设置了指定标签的自定义资源上运行。Operator 不会看到没有这些标签的资源。标签选择器适用于 Kafka, KafkaConnect, 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，才会操作 KafkaRebalance 和 KafkaConnector 资源。

```
env:
 - name: STRIMZI_CUSTOM_RESOURCE_SELECTOR
   value: label1=value1,label2=value2
```

STRIMZI_KAFKA_IMAGES

必需。从 Kafka 版本到包含该版本的 Kafka 代理的对应镜像的映射。例如 3.6.0=registry.redhat.io/amq-streams/kafka-36-rhel9:2.7.0, 3.7.0=registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0。

STRIMZI_KAFKA_CONNECT_IMAGES

必需。从 Kafka 版本到该版本的 Kafka Connect 的对应镜像的映射。例如 3.6.0=registry.redhat.io/amq-streams/kafka-36-rhel9:2.7.0, 3.7.0=registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0。

STRIMZI_KAFKA_MIRROR_MAKER2_IMAGES

必需。从 Kafka 版本到该版本的 MirrorMaker 2 的对应镜像的映射。例如 3.6.0=registry.redhat.io/amq-streams/kafka-36-rhel9:2.7.0, 3.7.0=registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0。

（已弃用） STRIMZI_KAFKA_MIRROR_MAKER_IMAGES

必需。从 Kafka 版本到该版本的 MirrorMaker 的对应镜像的映射。例如 3.6.0=registry.redhat.io/amq-streams/kafka-36-rhel9:2.7.0, 3.7.0=registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0。

STRIMZI_DEFAULT_TOPIC_OPERATOR_IMAGE

可选。默认为 registry.redhat.io/amq-streams/strimzi-rhel9-operator:2.7.0。如果没有将镜像指定为 Kafka.spec.entityOperator.topicOperator.image 中的 Kafka.spec.entityOperator.image，则部署主题 Operator 时 用作默认 镜像名称。

STRIMZI_DEFAULT_USER_OPERATOR_IMAGE

可选。默认为 registry.redhat.io/amq-streams/strimzi-rhel9-operator:2.7.0。如果没有将镜像指定为 Kafka.spec.entityOperator.userOperator.image 中的 Kafka.spec.entityOperator.image，则部署 User Operator 时要使用的 镜像名称。

STRIMZI_DEFAULT_TLS_SIDECAR_ENTITY_OPERATOR_IMAGE

可选。默认为 registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0。如果没有将镜像指定为 Kafka.spec.entityOperator.tlsSidecar.image 中的 Kafka.spec.entityOperator.tlsSidecar.image

时，用作实体 Operator 的 sidecar 容器的默认值。sidecar 提供 TLS 支持。

STRIMZI_DEFAULT_KAFKA_EXPORTER_IMAGE

可选。默认为 registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0。如果没有将镜像指定为 Kafka 资源中的 Kafka.spec.kafkaExporter.image，则部署 Kafka Exporter 时要使用的镜像名称。

STRIMZI_DEFAULT_CRUISE_CONTROL_IMAGE

可选。默认为 registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0。如果没有将镜像指定为 Kafka.spec.cruiseControl.image 中的 Kafka.spec.cruiseControl.image，则部署 Cruise Control 时用作默认设置的镜像名称。

STRIMZI_DEFAULT_KAFKA_BRIDGE_IMAGE

可选。默认为 registry.redhat.io/amq-streams/bridge-rhel9:2.7.0。如果没有将镜像指定为 Kafka 资源中的 Kafka.spec.kafkaBridge.image，则部署 Kafka Bridge 时用作默认 镜像名称。

STRIMZI_DEFAULT_KAFKA_INIT_IMAGE

可选。默认为 registry.redhat.io/amq-streams/strimzi-rhel9-operator:2.7.0。如果没有在 Kafka 资源的 brokerRackInitImage 或 Kafka Connect 资源的 clientRackInitImage 中指定镜像，则用作 Kafka initializer 容器的默认镜像名称。init 容器在 Kafka 集群前启动，用于初始配置，如机架支持。

STRIMZI_IMAGE_PULL_POLICY

可选。应用到由 Cluster Operator 管理的所有 pod 中的容器的 ImagePullPolicy。有效值为 Always、ifNotP resent 和 Never。如果未指定，则使用 OpenShift 默认值。更改策略将导致对所有 Kafka、Kafka Connect 和 Kafka MirrorMaker 集群进行滚动更新。

STRIMZI_IMAGE_PULL_SECRETS

可选。以逗号分隔的 Secret 名称列表。此处引用的 secret 包含拉取容器镜像的容器 registry 的凭证。secret 在 Cluster Operator 创建的所有 pod 的 imagePullSecrets 属性中指定。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 OpenShift 版本信息。

OpenShift 版本覆盖配置示例

```
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 集群中分配的服务具有使用默认后缀 cluster.local 的 DNS 域名。

例如，对于 broker *kafka-0* ：

```
<cluster-name>-kafka-0.<cluster-name>-kafka-brokers.<namespace>.svc.cluster.local
```

DNS 域名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。

STRIMZI_CONNECT_BUILD_TIMEOUT_MS

可选，默认的 300000 ms。使用额外连接器构建新 Kafka Connect 镜像的超时时间（以毫秒为单位）。在使用 Apache Kafka 的 Streams 构建容器镜像时，请考虑增加这个值，以构建包含多个连接器或使用较慢的容器 registry 的容器镜像。

STRIMZI_NETWORK_POLICY_GENERATION

可选，默认 true。资源的网络策略。网络策略允许 Kafka 组件间的连接。

将此环境变量设置为 false 可禁用网络策略生成。例如，如果要使用自定义网络策略，您可以执行此操作。自定义网络策略可让更多地控制组件间的连接。

STRIMZI_DNS_CACHE_TTL

可选，默认为 30。在本地 DNS 解析器中缓存成功名称查找的秒数。任何负值表示永久缓存。零表示不缓存，这对于避免应用较长缓存策略的连接错误非常有用。

STRIMZI_POD_SET_RECONCILIATION_ONLY

可选，默认 false。当设置为 true 时，Cluster Operator 会仅协调 StrimziPodSet 资源，并忽略其他自定义资源(Kafka、KafkaConnect 等)的更改。此模式可用于确保在需要时重新创建 pod，但不会对集群进行其他更改。

STRIMZI_FEATURE_GATES

可选。启用或禁用由 功能门控制的功能。

STRIMZI_POD_SECURITY_PROVIDER_CLASS

可选。可插拔的 PodSecurityProvider 类配置，可用于为 Pod 和容器提供安全上下文配置。

9.5.1. 使用网络策略限制对 Cluster Operator 的访问

使用 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。

Cluster Operator 可以在与它管理的资源或单独的命名空间中运行。默认情况下，STRIMZI_OPERATOR_NAMESPACE 环境变量配置为使用 Downward API 来查找 Cluster Operator 运行的命名空间。如果 Cluster Operator 在与资源相同的命名空间中运行，则只有本地访问需要本地访问，并被 Apache Kafka Streams 允许。

如果 Cluster Operator 在一个独立的命名空间中运行，则 OpenShift 集群中的所有命名空间都可以访问 Cluster Operator，除非配置了网络策略。通过添加命名空间标签，对 Cluster Operator 的访问仅限于指定的命名空间。

为 Cluster Operator 部署配置的网络策略

```
#...
env:
  # ...
  - name: STRIMZI_OPERATOR_NAMESPACE_LABELS
    value: label1=value1,label2=value2
  #...
```

9.5.2. 设置自定义资源的定期协调

使用 STRIMZI_FULL_RECONCILIATION_INTERVAL_MS 变量设置 Cluster Operator 定期协调的时间间隔。将其值替换为所需间隔（以毫秒为单位）。

为 Cluster Operator 部署配置的协调周期

```
#...
env:
  # ...
  - name: STRIMZI_FULL_RECONCILIATION_INTERVAL_MS
    value: "120000"
  #...
```

Cluster Operator 对从 OpenShift 集群接收的适用集群资源的所有通知做出反应。如果 Operator 没有运行，或者因为某种原因没有收到通知，资源将与正在运行的 OpenShift 集群的状态不同步。为了正确处理故障转移，Cluster Operator 会执行定期协调过程，以便它可以将资源的状态与当前集群部署进行比较，以便在所有环境中具有一致的状态。

其他资源

- Downward API

9.5.3. 使用注解暂停自定义资源的协调

有时，暂停由 Streams for Apache Kafka operator 管理的自定义资源的协调，以便您可以执行修复或更新。如果协调暂停，Operator 会忽略对自定义资源所做的任何更改，直到暂停结束为止。

如果要暂停自定义资源的协调，请在其配置中将 strimzi.io/pause-reconciliation 注解设置为 true。这指示适当的 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，以便由 Cluster Operator 协调暂停。

您还可以创建启用 pause 注解的自定义资源。自定义资源已创建，但会被忽略。

先决条件

- 管理自定义资源的 Apache Kafka Operator 的 Streams 正在运行。

流程

1. 在 OpenShift 中注解自定义资源，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 ReconciliationPaused 的更改：

```
oc describe <kind_of_custom_resource> <name_of_custom_resource>
```

在 lastTransitionTime 中 type 条件会变为 ReconciliationPaused。

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
```

从暂停恢复

- 要恢复协调，您可以将注解设置为 false，或删除注解。

其他资源

- 查找自定义资源的状态

9.5.4. 使用领导选举机制运行多个 Cluster Operator 副本

默认 Cluster Operator 配置可让领导选举机制运行 Cluster Operator 的多个并行副本。一个副本被选为活跃领导值，并运行部署的资源。其他副本以待机模式运行。当领导机停止或失败时，其中一个备用副本被选为新的领导，并开始操作部署的资源。

默认情况下，Apache Kafka 的 Streams 使用始终为领导副本的单个 Cluster Operator 副本运行。当单个 Cluster Operator 副本停止或失败时，OpenShift 会启动新的副本。

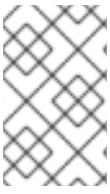
运行具有多个副本的 Cluster Operator 并不是必须的。但是，在出现重大故障导致的大规模中断时，在待机时具有副本非常有用。例如，假设多个 worker 节点或整个可用区失败。故障可能会导致 Cluster Operator pod 和许多 Kafka pod 同时停机。如果后续 pod 调度会导致缺少资源，这可能会在运行单个 Cluster Operator 时延迟操作。

9.5.4.1. 为 Cluster Operator 副本启用领导选举机制

在运行额外的 Cluster Operator 副本时，配置领导选举环境变量。支持以下环境变量：

STRIMZI_LEADER_ELECTION_ENABLED

可选，默认禁用(false)。启用或禁用领导选举机制，允许其他 Cluster Operator 副本在待机上运行。



注意

默认情况下，领导选举机制被禁用。它仅在安装时应用此环境变量时启用。

STRIMZI_LEADER_ELECTION_LEASE_NAME

启用领导选举机制时需要。用于领导选举的 OpenShift Lease 资源的名称。

STRIMZI_LEADER_ELECTION_LEASE_NAMESPACE

启用领导选举机制时需要。创建用于领导选举 OpenShift Lease 资源的命名空间。您可以使用 Downward API 将其配置为部署 Cluster Operator 的命名空间。

```
env:
  - name: STRIMZI_LEADER_ELECTION_LEASE_NAMESPACE
    valueFrom:
      fieldRef:
        fieldPath: metadata.namespace
```

STRIMZI_LEADER_ELECTION_IDENTITY

启用领导选举机制时需要。配置在领导选举过程中使用的给定 Cluster Operator 实例的身份。每个 operator 实例的身份必须是唯一的。您可以使用 Downward API 将其配置为部署 Cluster Operator 的 pod 的名称。

```
env:
  - name: STRIMZI_LEADER_ELECTION_IDENTITY
    valueFrom:
      fieldRef:
        fieldPath: metadata.name
```

STRIMZI_LEADER_ELECTION_LEASE_DURATION_MS

可选，默认为 15000 ms。指定获取租期的有效持续时间。

STRIMZI_LEADER_ELECTION_RENEW_DEADLINE_MS

可选，默认为 10000 ms。指定领导机应尝试保持领导状态的期间。

STRIMZI_LEADER_ELECTION_RETRY_PERIOD_MS

可选，默认为 2000 ms。指定领导对租期锁定的更新频率。

9.5.4.2. 配置 Cluster Operator 副本

要以待机模式运行额外的 Cluster Operator 副本，您需要增加副本数并启用领导选举机制。要配置领导选举机制，请使用领导选举环境变量。

要进行所需的更改，请配置位于 install/cluster-operator/ 中的以下 Cluster Operator 安装文件：

- 060-Deployment-strimzi-cluster-operator.yaml

- 022-ClusterRole-strimzi-cluster-operator-role.yaml
- 022-RoleBinding-strimzi-cluster-operator.yaml

领导选举机制具有自己的 ClusterRole 和 RoleBinding RBAC 资源，这些资源以 Cluster Operator 运行的命名空间为目标，而不是其监视的命名空间。

默认部署配置在与 Cluster Operator 相同的命名空间中创建一个名为 strimzi-cluster-operator 的 Lease 资源。Cluster Operator 使用租期来管理领导选举机制。RBAC 资源提供使用 Lease 资源的权限。如果您使用不同的 Lease 名称或命名空间，请相应地更新 ClusterRole 和 RoleBinding 文件。

先决条件

- 您需要具有创建和管理 CustomResourceDefinition 和 RBAC (ClusterRole、RoleBinding和 RoleBinding)资源的权限的帐户。

流程

编辑用于部署 Cluster Operator 的 Deployment 资源，该资源在 060-Deployment-strimzi-cluster-operator.yaml 文件中定义。

1. 将 replicas 属性从默认值(1)更改为与所需副本数匹配的值。

增加 Cluster Operator 副本数

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-cluster-operator
  labels:
    app: strimzi
spec:
  replicas: 3
```

2. 检查领导选举 env 属性是否已设置。

如果没有设置，请配置它们。

要启用领导选举机制，STRIMZI_LEADER_ELECTION_ENABLED 必须设置为 true （默认）。

在本例中，租期的名称被改为 my-strimzi-cluster-operator。

为 Cluster Operator 配置领导选举环境变量

```
# ...
spec
  containers:
    - name: strimzi-cluster-operator
      # ...
      env:
        - name: STRIMZI_LEADER_ELECTION_ENABLED
          value: "true"
        - name: STRIMZI_LEADER_ELECTION_LEASE_NAME
          value: "my-strimzi-cluster-operator"
        - name: STRIMZI_LEADER_ELECTION_LEASE_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
        - name: STRIMZI_LEADER_ELECTION_IDENTITY
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
```

有关可用环境变量的描述，请参阅 第 9.5.4.1 节 “为 Cluster Operator 副本启用领导选举机制”。

如果您为领导选举机制中使用的 Lease 资源指定了不同的名称或命名空间，请更新 RBAC 资源。

3. （可选）在 022-ClusterRole-strimzi-cluster-operator-role.yaml 文件中编辑 ClusterRole

资源。

使用 Lease 资源的名称更新 resourceNames。

更新对租期的 ClusterRole 引用

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: strimzi-cluster-operator-leader-election
  labels:
    app: strimzi
rules:
  - apiGroups:
      - coordination.k8s.io
    resourceNames:
      - my-strimzi-cluster-operator
# ...
```

4. （可选）在 022-RoleBinding-strimzi-cluster-operator.yaml 文件中编辑 RoleBinding 资源。

使用 Lease 资源的名称以及创建的命名空间更新 subjects.name 和 subjects.namespace。

更新对租期的 RoleBinding 引用

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: strimzi-cluster-operator-leader-election
  labels:
    app: strimzi
subjects:
  - kind: ServiceAccount
    name: my-strimzi-cluster-operator
    namespace: myproject
# ...
```

5. 部署 Cluster Operator：

```
oc create -f install/cluster-operator -n myproject
```

6. 检查部署的状态：

```
oc get deployments -n myproject
```

输出显示部署名称和就绪

```
NAME                      READY  UP-TO-DATE  AVAILABLE
strimzi-cluster-operator  3/3    3           3
```

READY 显示就绪/预期的副本数。当 AVAILABLE 输出显示正确的副本数时，部署可以成功。

9.5.5. 配置 Cluster Operator HTTP 代理设置

如果您在 HTTP 代理后运行 Kafka 集群，您仍然可以在集群内和移出数据。例如，您可以使用连接器运行 Kafka Connect，该连接器从代理外部推送和拉取镜像。或者，您可以使用代理与授权服务器连接。

配置 Cluster Operator 部署来指定代理环境变量。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 Apache Kafka 容器的所有流。

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，格式为 http://<username>:<password>@<ip-address>:<port_number>。

先决条件

- 您需要具有创建和管理 CustomResourceDefinition 和 RBAC

(ClusterRole、RoleBinding和 RoleBinding)资源的权限的帐户。

流程

1. 要在 Cluster Operator 中添加代理环境变量，请更新其部署配置 (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 直接作为代理服务器例外访问的服务器地址。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
```

其他资源

- 主机别名
- 为 Apache Kafka 管理员设计流

9.5.6. 使用 Cluster Operator 配置禁用 FIPS 模式

在启用了 FIPS 的 OpenShift 集群中运行时，Apache Kafka 的 Streams 会自动切换到 FIPS 模式。通过在 Cluster Operator 的部署配置 中禁用 FIPS_MODE 环境变量来禁用 FIPS 模式。禁用 FIPS 模式后，Apache Kafka 的 Streams 会在 OpenJDK 中为所有组件自动禁用 FIPS。禁用 FIPS 模式后，Apache Kafka 的 Streams 不兼容 FIPS。Apache Kafka operator 以及所有操作对象的流运行方式与启用了 FIPS 的 OpenShift 集群中运行的相同。

流程

1. 要在 Cluster Operator 中禁用 FIPS 模式，更新其部署配置install/cluster-operator/060-Deployment-strimzi-cluster-operator.yaml) 并增加 FIPS_MODE 环境变量。

   Cluster Operator 的 FIPS 配置示例

```
apiVersion: apps/v1
kind: Deployment
spec:
  # ...
  template:
    spec:
      serviceAccountName: strimzi-cluster-operator
      containers:
        # ...
        env:
```

```
      # ...
      - name: "FIPS_MODE"
        value: "disabled" 1
  # ...
```

1

禁用 FIPS 模式。

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 apply -f install/cluster-operator/060-Deployment-strimzi-cluster-operator.yaml
```

9.6. 配置 KAFKA CONNECT

更新 KafkaConnect 自定义资源的 spec 属性来配置 Kafka Connect 部署。

使用 Kafka Connect 为 Kafka 集群设置外部数据连接。使用 KafkaConnect 资源的属性来配置 Kafka Connect 部署。

要深入了解 Kafka Connect 集群配置选项，请参阅 Apache Kafka 自定义资源 API 参考。

KafkaConnector 配置

KafkaConnector 资源允许您以 OpenShift 原生的方式创建和管理 Kafka Connect 的连接器实例。

在 Kafka Connect 配置中，您可以通过添加 strimzi.io/use-connector-resources 注解来为 Kafka Connect 集群启用 KafkaConnectors。您还可以添加 构建配置，以便 Apache Kafka 的 Streams 会自动使用您数据连接所需的连接器插件构建容器镜像。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 KafkaConnector 自定义资源或 Kafka Connect REST API。KafkaConnector 资源必须部署到它们所链接的 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 添加连接器。

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 来配置连接器的特定元素。然后，您可以在 HTTP REST 命令中引用配置值，这样可保持配置独立且更安全。此方法特别适用于机密数据，如用户名、密码或证书。

处理大量信息

您可以调整配置以处理大量信息。如需更多信息，请参阅 处理大量信息。

KafkaConnect 自定义资源配置示例

```
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: connector-1
        artifacts:
          - type: tgz
            url: <url_to_download_connector_1_artifact>
            sha512sum: <SHA-512_checksum_of_connector_1_artifact>
      - name: connector-2
        artifacts:
          - type: jar
            url: <url_to_download_connector_2_artifact>
            sha512sum: <SHA-512_checksum_of_connector_2_artifact>
  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: OTEL_SERVICE_NAME
        value: my-otel-service
      - name: OTEL_EXPORTER_OTLP_ENDPOINT
        value: "http://otlp-host:4317"
tracing:
  type: opentelemetry 21
```

1

使用 KafkaConnect。

2

为 Kafka Connect 集群启用 KafkaConnectors。

3

运行任务的 worker 的副本节点数量。

4

Kafka Connect 集群的身份验证，指定为 mTLS、基于令牌的 OAuth、基于 SASL 的 SCRAM-SHA-256/SCRAM-SHA-512 或 PLAIN。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 集群的 bootstrap 服务器。

6

TLS 加密，使用密钥名称，其中 TLS 证书存储为集群的 X.509 格式。如果证书存储在同一 secret 中，则可以多次列出。

7

worker 的 Kafka 连接配置（而不是连接器）。标准 Apache Kafka 配置可能会提供，仅限于不直接由 Apache Kafka 的 Streams 管理的属性。

8

构建用于自动使用连接器插件构建容器镜像的配置属性。

9

（必需）推送新镜像的容器 registry 的配置。

10

（必需）连接器插件及其工件列表，以添加到新容器镜像中。每个插件必须配置至少一个 工件。

11

使用环境变量的连接器的外部配置，如此处或卷所示。您还可以使用配置供应商插件从外部来源加载配置值。

12

为保留支持的资源（当前 cpu 和 memory ）的请求，以及指定可消耗的最大资源的限制。

13

指定 Kafka Connect 日志记录器和日志级别直接(内联)或通过 ConfigMap 间接(外部)。自定义 Log4j 配置必须放在 ConfigMap 中的 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

检查检查以了解何时重启容器（存活度）以及何时容器可以接受流量（就绪度）。

15

Prometheus 指标，通过引用包含在此示例中 Prometheus JMX 导出器配置的 ConfigMap 启用。您可以使用对 metricsConfig.valueFrom.configMapKeyRef.key 下包含空文件的 ConfigMap 的引用来启用指标。

16

JVM 配置选项，用于优化运行 Kafka Connect 的虚拟机(VM)的性能。

17

ProductShortName OPTION: 容器镜像配置，仅在特殊情况下推荐使用。

18

SPECIALIZED OPTION：部署的机架感知配置。这是用于在同一位置（而非跨地区）部署的专用选项。如果您希望连接器从最接近的副本而不是领导副本使用，则使用此选项。在某些情况下，使用来自最接近的副本的消耗可以提高网络利用率或降低成本。topologyKey 必须与包含机架 ID 的节点标签匹配。此配置中使用的示例使用标准 topology.kubernetes.io/zone 标签指定区。要从最接近的副本使用，请在 Kafka 代理配置中启用 RackAwareReplicaSelector。

19

模板自定义。此处的 pod 使用反关联性调度，因此 pod 不会调度到具有相同主机名的节点。

20

为分布式追踪设置环境变量。

21

使用 OpenTelemetry 启用分布式追踪。

9.6.1. 为多个实例配置 Kafka 连接

默认情况下，Apache Kafka 的 Streams 配置 Kafka Connect 使用的内部主题的组 ID 和名称。在运行多个 Kafka Connect 实例时，您必须使用以下配置属性更改 这些默认设置 ：

```
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. Kafka 中的 Kafka Connect 集群组 ID。

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

对于具有相同 group.id 的所有实例，三个主题的值必须相同。

除非修改了这些默认设置，否则每个连接到同一 Kafka 集群的实例都会使用相同的值部署。在实践中，这意味着所有实例组成一个集群并使用相同的内部主题。

尝试使用同一内部主题的多个实例将导致意外错误，因此您必须更改每个实例的这些属性值。

9.6.2. 配置 Kafka Connect 用户授权

在 Kafka 中使用授权时，Kafka Connect 用户需要对集群组和 Kafka Connect 的内部主题进行读/写访问。此流程概述了如何使用 简单 授权和 ACL 授予访问权限。

Kafka Connect 集群组 ID 和内部主题的属性由 Apache Kafka 的 Streams 配置。另外，您可以在 KafkaConnect 资源的 spec 中显式定义它们。这在 为多个实例配置 Kafka Connect 时很有用，因为运行多个 Kafka Connect 实例时组 ID 和主题的值必须有所不同。

简单授权使用由 Kafka AclAuthorizer 和 StandardAuthorizer 插件管理的 ACL 规则来确保适当的访问级别。有关将 KafkaUser 资源配置为使用简单授权的更多信息，请参阅 AclRule 模式参考。

先决条件

- 一个 OpenShift 集群
- 正在运行的 Cluster Operator

流程

1. 编辑 KafkaUser 资源中的 authorization 属性，为用户提供访问权限。

使用 字面 名称值为 Kafka Connect 主题和集群组配置访问权限。下表显示了为主题和集群组 ID 配置的默认名称。

表 9.2. 访问权限配置的名称

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

在本例中，默认名称用于指定访问权限。如果您要为 Kafka Connect 实例使用不同的名称，请在 ACL 配置中使用这些名称。

简单授权配置示例

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

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

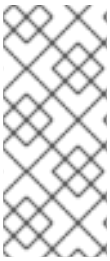
2. 创建或更新资源。

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

9.6.3. 手动停止或暂停 Kafka Connect 连接器

如果您使用 KafkaConnector 资源配置连接器，请使用 state 配置来停止或暂停连接器。与连接器和任务保持实例化的暂停状态不同，停止连接器只保留配置，且没有活跃进程。从运行停止连接器可能更适

合长时间运行，而不是只暂停。虽然暂停的连接器速度更快恢复，但已停止的连接器具有释放内存和资源的优点。



注意

state 配置替换 KafkaConnectorSpec 模式中的（已弃用） pause 配置，它允许在连接器上暂停。如果您之前使用 pause 配置来暂停连接器，我们建议您只使用 state 配置过渡到 以避免冲突。

先决条件

- Cluster Operator 正在运行。

流程

1. 查找控制您要暂停或停止连接器的 KafkaConnector 自定义资源的名称：

```
oc get KafkaConnector
```

2. 编辑 KafkaConnector 资源，以停止或暂停连接器。

停止 Kafka Connect 连接器的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-source-connector
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
  class: org.apache.kafka.connect.file.FileStreamSourceConnector
  tasksMax: 2
  config:
    file: "/opt/kafka/LICENSE"
    topic: my-topic
  state: stopped
  # ...
```

将 state 配置更改为 stopped 或 paused。当此属性没有设置时，连接器的默认状态为 running。

3. 对 KafkaConnector 配置应用更改。

您可以通过将 state 改为 running，或删除配置来恢复连接器。

注意

另外，您可以 公开 Kafka Connect API，并使用 stop 和 pause 端点停止连接器运行。例如，PUT /connectors/<connector_name>/stop。然后，您可以使用 resume 端点重启它。

9.6.4. 手动重启 Kafka 连接连接器

如果您使用 KafkaConnector 资源来管理连接器，请使用 strimzi.io/restart 注解来手动触发连接器的重启。

先决条件

- Cluster Operator 正在运行。

流程

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

```
oc get KafkaConnector
```

2. 通过在 OpenShift 中注解 KafkaConnector 资源来重启连接器。

```
oc annotate KafkaConnector <kafka_connector_name> strimzi.io/restart="true"
```

重启 注解设置为 true。

3. 等待下一个协调发生（默认为两分钟）。

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

9.6.5. 手动重启 Kafka Connect 连接器任务

如果您使用 KafkaConnector 资源来管理连接器，请使用 strimzi.io/restart-task 注解来手动触发连接器任务的重启。

先决条件

- Cluster Operator 正在运行。

流程

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

```
oc get KafkaConnector
```

2. 查找从 KafkaConnector 自定义资源重启的任务 ID：

```
oc describe KafkaConnector <kafka_connector_name>
```

任务 ID 是非负整数，从 0 开始。

3. 通过注解 OpenShift 中的 KafkaConnector 资源，使用 ID 重启连接器任务：

```
oc annotate KafkaConnector <kafka_connector_name> strimzi.io/restart-task="0"
```

在本例中，任务 0 被重启。

4. 等待下一个协调发生（默认为两分钟）。

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

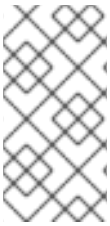
9.7. 配置 KAFKA MIRRORMAKER 2

更新 KafkaMirrorMaker2 自定义资源的 spec 属性，以配置 MirrorMaker 2 部署。MirrorMaker 2 使用源集群配置进行数据消耗和目标集群配置进行数据输出。

MirrorMaker 2 基于 Kafka Connect 框架，*连接器* 管理集群之间的数据传输。

您可以配置 MirrorMaker 2 以定义 Kafka Connect 部署，包括源和目标集群的连接详情，然后运行 MirrorMaker 2 连接器的集合来进行连接。

MirrorMaker 2 支持源和目标集群之间的主题配置同步。您可以在 MirrorMaker 2 配置中指定源主题。MirrorMaker 2 监控源主题。MirrorMaker 2 会检测并将更改传播到源主题到远程主题。更改可能包括自动创建缺少的主题和分区。



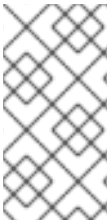
注意

在大多数情况下，您写入本地主题并从远程主题读取。虽然对远程主题不阻止写操作，但应该避免使用它们。

配置必须指定：

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

要深入了解 Kafka MirrorMaker 2 集群配置选项，请参阅 Apache Kafka 自定义资源 API 参考。



注意

MirrorMaker 2 资源配置与之前的 MirrorMaker 版本不同，它现已弃用。当前不支持旧支持，因此任何资源都必须手动转换为新格式。

默认配置

MirrorMaker 2 为复制因素等属性提供默认配置值。最小配置保持不变，默认值如下：

MirrorMaker 2 的最小配置

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

您可以使用 mTLS 或 SASL 身份验证为源和目标集群配置访问控制。此流程演示了如何为源和目标集群使用 TLS 加密和 mTLS 身份验证的配置。

您可以在 KafkaMirrorMaker2 资源中指定您要从源集群复制的主题和消费者组。您可以使用 topicsPattern 和 groupsPattern 属性进行此操作。您可以提供名称列表或使用正则表达式。默认情况下，如果您未设置 topicsPattern 和 groupsPattern 属性，则会复制所有主题和消费者组。您可以使用 ".*" 正则表达式来复制所有主题和消费者组。但是，尝试只指定您需要指定主题和消费者组，以避免在集群中造成不必要的额外负载。

处理大量信息

您可以调整配置以处理大量信息。如需更多信息，请参阅 处理大量信息。

KafkaMirrorMaker2 自定义资源配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  version: 3.7.0 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
    tls: 13
      trustedCertificates:
      - certificate: ca.crt
        secretName: my-cluster-target-cluster-ca-cert
  mirrors: 14
  - sourceCluster: "my-cluster-source" 15
    targetCluster: "my-cluster-target" 16
    sourceConnector: 17
      tasksMax: 10 18
      autoRestart: 19
        enabled: true
      config
```

```
      replication.factor: 1 20
      offset-syncs.topic.replication.factor: 1 21
      sync.topic.acls.enabled: "false" 22
      refresh.topics.interval.seconds: 60 23
      replication.policy.class: "org.apache.kafka.connect.mirror.IdentityReplicationPolicy" 24
  heartbeatConnector: 25
    autoRestart:
      enabled: true
    config:
      heartbeats.topic.replication.factor: 1 26
      replication.policy.class: "org.apache.kafka.connect.mirror.IdentityReplicationPolicy"
  checkpointConnector: 27
    autoRestart:
      enabled: true
    config:
      checkpoints.topic.replication.factor: 1 28
      refresh.groups.interval.seconds: 600 29
      sync.group.offsets.enabled: true 30
      sync.group.offsets.interval.seconds: 60 31
      emit.checkpoints.interval.seconds: 60 32
      replication.policy.class: "org.apache.kafka.connect.mirror.IdentityReplicationPolicy"
  topicsPattern: "topic1|topic2|topic3" 33
  groupsPattern: "group1|group2|group3" 34
resources: 35
  requests:
    cpu: "1"
    memory: 2Gi
  limits:
    cpu: "2"
    memory: 2Gi
logging: 36
  type: inline
  loggers:
    connect.root.logger.level: INFO
readinessProbe: 37
  initialDelaySeconds: 15
  timeoutSeconds: 5
livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
jvmOptions: 38
  "-Xmx": "1g"
  "-Xms": "1g"
image: my-org/my-image:latest 39
rack:
  topologyKey: topology.kubernetes.io/zone 40
template: 41
  pod:
    affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
```

```
            - key: application
              operator: In
              values:
                - postgresql
                - mongodb
          topologyKey: "kubernetes.io/hostname"
  connectContainer: 42
    env:
      - name: OTEL_SERVICE_NAME
        value: my-otel-service
      - name: OTEL_EXPORTER_OTLP_ENDPOINT
        value: "http://otlp-host:4317"
tracing:
  type: opentelemetry 43
externalConfiguration: 44
  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 MirrorMaker 2 版本，它们始终相同。

2

运行任务的 worker 的副本节点数量。

3

Kafka Connect 的 Kafka 集群别名，它必须 指定目标 Kafka 集群。Kafka Connect 使用 Kafka 集群用于其内部主题。

4

正在同步的 Kafka 集群的规格。

5

源 Kafka 集群的集群别名。

6

源集群的身份验证，指定为 mTLS、基于令牌的 OAuth、基于 SASL 的 SCRAM-SHA-256/SCRAM-SHA-512 或 PLAIN。

7

用于连接到源 Kafka 集群的 Bootstrap 服务器。

8

TLS 加密，使用密钥名称，其中 TLS 证书存储为源 Kafka 集群的 X.509 格式。如果证书存储在同一 secret 中，则可以多次列出。

9

目标 Kafka 集群的集群别名。

10

目标 Kafka 集群的身份验证配置方式与源 Kafka 集群相同。

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，仅限于不直接由 Apache Kafka 的 Streams 管理的属性。

13

目标 Kafka 集群的 TLS 加密配置方式与源 Kafka 集群相同。

14

MirrorMaker 2 连接器。

15

MirrorMaker 2 连接器使用的源集群的集群别名。

16

MirrorMaker 2 连接器使用的目标集群的集群别名。

17

MirrorSourceConnector 的配置，用于创建远程主题。配置会覆盖 默认配置选项。

18

连接器可创建的最大任务数量。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 在集群成员间分发任务。如果任务数量超过 worker，则 worker 会被分配多个任务。对于 sink 连接器，旨在为每个主题分区消耗一个任务。对于源连接器，可以并行运行的任务数量也可能依赖于外部系统。如果无法实现并行性，则连接器会创建少于最大任务数。

19

启用自动重启失败的连接器和任务。默认情况下，重启数量是无限的，但您可以使用 maxRestarts 属性设置自动重启次数的最大值。

20

在目标集群中创建的镜像主题的复制因素。

21

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。

22

启用 ACL 规则同步后，会将 ACL 应用到同步主题。默认值是 true。此功能与 User Operator 不兼容。如果使用 User Operator，请将此属性设置为 false。

23

可选设置，用于更改新主题的检查频率。默认值为每 10 分钟进行一次检查。

24

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。此可选设置可用于主动/被动备份和数据迁移。必须为所有连接器指定属性。对于双向 (active/active)复制，请使用 DefaultReplicationPolicy 类自动重命名远程主题，并为所有连接器指定 replication.policy.separator 属性来添加自定义分隔符。

25

执行连接检查的 MirrorHeartbeatConnector 的配置。配置会覆盖 默认配置选项。

26

在目标集群中创建的心跳主题的复制因素。

27

用于跟踪偏移的 MirrorCheckpointConnector 的配置。配置会覆盖 默认配置选项。

28

在目标集群中创建的检查点主题的复制因素。

29

可选设置，用于更改新消费者组的检查频率。默认值为每 10 分钟进行一次检查。

30

用于同步消费者组偏移的可选设置，这对于在主动/被动配置中恢复非常有用。默认不启用同步。

31

如果启用了使用者组偏移的同步，您可以调整同步的频率。

32

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

33

定义为以逗号分隔的列表或正则表达式模式的源集群的主题复制。源连接器复制指定的主题。checkpoint 连接器跟踪指定主题的偏移。此处我们按名称请求三个主题。

34

从以逗号分隔列表或正则表达式模式定义的源集群的消费者组复制。checkpoint 连接器复制指定的消费者组。此处我们按名称请求三个消费者组。

35

为保留支持的资源（当前 cpu 和 memory ）的请求，以及指定可消耗的最大资源的限制。

36

指定 Kafka Connect 日志记录器和日志级别直接(内联)或通过 ConfigMap 间接(外部)。自定义 Log4j 配置必须放在 ConfigMap 中的 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。

37

检查检查以了解何时重启容器（存活度）以及何时容器可以接受流量（就绪度）。

38

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

39

ProductShortName OPTION: 容器镜像配置，仅在特殊情况下推荐使用。

40

SPECIALIZED OPTION：部署的机架感知配置。这是用于在同一位置（而非跨地区）部署的专用选项。如果您希望连接器从最接近的副本而不是领导副本使用，则使用此选项。在某些情况下，使用来自最接近的副本的消耗可以提高网络利用率或降低成本。topologyKey 必须与包含机架 ID 的节点标签匹配。此配置中使用的示例使用标准 topology.kubernetes.io/zone 标签指定区。要从最接近的副本使用，请在 Kafka 代理配置中启用 RackAwareReplicaSelector。

41

模板自定义。此处的 pod 使用反关联性调度，因此 pod 不会调度到具有相同主机名的节点。

42

为分布式追踪设置环境变量。

43

使用 OpenTelemetry 启用分布式追踪。

44

作为环境变量挂载到 Kafka MirrorMaker 的 OpenShift Secret 的外部配置。您还可以使用配置供应商插件从外部来源加载配置值。

9.7.1. 配置主动/主动或主动/被动模式

您可以在*主动/被动*或*主动/主动*集群配置中使用 MirrorMaker 2。

主动/主动集群配置

主动/主动配置有两个主动集群双向复制数据。应用程序可以使用任一集群。每个集群都可以提供相同的数据。这样，您可以在不同的地理位置提供相同的数据。因为消费者组在两个集群中都活跃，复制主题的使用者偏移不会重新同步到源集群。

主动/被动集群配置

主动/被动配置具有主动集群将数据复制到被动集群。被动集群保持在待机状态。在出现系统失败时，您可以使用被动集群进行数据恢复。

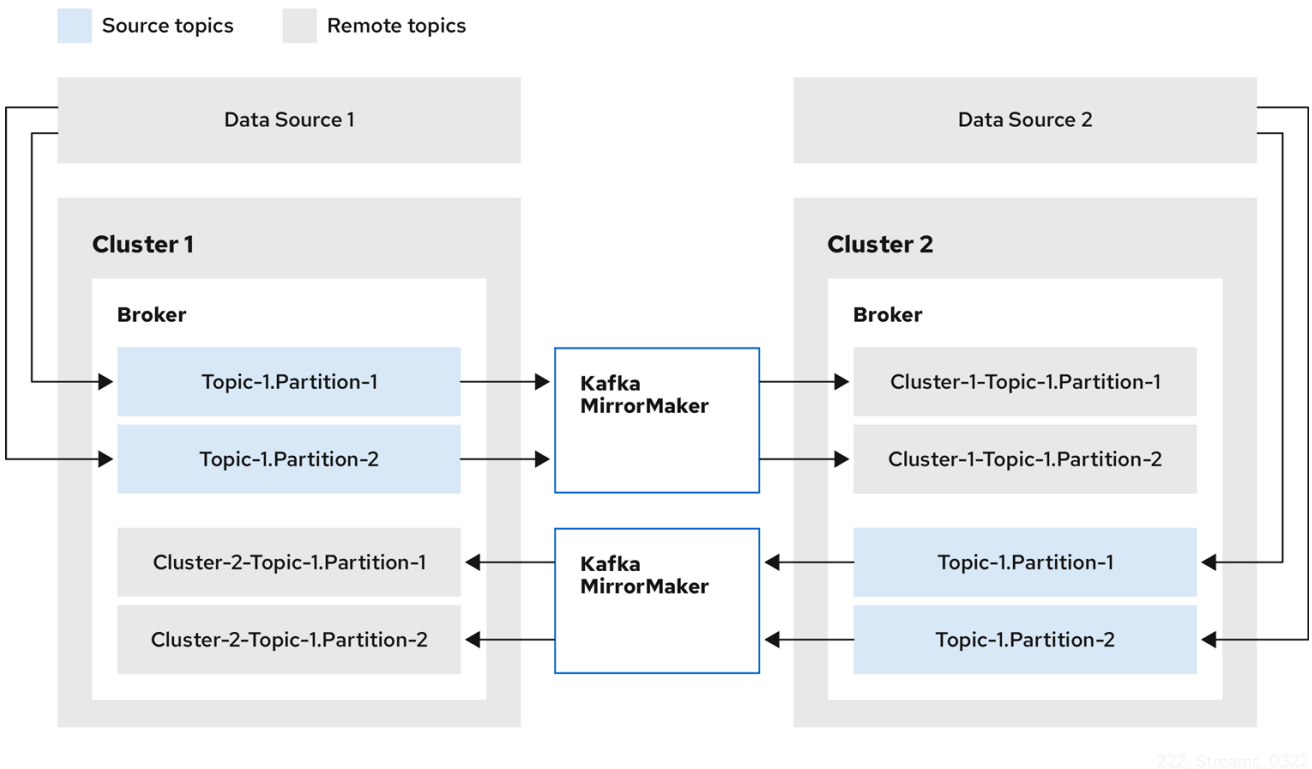
预期的结构是，生成者和消费者仅连接到活跃集群。每个目标目的地都需要一个 MirrorMaker 2 集群。

9.7.1.1. 双向复制（主动/主动）

MirrorMaker 2 架构支持 *主动/主动集群配置中* 的双向复制。

每个集群使用 *source* 和 *remote* 主题的概念复制其他集群的数据。由于同一主题存储在每个集群中，因此远程主题由 MirrorMaker 2 自动重命名，以代表源集群。原始集群的名称前面是主题名称的前面。

图 9.1. 主题重命名



通过标记原始集群，主题不会复制到该集群。

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集群中的源和目标主题，而无需单独的聚合集群。

9.7.1.2. 单向复制（主动/被动）

MirrorMaker 2 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* 配置来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。

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

9.7.2. 为多个实例配置 MirrorMaker 2

默认情况下，Apache Kafka 的 Streams 配置 MirrorMaker 2 运行的 Kafka Connect 框架使用的内部主题的组 ID 和名称。当运行多个 MirrorMaker 2 实例时，它们共享相同的 connectCluster 值，您必须使用以下配置属性更改 这些默认设置 ：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  connectCluster: "my-cluster-target"
  clusters:
  - alias: "my-cluster-target"
    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

Kafka 中的 Kafka Connect 集群组 ID。

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

对于具有相同 group.id 的所有实例，三个主题的值必须相同。

connectCluster 设置指定 Kafka Connect 用于其内部主题的目标 Kafka 集群的别名。因此，对 connectCluster、组 ID 和内部主题命名配置进行修改特定于目标 Kafka 集群。如果两个 MirrorMaker 2 实例使用相同的源 Kafka 集群或主动-主动模式，则不需要进行更改，其中每个 MirrorMaker 2 实例都有不同的 connectCluster 设置和目标集群。

但是，如果多个 MirrorMaker 2 实例共享相同的 connectCluster，则每个连接到同一目标 Kafka 集群的实例都会使用相同的值部署。在实践中，这意味着所有实例组成一个集群并使用相同的内部主题。

尝试使用同一内部主题的多个实例将导致意外错误，因此您必须更改每个实例的这些属性值。

9.7.3. 配置 MirrorMaker 2 连接器

将 MirrorMaker 2 连接器配置用于编配 Kafka 集群之间的数据同步的内部连接器。

MirrorMaker 2 由以下连接器组成：

MirrorSourceConnector

源连接器将主题从源集群复制到目标集群。它还复制 ACL，且是 MirrorCheckpointConnector 才能运行所必需的。

MirrorCheckpointConnector

checkpoint 连接器会定期跟踪偏移。如果启用，它还在源和目标集群之间同步消费者组偏移。

MirrorHeartbeatConnector

heartbeat 连接器会定期检查源和目标集群之间的连接。

下表描述了连接器属性以及您配置为使用它们的连接器。

表 9.3. MirrorMaker 2 连接器配置属性

属性	sourceConnector	checkpointConnector	heartbeatConnector
admin.timeout.ms 管理任务的超时，如检测新主题。默认值为 **60000** (1 分钟)。	✓	✓	✓
replication.policy.class 定义远程主题命名约定的策略。默认为 **org.apache.kafka.connect.mirror.DefaultReplicationPolicy**。	✓	✓	✓
replication.policy.separator 在目标集群中用于主题命名的分隔符。默认情况下，分隔符设置为点(.)。分隔符配置仅适用于 **DefaultReplicationPolicy** 复制策略类，用于定义远程主题名称。**IdentityReplicationPolicy** 类不使用属性，因为主题会保留其原始名称。	✓	✓	✓
consumer.poll.timeout.ms 轮询源集群时超时。默认值为 **1000** (1 秒)。	✓	✓	
offset-syncs.topic.location **offset-syncs** 主题的位置，可以是 **源**（默认）或 **目标集群**。	✓	✓	
topic.filter.class 选择要复制的主题的主题过滤器。默认为 **org.apache.kafka.connect.mirror.DefaultTopicFilter**。	✓	✓	

属性	sourceConnector	checkpointConnector	heartbeatConnector
config.property.filter.class 用于选择要复制的主题配置属性的主题过滤器。默认为 **org.apache.kafka.connect.mirror.DefaultConfigPropertyFilter**。	✓		
config.properties.exclude 不应复制的主题配置属性。支持以逗号分隔的属性名称和正则表达式。	✓		
offset.lag.max 在同步远程分区前，最大允许（不同步）偏移滞后。默认值为 **100**。	✓		
offset-syncs.topic.replication.factor 内部 **offset-syncs** 主题的复制因素。默认值为 **3**。	✓		
refresh.topics.enabled 启用检查新主题和分区。默认为 **true**。	✓		
refresh.topics.interval.seconds 主题刷新的频率。默认为 **600** (10 分钟)。默认情况下，检查源集群中的新主题每 10 分钟进行一次。您可以通过在源连接器配置中添加 **refresh.topics.interval.seconds** 来更改频率。	✓		
replication.factor 新主题的复制因素。默认值为 **2**。	✓		
sync.topic.acls.enabled 启用从源集群同步 ACL。默认为 **true**。如需更多信息，请参阅 第 9.7.6 节 “为远程主题同步 ACL 规则”。	✓		

属性	sourceConnector	checkpointConnector	heartbeatConnector
sync.topic.acls.interval.seconds ACL 同步的频率。默认为 **600** (10 分钟)。	✓		
sync.topic.configs.enabled 启用从源集群同步主题配置。默认为 **true**。	✓		
sync.topic.configs.interval.seconds 主题配置同步的频率。默认 **600** (10 分钟)。	✓		
checkpoints.topic.replication.factor 内部 **检查点** 主题的复制因素。默认值为 **3**。		✓	
emit.checkpoints.enabled 启用将消费者偏移同步到目标集群。默认为 **true**。		✓	
emit.checkpoints.interval.seconds 消费者偏移同步的频率。默认值为 **60** (1 分钟)。		✓	
group.filter.class 组过滤器，以选择要复制的消费者组。默认为 **org.apache.kafka.connect.mirror.DefaultGroupFilter**。		✓	
refresh.groups.enabled 启用检查新的消费者组。默认为 **true**。		✓	
refresh.groups.interval.seconds 消费者组刷新的频率。默认为 **600** (10 分钟)。		✓	

属性	sourceConnector	checkpointConnector	heartbeatConnector
sync.group.offsets.enabled 启用将消费者组偏移同步到目标集群 **__consumer_offsets** 主题。默认为 **false**。		✓	
sync.group.offsets.interval.seconds 消费者组偏移同步的频率。默认值为 **60** (1 分钟)。		✓	
emit.heartbeats.enabled 在目标集群中启用连接检查。默认为 **true**。			✓
emit.heartbeats.interval.seconds 连接检查的频率。默认为 **1 (** 1 秒)。			✓
heartbeats.topic.replication.factor 内部 **心跳** 主题的复制因素。默认值为 **3**。			✓

9.7.3.1. 更改消费者组偏移主题的位置

MirrorMaker 2 使用内部主题跟踪消费者组的偏移。

offset-syncs 主题

offset-syncs 主题映射复制主题元数据的源和目标偏移。

checkpoints 主题

checkpoints 主题映射源和目标集群中每个消费者组中复制的主题分区的最后提交偏移量。

因为它们被 MirrorMaker 2 内部使用，所以您不会直接与这些主题交互。

MirrorCheckpointConnector 为偏移跟踪发出 *检查点*。checkpoints 主题的偏移通过配置以预先确定的间隔进行跟踪。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。

offset-syncs 主题的位置是 源集群。您可以使用 offset-syncs.topic.location 连接器配置将其更改为 目标集群。您需要对包含该主题的集群进行读/写访问。使用目标集群作为 offset-syncs 主题的位置，您也可以使用 MirrorMaker 2，即使您只有对源集群的读访问权限。

9.7.3.2. 同步消费者组偏移

__consumer_offsets 主题存储各个消费者组的提交偏移信息。偏移同步会定期将源集群的消费者组的使用者偏移转移到目标集群的使用者偏移量中。

偏移同步在*主动/被动*配置中特别有用。如果主动集群停机，消费者应用程序可以切换到被动 (standby)集群，并从最后一个传输的偏移位置获取。

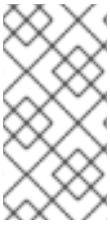
要使用主题偏移同步，请通过将 sync.group.offsets.enabled 添加到检查点连接器配置来启用同步，并将属性设置为 true。默认情况下禁用同步。

在源连接器中使用 IdentityReplicationPolicy 时，还必须在检查点连接器配置中进行配置。这样可确保为正确的主题应用镜像的消费者偏移。

消费者偏移仅针对目标集群中未激活的消费者组同步。如果消费者组位于目标集群中，则无法执行同步，并返回 UNKNOWN_MEMBER_ID 错误。

如果启用，则会定期从源集群同步偏移。您可以通过在检查点连接器配置中添加 sync.group.offsets.interval.seconds 和 emit.checkpoints.interval.seconds 来更改频率。属性指定同步消费者组偏移的频率，以及为偏移跟踪发送检查点的频率。这两个属性的默认值为 60 秒。您还可以使用 refresh.groups.interval.seconds 属性更改检查新消费者组的频率，该属性默认为每 10 分钟执行。

由于同步基于时间，因此消费者到被动集群的任何切换都可能会导致一些消息重复。



注意

如果您有使用 Java 编写的应用程序，您可以使用 RemoteClusterUtils.java 工具通过应用同步偏移。实用程序从 checkpoints 主题获取消费者组的远程偏移。

9.7.3.3. 决定使用 heartbeat 连接器的时间

heartbeat 连接器发出心跳来检查源和目标 Kafka 集群之间的连接。内部 心跳 主题从源集群复制，这

意味着 heartbeat 连接器必须连接到源集群。heartbeat 主题位于目标集群上，它允许它执行以下操作：

- 识别它要从中镜像数据的所有源集群
- 验证镜像进程的存活度和延迟

这有助于确保进程不会因为任何原因而卡住或已停止。虽然 heartbeat 连接器是监控 Kafka 集群之间的镜像进程的有价值的工具，但并非总是需要使用它。例如，如果您的部署具有低网络延迟或少量主题，您可能需要使用日志消息或其他监控工具来监控镜像过程。如果您决定不使用 heartbeat 连接器，只需从 MirrorMaker 2 配置中省略它。

9.7.3.4. 对齐 MirrorMaker 2 连接器的配置

为确保 MirrorMaker 2 连接器正常工作，请确保在连接器之间保持一致某些配置设置。具体来说，请确保以下属性在所有适用的连接器中具有相同的值：

- replication.policy.class
- replication.policy.separator
- offset-syncs.topic.location
- topic.filter.class

例如，source、检查点和 heartbeat 连接器的 replication.policy.class 的值必须相同。不匹配或缺失的设置会导致数据复制或偏移同步出现问题，因此必须使用同一设置保持所有相关连接器配置。

9.7.4. 配置 MirrorMaker 2 连接器制作者和消费者

MirrorMaker 2 连接器使用内部生产者和消费者。如果需要，您可以配置这些制作者和消费者来覆盖默认设置。

例如，您可以增加源制作者的 batch.size，将主题发送到目标 Kafka 集群，以更好地容纳大量信息。

重要

生产者和消费者配置选项取决于 MirrorMaker 2 的实施，并可能随时更改。

下表描述了每个连接器的生产者和消费者，以及您可以添加配置的位置。

表 9.4. 源连接器生成者和消费者

类型	Description	配置
制作者	将主题信息发送到目标 Kafka 集群。在处理大量数据时，请考虑调整此制作者的配置。	**mirrors.sourceConnector.config: producer.override prerequisites**
制作者	写入 **offset-syncs** 主题，它将映射复制主题分区的源和目标偏移。	**mirrors.sourceConnector.config: producer.***
消费者	从源 Kafka 集群检索主题信息。	**mirrors.sourceConnector.config: consumer.***

表 9.5. Checkpoint 连接器生成者和消费者

类型	Description	配置
制作者	发送消费者偏移检查点。	**mirrors.checkpointConnector.config: producer.override prerequisites**
消费者	加载 **offset-syncs** 主题。	**mirrors.checkpointConnector.config: consumer.***

注意

您可以将 offset-syncs.topic.location 设置为 target 来使用目标 Kafka 集群作为 offset-syncs 主题的位置。

表 9.6. heartbeat 连接器制作者

类型	Description	配置

类型	Description	配置
制作者	发送心跳。	**mirrors.heartbeatConnector.config: producer.override prerequisites**

以下示例演示了如何配置制作者和消费者。

连接器制作者和消费者的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  version: 3.7.0
  # ...
  mirrors:
  - sourceCluster: "my-cluster-source"
    targetCluster: "my-cluster-target"
    sourceConnector:
      tasksMax: 5
      config:
        producer.override.batch.size: 327680
        producer.override.linger.ms: 100
        producer.request.timeout.ms: 30000
        consumer.fetch.max.bytes: 52428800
        # ...
    checkpointConnector:
      config:
        producer.override.request.timeout.ms: 30000
        consumer.max.poll.interval.ms: 300000
        # ...
    heartbeatConnector:
      config:
        producer.override.request.timeout.ms: 30000
        # ...
```

9.7.5. 指定最大数据复制任务数

连接器创建负责在 Kafka 中移动数据的任务。每个连接器由一个或多个任务组成，它们分布到运行任务的一组 worker pod 中。在复制大量分区或同步大量消费者组的偏移时，增加任务数量可以帮助解决性

能问题。

任务并行运行。为 worker 分配一个或多个任务。单个任务由一个 worker pod 处理，因此您不需要多个 worker pod 超过任务。如果有多个任务，worker 会处理多个任务。

您可以使用 tasksMax 属性指定 MirrorMaker 配置中的最大连接器任务数量。在不指定最大任务数量的情况下，默认设置是单个任务。

heartbeat 连接器始终使用单个任务。

为源和检查点连接器启动的任务数量是最大可能任务数和 tasksMax 的值之间的较低值。对于源连接器，可能的最大任务数是从源集群复制的每个分区。对于检查点连接器，可能的最大任务数都是从源集群复制的每个消费者组。在设置最多任务数量时，请考虑分区数量和支持进程的硬件资源。

如果基础架构支持处理开销，增加任务数量可以提高吞吐量和延迟。例如，添加更多任务可减少在有大量分区或消费者组时轮询源集群所需的时间。

当您有大量分区时，为源连接器增加任务数量很有用。

为源连接器增加任务数量

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  # ...
  mirrors:
  - sourceCluster: "my-cluster-source"
    targetCluster: "my-cluster-target"
    sourceConnector:
      tasksMax: 10
  # ...
```

当您有大量消费者组时，为检查点连接器增加任务数量很有用。

增加检查点连接器的任务数量

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  # ...
  mirrors:
  - sourceCluster: "my-cluster-source"
    targetCluster: "my-cluster-target"
    checkpointConnector:
      tasksMax: 10
  # ...
```

默认情况下，MirrorMaker 2 每 10 分钟检查新消费者组。您可以调整 refresh.groups.interval.seconds 配置以更改频率。在调整降低时请小心。更频繁的检查可能会对性能造成负面影响。

9.7.5.1. 检查连接器任务操作

如果使用 Prometheus 和 Grafana 监控部署，您可以检查 MirrorMaker 2 性能。由 Apache Kafka 的 Streams 提供的 MirrorMaker 2 Grafana 仪表板示例显示了以下与任务和延迟相关的指标。

- 任务数量
- 复制延迟
- 偏移同步延迟

其他资源

- *第 21 章 为 Apache Kafka 的 Streams 设置指标和仪表板*

9.7.6. 为远程主题同步 ACL 规则

将 MirrorMaker 2 与 Apache Kafka 的 Streams 搭配使用时，可以同步远程主题的 ACL 规则。但是，只有在您没有使用 User Operator 时，此功能才可用。

如果您使用 类型：在没有 User Operator 的情况下进行简单 授权，管理对代理的访问的 ACL 规则也适用于远程主题。这意味着，对源主题具有读取访问权限的用户也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。

9.7.7. 保护 Kafka MirrorMaker 2 部署

此流程描述了概述保护 MirrorMaker 2 部署所需的配置。

源 Kafka 集群和目标 Kafka 集群需要单独的配置。您还需要单独的用户配置来提供 MirrorMaker 连接到源和目标 Kafka 集群所需的凭证。

对于 Kafka 集群，您可以为 OpenShift 集群内的安全连接指定内部监听程序，以及用于 OpenShift 集群外的连接的外部监听程序。

您可以配置身份验证和授权机制。为源和目标 Kafka 集群实施的安全选项必须与为 MirrorMaker 2 实施的安全选项兼容。

创建集群和用户身份验证凭证后，您可以在 MirrorMaker 配置中指定它们以进行安全连接。

注意

在此过程中，会使用 Cluster Operator 生成的证书，但 您可以通过安装自己的证书 来替换它们。您还可以将侦听器 配置为使用由外部 CA （证书颁发机构）管理的 Kafka 侦听器证书。

开始前

在开始这个过程前，请查看 Streams for Apache Kafka 提供 的示例配置文件。它们包括使用 mTLS 或 SCRAM-SHA-512 身份验证保护 MirrorMaker 2 部署的示例。示例指定用于在 OpenShift 集群内连接

的内部监听程序。

这个示例还为完整的授权提供配置，包括允许用户在源和目标 Kafka 集群上操作的 ACL。

在配置对源和目标 Kafka 集群的用户访问时，ACL 必须授予对内部 MirrorMaker 2 连接器和目标集群组和目标集群中底层 Kafka Connect 框架使用的内部主题的访问权限。如果您重命名了集群组或内部主题，比如 为多个实例配置 MirrorMaker 2 时，请在 ACL 配置中使用这些名称。

简单授权使用由 Kafka AclAuthorizer 和 StandardAuthorizer 插件管理的 ACL 规则来确保适当的访问级别。有关将 KafkaUser 资源配置为使用简单授权的更多信息，请参阅 AclRule 模式参考。

先决条件

- Apache Kafka 的流正在运行
- 源和目标集群的独立命名空间

此流程假设将源和目标集群安装到单独的命名空间。如果要使用 Topic Operator，则需要执行此操作。主题 Operator 只监视指定命名空间中的单个集群。

通过将集群划分为命名空间，您需要复制集群 secret，以便可以在命名空间外访问它们。您需要引用 MirrorMaker 配置中的 secret。

流程

1. 配置两个 Kafka 资源，一个用于保护源 Kafka 集群，另一个用于保护目标 Kafka 集群。

   您可以为身份验证和启用授权添加监听程序配置。

   在本例中，为带有 TLS 加密和 mTLS 身份验证的 Kafka 集群配置了内部监听程序。启用 Kafka 简单 授权。

   使用 TLS 加密和 mTLS 身份验证的源 Kafka 集群配置示例

   -

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-source-cluster
spec:
  kafka:
    version: 3.7.0
    replicas: 1
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
    authorization:
      type: simple
    config:
      offsets.topic.replication.factor: 1
      transaction.state.log.replication.factor: 1
      transaction.state.log.min.isr: 1
      default.replication.factor: 1
      min.insync.replicas: 1
      inter.broker.protocol.version: "3.7"
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
  zookeeper:
    replicas: 1
    storage:
      type: persistent-claim
      size: 100Gi
      deleteClaim: false
  entityOperator:
    topicOperator: {}
    userOperator: {}
```

使用 TLS 加密和 mTLS 身份验证的目标 Kafka 集群配置示例

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

```
    version: 3.7.0
    replicas: 1
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
    authorization:
      type: simple
    config:
      offsets.topic.replication.factor: 1
      transaction.state.log.replication.factor: 1
      transaction.state.log.min.isr: 1
      default.replication.factor: 1
      min.insync.replicas: 1
      inter.broker.protocol.version: "3.7"
    storage:
      type: jbod
      volumes:
        - id: 0
          type: persistent-claim
          size: 100Gi
          deleteClaim: false
  zookeeper:
    replicas: 1
    storage:
      type: persistent-claim
      size: 100Gi
      deleteClaim: false
  entityOperator:
    topicOperator: {}
    userOperator: {}
```

2. 在单独的命名空间中创建或更新 Kafka 资源。

```
oc apply -f <kafka_configuration_file> -n <namespace>
```

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。

证书在 secret < *cluster_name*> -cluster-ca-cert 中创建。

3. 配置两个 KafkaUser 资源，一个用于源 Kafka 集群的用户，另一个用于目标 Kafka 集群的

用户。

a. 配置与对应的源和目标 Kafka 集群相同的身份验证和授权类型。例如，如果您在源 Kafka 集群的 Kafka 配置中使用了 tls 验证和 simple 授权类型，请在 KafkaUser 配置中使用相同的。

b. 配置 MirrorMaker 2 所需的 ACL，以允许对源和目标 Kafka 集群执行操作。

mTLS 验证的源用户配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-source-user
  labels:
    strimzi.io/cluster: my-source-cluster
spec:
  authentication:
    type: tls
  authorization:
    type: simple
    acls:
      # MirrorSourceConnector
      - resource: # Not needed if offset-syncs.topic.location=target
          type: topic
          name: mm2-offset-syncs.my-target-cluster.internal
        operations:
          - Create
          - DescribeConfigs
          - Read
          - Write
      - resource: # Needed for every topic which is mirrored
          type: topic
          name: "*"
        operations:
          - DescribeConfigs
          - Read
      # MirrorCheckpointConnector
      - resource:
          type: cluster
        operations:
          - Describe
      - resource: # Needed for every group for which offsets are synced
          type: group
          name: "*"
        operations:
          - Describe
      - resource: # Not needed if offset-syncs.topic.location=target
```

```
      type: topic
      name: mm2-offset-syncs.my-target-cluster.internal
    operations:
      - Read
```

mTLS 验证的目标用户配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-target-user
  labels:
    strimzi.io/cluster: my-target-cluster
spec:
  authentication:
    type: tls
  authorization:
    type: simple
    acls:
      # cluster group
      - resource:
          type: group
          name: mirrormaker2-cluster
        operations:
          - Read
      # access to config.storage.topic
      - resource:
          type: topic
          name: mirrormaker2-cluster-configs
        operations:
          - Create
          - Describe
          - DescribeConfigs
          - Read
          - Write
      # access to status.storage.topic
      - resource:
          type: topic
          name: mirrormaker2-cluster-status
        operations:
          - Create
          - Describe
          - DescribeConfigs
          - Read
          - Write
      # access to offset.storage.topic
      - resource:
          type: topic
          name: mirrormaker2-cluster-offsets
```

```
    operations:
      - Create
      - Describe
      - DescribeConfigs
      - Read
      - Write
  # MirrorSourceConnector
  - resource: # Needed for every topic which is mirrored
      type: topic
      name: "*"
    operations:
      - Create
      - Alter
      - AlterConfigs
      - Write
  # MirrorCheckpointConnector
  - resource:
      type: cluster
    operations:
      - Describe
  - resource:
      type: topic
      name: my-source-cluster.checkpoints.internal
    operations:
      - Create
      - Describe
      - Read
      - Write
  - resource: # Needed for every group for which the offset is synced
      type: group
      name: "*"
    operations:
      - Read
      - Describe
  # MirrorHeartbeatConnector
  - resource:
      type: topic
      name: heartbeats
    operations:
      - Create
      - Describe
      - Write
```

注意

您可以通过将 type 设置为 tls-external 来使用 User Operator 外部发布的证书。如需更多信息，请参阅 KafkaUserSpec 模式参考。

4. 在您为源和目标 Kafka 集群创建的每个命名空间中创建或更新 KafkaUser 资源。

```
oc apply -f <kafka_user_configuration_file> -n <namespace>
```

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

User Operator 创建一个名称与 KafkaUser 资源相同的新 secret。secret 包含 mTLS 验证的私钥和公钥。公钥包含在用户证书中，该证书由客户端 CA 签名。

5. 使用身份验证详情配置 KafkaMirrorMaker2 资源，以连接到源和目标 Kafka 集群。

带有 TLS 加密和 mTLS 身份验证的 MirrorMaker 2 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker-2
spec:
  version: 3.7.0
  replicas: 1
  connectCluster: "my-target-cluster"
  clusters:
    - alias: "my-source-cluster"
      bootstrapServers: my-source-cluster-kafka-bootstrap:9093
      tls: 1
        trustedCertificates:
          - secretName: my-source-cluster-cluster-ca-cert
            certificate: ca.crt
      authentication: 2
        type: tls
        certificateAndKey:
          secretName: my-source-user
          certificate: user.crt
          key: user.key
    - alias: "my-target-cluster"
      bootstrapServers: my-target-cluster-kafka-bootstrap:9093
      tls: 3
        trustedCertificates:
          - secretName: my-target-cluster-cluster-ca-cert
            certificate: ca.crt
      authentication: 4
        type: tls
        certificateAndKey:
          secretName: my-target-user
```

```
        certificate: user.crt
        key: user.key
    config:
      # -1 means it will use the default replication factor configured in the broker
      config.storage.replication.factor: -1
      offset.storage.replication.factor: -1
      status.storage.replication.factor: -1
mirrors:
  - sourceCluster: "my-source-cluster"
    targetCluster: "my-target-cluster"
    sourceConnector:
      config:
        replication.factor: 1
        offset-syncs.topic.replication.factor: 1
        sync.topic.acls.enabled: "false"
    heartbeatConnector:
      config:
        heartbeats.topic.replication.factor: 1
    checkpointConnector:
      config:
        checkpoints.topic.replication.factor: 1
        sync.group.offsets.enabled: "true"
    topicsPattern: "topic1|topic2|topic3"
    groupsPattern: "group1|group2|group3"
```

1. 源 Kafka 集群的 TLS 证书。如果它们位于单独的命名空间中，请将集群 secret 从 Kafka 集群的命名空间中复制。
2. 使用 TLS 机制访问源 Kafka 集群的用户身份验证。
3. 目标 Kafka 集群的 TLS 证书。
4. 访问目标 Kafka 集群的用户身份验证。

6. 在与目标 Kafka 集群相同的命名空间中创建或更新 KafkaMirrorMaker2 资源。

```
oc apply -f <mirrormaker2_configuration_file> -n <namespace_of_target_cluster>
```

9.7.8. 手动停止或暂停 MirrorMaker 2 连接器

如果您使用 KafkaMirrorMaker2 资源来配置内部 MirrorMaker 连接器，请使用 state 配置来停止或暂停连接器。与连接器和任务保持实例化的暂停状态不同，停止连接器只保留配置，且没有活跃进程。从运行停止连接器可能更适合长时间运行，而不是只暂停。虽然暂停的连接器速度更快恢复，但已停止的连接器具有释放内存和资源的优点。

注意

state 配置替换 KafkaMirrorMaker2ConnectorSpec 模式中的（已弃用） pause 配置，它允许暂停连接器。如果您之前使用 pause 配置来暂停连接器，我们建议您只使用 state 配置过渡到 以避免冲突。

先决条件

- Cluster Operator 正在运行。

流程

1. 查找控制要暂停或停止 MirrorMaker 2 连接器的 KafkaMirrorMaker2 自定义资源的名称：

```
oc get KafkaMirrorMaker2
```

2. 编辑 KafkaMirrorMaker2 资源，以停止或暂停连接器。

停止 MirrorMaker 2 连接器的配置示例

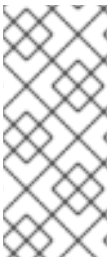
```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  version: 3.7.0
  replicas: 3
  connectCluster: "my-cluster-target"
  clusters:
    # ...
  mirrors:
  - sourceCluster: "my-cluster-source"
    targetCluster: "my-cluster-target"
    sourceConnector:
      tasksMax: 10
```

```
    autoRestart:
      enabled: true
    state: stopped
  # ...
```

将 state 配置更改为 stopped 或 paused。当此属性没有设置时，连接器的默认状态为 running。

3. 将更改应用到 KafkaMirrorMaker2 配置。

您可以通过将 state 改为 running，或删除配置来恢复连接器。



注意

另外，您可以 公开 Kafka Connect API，并使用 stop 和 pause 端点停止连接器运行。例如，PUT /connectors/<connector_name>/stop。然后，您可以使用 resume 端点重启它。

9.7.9. 手动重启 MirrorMaker 2 连接器

使用 strimzi.io/restart-connector 注解来手动触发 MirrorMaker 2 连接器的重启。

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 连接器的 KafkaMirrorMaker2 自定义资源的名称：

```
oc get KafkaMirrorMaker2
```

2. 查找要从 KafkaMirrorMaker2 自定义资源重启的 Kafka MirrorMaker 2 连接器的名称：

```
oc describe KafkaMirrorMaker2 <mirrormaker_cluster_name>
```

3. 通过在 OpenShift 中注解 KafkaMirrorMaker2 资源，使用连接器的名称来重启连接器：

```
oc annotate KafkaMirrorMaker2 <mirrormaker_cluster_name> "strimzi.io/restart-connector=<mirrormaker_connector_name>"
```

在本例中，my-mirror-maker-2 集群中的连接器 my-connector 被重启：

```
oc annotate KafkaMirrorMaker2 my-mirror-maker-2 "strimzi.io/restart-connector=my-connector"
```

4. 等待下一个协调发生（默认为两分钟）。

MirrorMaker 2 连接器会重启，只要协调过程检测到注解。当 MirrorMaker 2 接受请求时，注解会从 KafkaMirrorMaker2 自定义资源中删除。

9.7.10. 手动重启 MirrorMaker 2 连接器任务

使用 strimzi.io/restart-connector-task 注解来手动触发 MirrorMaker 2 连接器的重启。

先决条件

- Cluster Operator 正在运行。

流程

1. 查找控制您要重启的 MirrorMaker 2 连接器任务的 KafkaMirrorMaker2 自定义资源的名称：

```
oc get KafkaMirrorMaker2
```

2. 查找连接器的名称和要从 KafkaMirrorMaker2 自定义资源重启的任务 ID：

```
oc describe KafkaMirrorMaker2 <mirrormaker_cluster_name>
```

任务 ID 是非负整数，从 0 开始。

3. 通过在 OpenShift 中注解 KafkaMirrorMaker2 资源，使用名称和 ID 重启连接器任务：

```
oc annotate KafkaMirrorMaker2 <mirrormaker_cluster_name> "strimzi.io/restart-connector-task=<mirrormaker_connector_name>:<task_id>"
```

在本例中，在 my-mirror-maker-2 集群中连接器 my-connector 的任务 0 被重启：

```
oc annotate KafkaMirrorMaker2 my-mirror-maker-2 "strimzi.io/restart-connector-task=my-connector:0"
```

4. 等待下一个协调发生（默认为两分钟）。

MirrorMaker 2 连接器任务会重启，只要协调过程检测到注解。当 MirrorMaker 2 接受请求时，注解会从 KafkaMirrorMaker2 自定义资源中删除。

9.8. 配置 KAFKA MIRRORMAKER （已弃用）

更新 KafkaMirrorMaker 自定义资源的 spec 属性，以配置 Kafka MirrorMaker 部署。

您可以使用 TLS 或 SASL 身份验证为生产者和消费者配置访问控制。此流程演示了如何在消费者和生成器端使用 TLS 加密和 mTLS 身份验证的配置。

要深入了解 Kafka MirrorMaker 集群配置选项，请参阅 Apache Kafka 自定义资源 API 参考。

重要

Kafka MirrorMaker 1 （称为文档中的 *MirrorMaker*）已在 Apache Kafka 3.0.0 中弃用，并将在 Apache Kafka 4.0.0 中删除。因此，用于部署 Kafka MirrorMaker 1 的 KafkaMirrorMaker 自定义资源也已在 Apache Kafka 的 Streams 中弃用。当使用 Apache Kafka 4.0.0 时，KafkaMirrorMaker 资源将从 Apache Kafka 的 Streams 中删除。作为替代方法，在 IdentityReplicationPolicy 中使用 KafkaMirrorMaker2 自定义资源。

KafkaMirrorMaker 自定义资源配置示例

```
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
  producer:
    bootstrapServers: my-target-cluster-kafka-bootstrap:9092
    abortOnSendFailure: false (9)
    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
  include: "my-topic|other-topic" (10)
  resources: (11)
    requests:
      cpu: "1"
      memory: 2Gi
    limits:
      cpu: "2"
      memory: 2Gi
  logging: (12)
    type: inline
    loggers:
```

```
    mirrormaker.root.logger: INFO
readinessProbe: 13
  initialDelaySeconds: 15
  timeoutSeconds: 5
livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
metricsConfig: 14
 type: jmxPrometheusExporter
 valueFrom:
   configMapKeyRef:
     name: my-config-map
     key: my-key
jvmOptions: 15
  "-Xmx": "1g"
  "-Xms": "1g"
image: my-org/my-image:latest 16
template: 17
  pod:
    affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
                - key: application
                  operator: In
                  values:
                    - postgresql
                    - mongodb
            topologyKey: "kubernetes.io/hostname"
  mirrorMakerContainer: 18
    env:
      - name: OTEL_SERVICE_NAME
        value: my-otel-service
      - name: OTEL_EXPORTER_OTLP_ENDPOINT
        value: "http://otlp-host:4317"
tracing: 19
  type: opentelemetry
```

1

副本节点的数量。

2

用于消费者和制作者的 Bootstrap 服务器。

3

消费者的组 ID。

4

消费者流的数量。

5

偏移 auto-commit 间隔（以毫秒为单位）。

6

TLS 加密，使用密钥名称，其中 TLS 证书存储为 X.509 格式，用于消费者或生成者。如果证书存储在同一 secret 中，则可以多次列出。

7

为消费者或生成者（指定为 mTLS、基于令牌的 OAuth、基于 SASL 的 SCRAM-SHA-256/SCRAM-SHA-512 或 PLAIN）进行身份验证。

8

consumer 和 producer 的 Kafka 配置选项。

9

如果将 abortOnSendFailure 属性设置为 true，则 Kafka MirrorMaker 将退出，容器将按照消息发送失败重启。

10

从源镜像到目标 Kafka 集群包含的主题列表。

11

为保留支持的资源（当前 cpu 和 memory ）的请求，以及指定可消耗的最大资源的限制。

12

指定日志记录器和日志级别直接(内联)或通过 ConfigMap 间接添加(外部)。自定义 Log4j 配置必须放在 ConfigMap 中的 log4j.properties 或 log4j2.properties 键下。MirrorMaker 只有一个日志记录器，名为 mirrormaker.root.logger。您可以将日志级别设置为 INFO, ERROR, WARN, TRACE, DEBUG, FATAL 或 OFF。

13

14

Prometheus 指标，通过引用包含在此示例中 Prometheus JMX 导出器配置的 ConfigMap 启用。您可以使用对 metricsConfig.valueFrom.configMapKeyRef.key 下包含空文件的 ConfigMap 的引用来启用指标。

15

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

16

ProductShortName OPTION: 容器镜像配置，仅在特殊情况下推荐使用。

17

模板自定义。此处的 pod 使用反关联性调度，因此 pod 不会调度到具有相同主机名的节点。

18

为分布式追踪设置环境变量。

19

使用 OpenTelemetry 启用分布式追踪。

警告

将 abortOnSendFailure 属性设置为 false 时，生产者会尝试在主题中发送下一个消息。原始消息可能会丢失，因为没有尝试重新发送失败的消息。

9.9. 配置 KAFKA BRIDGE

更新 KafkaBridge 自定义资源的 spec 属性来配置 Kafka Bridge 部署。

为了防止在不同 Kafka Bridge 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将请求路由到正确的 Kafka Bridge 实例。另外，每个独立的 Kafka Bridge 实例都必须有副本。Kafka

Bridge 实例有自己的状态，它不与另一个实例共享。

要深入了解 Kafka Bridge 集群配置选项，请参阅 Apache Kafka 自定义资源 API 参考。

KafkaBridge 自定义资源配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  replicas: 3 (1)
  bootstrapServers: <cluster_name>-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: OTEL_SERVICE_NAME
        value: my-otel-service
      - name: OTEL_EXPORTER_OTLP_ENDPOINT
        value: "http://otlp-host:4317"
tracing:
  type: opentelemetry 16
```

1

副本节点的数量。

2

用于连接到目标 Kafka 集群的 bootstrap 服务器。使用 Kafka 集群的名称作为 <*cluster_name*>。

3

TLS 加密，使用密钥名称，其中 TLS 证书存储为源 Kafka 集群的 X.509 格式。如果证书存储在同一 secret 中，则可以多次列出。

4

5

对 Kafka 代理的 HTTP 访问。

6

CORS 访问指定所选资源和访问方法。请求中的其他 HTTP 标头描述了允许访问 Kafka 集群的源。

7

消费者配置选项。

8

制作者配置选项。

9

为保留支持的资源（当前 cpu 和 memory ）的请求，以及指定可消耗的最大资源的限制。

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)。自定义 Log4j 配置必须放在 ConfigMap 中的 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

JVM 配置选项，用于优化运行 Kafka Bridge 的虚拟机(VM)的性能。

12

检查检查以了解何时重启容器（存活度）以及何时容器可以接受流量（就绪度）。

13

可选：容器镜像配置，仅在特殊情况下推荐。

14

模板自定义。此处的 pod 使用反关联性调度，因此 pod 不会调度到具有相同主机名的节点。

15

16 使用 OpenTelemetry 启用分布式追踪。

其他资源

- 使用 Apache Kafka Bridge 的流

9.10. 配置 KAFKA 和 ZOOKEEPER 存储

Apache Kafka 的流提供了配置 Kafka 和 ZooKeeper 数据存储选项的灵活性。

支持的存储类型有：

- Ephemeral (推荐只在开发时使用)
- 持久性
- JBOD (只限 Kafka；对于 ZooKeeper 不可用)
- 分层存储(Early access)

要配置存储，您可以在组件的自定义资源中指定 storage 属性。存储类型使用 storage.type 属性设置。使用节点池时，您可以指定 Kafka 集群中使用的每个节点池的唯一存储配置。Kafka 资源可用的相同存储属性也可用于 KafkaNodePool 池资源。

分层存储通过利用具有不同特征的并行使用存储类型，为数据管理提供了更大的灵活性。例如，分层存储可能包括：

- 更高的性能和更高的成本块存储

- 降低性能并降低成本对象存储

分层存储是 Kafka 中的早期访问功能。要配置分层存储，您可以指定 分层存储 属性。分层存储仅在使用 Kafka 自定义资源的集群级别进行配置。

与存储相关的模式引用提供有关存储配置属性的更多信息：

- EphemeralStorage schema 参考
- PersistentClaimStorage schema 参考
- JbodStorage schema 参考
- TieredStorageCustom 模式参考

警告

部署 Kafka 集群后无法更改存储类型。

9.10.1. 数据存储注意事项

要使 Apache Kafka 正常工作的流，高效的数据存储基础架构非常重要。我们强烈建议您使用块存储。Apache Kafka 的流只测试用于块存储。文件存储（如 NFS）没有被测试，无法保证它可以正常工作。

为您的块存储选择以下选项之一：

- 基于云的块存储解决方案，如 Amazon Elastic Block Store (EBS)

- 使用 本地持久性卷的持久性存储
- 由 *光纤通道或 iSCSI*等协议访问的存储区域网络(SAN)卷

注意

Apache Kafka 的流不需要 OpenShift 原始块卷。

9.10.1.1. 文件系统

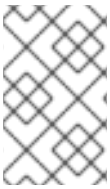
Kafka 使用文件系统来存储信息。Apache Kafka 的流与 XFS 和 ext4 文件系统兼容，它们通常与 Kafka 一起使用。在选择和设置文件系统时，请考虑部署的底层架构和要求。

如需更多信息，请参阅 Kafka 文档中的 Filesystem Selection。

9.10.1.2. 磁盘用量

为 Apache Kafka 和 ZooKeeper 使用单独的磁盘。

虽然使用固态驱动器 (SSD) 并不是必须的，但它可以在大型集群中提高 Kafka 的性能，其中数据会异步发送到多个主题，并从多个主题接收。SSD 与 ZooKeeper 特别有效，这需要快速、低延迟数据访问。



注意

您不需要置备复制存储，因为 Kafka 和 ZooKeeper 都有内置数据复制。

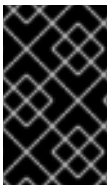
9.10.2. 临时存储

临时数据存储是临时的。节点上的所有 pod 共享本地临时存储空间。只要使用它的 pod 正在运行，数据就会保留。当 pod 被删除时，数据会丢失。虽然 pod 可以在高可用性环境中恢复数据。

由于其临时性质，仅推荐使用临时存储进行开发和测试。

临时存储使用 emptyDir 卷来存储数据。当 pod 分配给节点时，会创建一个 emptyDir 卷。您可以使

用 sizeLimit 属性为 emptyDir 设置存储总量。



重要

临时存储不适用于单节点 ZooKeeper 集群或 Kafka 主题，复制因子为 1。

要使用临时存储，您可以将 Kafka 或 ZooKeeper 资源中的存储类型配置设置为 临时。如果使用节点池，也可以在单个节点池的存储配置中指定 ephemeral。

临时存储配置示例

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

9.10.2.1. Kafka 日志目录的挂载路径

Kafka 代理使用临时卷来作为挂载到以下路径的日志目录：

```
/var/lib/kafka/data/kafka-log_IDX_
```

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

9.10.3. 持久性存储

持久数据存储在系统中断时保留数据。对于使用持久数据存储的 pod，数据会在 pod 失败后保留，并重启。由于其永久性质，建议在生产环境中使用持久性存储。

要使用 Apache Kafka 的 Streams 中的持久性存储，您可以在 Kafka 或 ZooKeeper 资源的存储配置中指定 persistent-claim。如果使用节点池，也可以在单个节点池的存储配置中指定 persistent-claim。

您可以配置资源，以便 pod 使用 持久性卷声明 (PVC) 在持久性卷 (PV) 上发出存储请求。PV 代表按需创建的存储卷，并独立于使用它们的 pod。PVC 请求创建 pod 时所需的存储量。PV 的底层存储基础架构不需要理解。如果 PV 与存储条件匹配，PVC 会绑定到 PV。

您有两个选项来指定存储类型：

storage.type: persistent-claim

如果您选择 persistent-claim 作为存储类型，则会定义一个持久性存储卷。

storage.type: jbod

当您选择 jbod 作为存储类型时，您可以灵活地使用唯一 ID 定义一组持久性存储卷。

在生产环境中，建议配置以下内容：

- 对于 Kafka 或节点池，使用一个或多个持久性卷将 storage.type 设置为 jbod。
- 对于 ZooKeeper，将 storage.type 设置为单个持久性卷的 persistent-claim。

持久性存储还具有以下配置选项：

id （可选）

存储标识号。对于 JBOD 存储声明中定义的存储卷，这个选项是必须的。默认值为 0。

Size （必需）

持久性卷声明的大小，如 "1000Gi"。

类 （可选）

PVC 可以通过指定 StorageClass 来请求不同类型的持久性存储。存储类定义存储配置集，并根据该配置集动态置备 PV。如果没有指定存储类，则使用 OpenShift 集群中标记为 default 的存储

类。持久性存储选项可能包括 SAN 存储类型或本地持久性卷。

selector （可选）

配置以指定特定 PV。提供 key:value 对，代表所选卷的标签。

deleteClaim (optional)

布尔值，用于指定在卸载集群时是否删除 PVC。默认为 false。

警告

只有在支持持久性卷大小的 OpenShift 版本中，才支持增大 Apache Kafka 集群现有流中的持久性卷大小。要重新定义大小的持久性卷必须使用支持卷扩展的存储类。对于不支持卷扩展的 OpenShift 和存储类的其他版本，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。

Kafka 和 ZooKeeper 持久性存储配置示例

```
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:
    storage:
```

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

使用特定存储类的持久性存储配置示例

```
# ...
storage:
  type: persistent-claim
  size: 500Gi
  class: my-storage-class
# ...
```

使用选择器 (selector)来指定提供某些功能的标记的持久性卷，如 SSD。

使用选择器的持久性存储配置示例

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

9.10.3.1. 存储类覆盖

您可以为一个或多个 Kafka 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，这非常有用。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 ：

带有类覆盖的存储配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  labels:
    app: my-cluster
  name: my-cluster
  namespace: myproject
spec:
  # ...
  kafka:
    replicas: 3
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
        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
  # ...
```

由于配置的 overrides 属性，卷使用以下存储类：

- ZooKeeper 节点 0 的持久性卷使用 my-storage-class-zone-1a。
- ZooKeeper 节点 1 的持久性卷使用 my-storage-class-zone-1b。
- ZooKeeper 节点 2 的持久性卷使用 my-storage-class-zone-1c。
- Kafka 代理 0 的持久性卷使用 my-storage-class-zone-1a。
- Kafka 代理 1 的持久性卷使用 my-storage-class-zone-1b。
- Kafka 代理 2 的持久性卷使用 my-storage-class-zone-1c。

overrides 属性目前仅用于覆盖存储类。目前不支持对其他存储配置属性覆盖。

9.10.3.2. 持久性存储的 PVC 资源

使用持久性存储时，它会使用以下名称创建 PVC：

data-*cluster-name*-kafka-*idx*

用于存储 Kafka 代理 pod *idx* 数据的卷的 PVC。

data-*cluster-name*-zookeeper-*idx*

用于为 ZooKeeper 节点 pod *idx* 存储数据的卷的 PVC。

9.10.3.3. Kafka 日志目录的挂载路径

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-log*IDX*
```

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

9.10.4. 重新调整持久性卷大小

只要存储基础架构支持，就可以调整集群使用的持久性卷而不造成数据丢失的风险。在进行了配置更新以更改存储的大小后，Apache Kafka 的 Streams 指示存储基础架构进行更改。使用 persistent-claim 卷的 Apache Kafka 集群支持存储扩展。

只有在每个代理使用多个磁盘时，才能减少存储。在将磁盘中的所有分区移动到同一代理(intra-broker)或同一集群（集群内）中的其他代理后，您可以删除磁盘。

重要

您无法缩小持久性卷的大小，因为它目前在 OpenShift 中不被支持。

先决条件

- 支持调整大小的 OpenShift 集群。
- Cluster Operator 正在运行。
- 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. 编辑集群的 Kafka 资源。

   更改 size 属性，以增加分配给 Kafka 集群、ZooKeeper 集群或两者的持久性卷大小。

   - 对于 Kafka 集群，更新 spec.kafka.storage 下的 size 属性。
   - 对于 ZooKeeper 集群，更新 spec.zookeeper.storage 下的 size 属性。

   将卷大小增加到 2000Gi的 Kafka 配置

```
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_configuration_file>
```

OpenShift 增加所选持久性卷的容量，以响应 Cluster Operator 的请求。完成调整大小后，Cluster Operator 会重启所有使用调整大小的持久性卷的 pod。这会自动发生。

3. 验证集群中相关 pod 的存储容量是否已增加：

```
oc get pv
```

带有增加存储的 Kafka 代理 pod

```
NAME             CAPACITY   CLAIM
pvc-0ca459ce-...   2000Gi     my-project/data-my-cluster-kafka-2
pvc-6e1810be-...   2000Gi     my-project/data-my-cluster-kafka-0
pvc-82dc78c9-...   2000Gi     my-project/data-my-cluster-kafka-1
```

输出显示了与代理 pod 关联的每个 PVC 的名称。

其他资源

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

9.10.5. JBOD 存储

JBOD 存储允许您将 Kafka 集群配置为使用多个磁盘或卷。这个方法为 Kafka 代理提供了数据存储容量，并可能导致性能改进。JBOD 配置由一个或多个卷定义，每个卷可以是 临时或持久。JBOD 卷声明的规则和约束与临时存储和持久性存储的规则和约束相同。例如，在置备后，您无法缩小持久性存储卷的大小，当类型为 临时 时，您无法更改 sizeLimit 的值。

注意

对 JBOD 存储的支持仅限 Kafka，不支持 ZooKeeper。

要使用 JBOD 存储，您可以将 Kafka 资源中的存储类型配置设置为 jbod。如果使用节点池，也可以在各个节点池的存储配置中指定 jbod。

volumes 属性允许您描述组成 JBOD 存储阵列或配置的磁盘。

JBOD 存储配置示例

```
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
  # ...
```

创建 JBOD 卷后无法更改 ID。您可以从 JBOD 配置中添加或删除卷。

9.10.5.1. JBOD 存储的 PVC 资源

当持久性存储用于声明 JBOD 卷时，它会创建一个具有以下名称的 PVC：

data-*id*-*cluster-name*-kafka-*idx*

用于存储 Kafka 代理 pod *idx* 数据的卷的 PVC。*id* 是用于存储 Kafka 代理 pod 数据的卷的 ID。

9.10.5.2. Kafka 日志目录的挂载路径

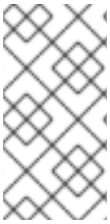
Kafka 代理使用 JBOD 卷作为挂载到以下路径的日志目录：

```
/var/lib/kafka/data-id/kafka-logidx
```

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

9.10.6. 将卷添加到 JBOD 存储

此流程描述了如何将卷添加到配置为使用 JBOD 存储的 Kafka 集群中。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

当在过去和删除的 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

- 一个 OpenShift 集群

- 正在运行的 Cluster Operator
- 具有 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 数组中。例如，使用 id 2 添加新卷：

```
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_configuration_file>
```

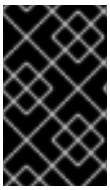
3. 创建新主题或将现有分区重新分配给新磁盘。

提示

Cruise Control 是一个重新分配分区的有效工具。要执行 intra-broker 磁盘平衡，您可以在 KafkaRebalance.spec 下将 rebalanceDisk 设置为 true。

9.10.7. 从 JBOD 存储中删除卷

此流程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 存储始终必须包含至少一个卷。



重要

为了避免数据丢失，您必须在删除卷前移动所有分区。

先决条件

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

流程

1. 从您要删除的磁盘中重新分配所有分区。分区中的任何数据仍被分配给要删除的磁盘。

提示

您可以使用 kafka-reassign-partitions.sh 工具重新分配分区。

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，使用 ids 1 和 2 删除卷：

```
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_configuration_file>
```

9.10.8. 分层存储（早期访问）

分层存储引入了一种灵活的方法，用于管理 Kafka 数据，其中日志片段被移到单独的存储系统。例如，您可以在代理中使用块存储，以频繁访问块存储中的数据，从块存储中卸载旧或更频繁的数据，到更经济、可扩展的远程存储解决方案，如 Amazon S3，而不影响数据可访问性和持久性。

警告

分层存储是一个早期访问 Kafka 功能，它也可用于 Apache Kafka 的流。由于其当前限制，不建议在生产环境中使用。

分层存储需要实现 Kafka 的 RemoteStorageManager 接口来处理 Kafka 和远程存储系统之间的通信，这通过配置 Kafka 资源启用。在启用了自定义分层存储时，Apache Kafka 的 Streams 使用 Kafka 的 TopicBasedRemoteLogMetadataManager for Remote Log Metadata Management (RLMM)。RLMM 管理与远程存储相关的元数据。

要使用自定义分层存储，请执行以下操作：

- 通过构建自定义容器镜像，在 Apache Kafka 镜像的 Streams 中包括 Kafka 的分层存储插件。该插件必须为由 Apache Kafka 管理的 Kafka 集群提供必要的功能，才能与分层存储解决方

案交互。

- 使用 Kafka 资源中分层存储属性为 分层存储 配置 Kafka。指定自定义 RemoteStorageManager 实现的类名称和路径，以及任何其他配置。
- 如果需要，指定特定于 RLMM 的分层存储配置。

Kafka 的自定义分层存储配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    tieredStorage:
      type: custom 1
      remoteStorageManager: 2
        className: com.example.kafka.tiered.storage.s3.S3RemoteStorageManager
        classPath: /opt/kafka/plugins/tiered-storage-s3/*
        config:
          storage.bucket.name: my-bucket 3
          # ...
    config:
      rlmm.config.remote.log.metadata.topic.replication.factor: 1 4
  # ...
```

1 类型 必须设置为 自定义。

2 自定义 RemoteStorageManager 实现的配置，包括类名称和路径。

3 配置传递给自定义 RemoteStorageManager 实现， Apache Kafka 的 Streams 会自动使用 rsm.config 前缀。

4

9.11. 配置 CPU 和内存限值和请求

默认情况下，Apache Kafka Cluster Operator 的 Streams 不会为其部署的操作对象指定 CPU 和内存资源请求和限值。确保足够分配资源对于在 Kafka 中保持稳定性并获得最佳性能至关重要。理想的资源分配取决于您的特定要求和用例。

建议通过 设置适当的请求和限值，为每个容器配置 CPU 和内存资源。

9.12. 配置 POD 调度

为了避免同一 OpenShift 节点上调度的应用程序之间的资源冲突导致性能下降，您可以独立于关键工作负载调度 Kafka pod。这可以通过选择特定节点或专门用于 Kafka 的一组节点来实现。

9.12.1. 指定关联性、容限和拓扑分布限制

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、tolerations 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
- 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 规格。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

其他资源

- 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

9.12.1.1. 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程序永远不会调度到同一磁盘上。在运行 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共享节点，如数据库。

9.12.1.2. 使用节点关联性将工作负载调度到特定的节点上

OpenShift 集群通常由许多不同类型的 worker 节点组成。有些工作负载针对 CPU 重度工作负载（某些用于内存）进行了优化，另一个则针对存储（快速本地 SSD）或网络进行了优化。使用不同节点有助于优化成本和性能。要达到最佳可能的性能，务必要允许为 Apache Kafka 组件调度流以使用正确的节点。

OpenShift 使用节点关联性将工作负载调度到特定的节点上。节点关联性允许您为要在其上调度 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）来指定该标签，以选择正确的节点。

9.12.1.3. 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来在专用节点上调度 Kafka pod。

集群管理员可以将所选 OpenShift 节点标记为污点。具有污点的节点不包括在常规调度中，常规 pod 不会被调度到它们上运行。只有可以容许节点上污点集的服务才能调度到该节点上。此类节点上运行的其他服务是唯一一个系统服务，如日志收集器或软件定义的网络。

在专用节点上运行 Kafka 及其组件会有很多优点。没有其他应用程序在同一节点上运行，这可能会导致距离或消耗 Kafka 所需的资源。从而提高了性能和稳定性。

9.12.2. 配置 pod 反关联性，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，则它们将同时不可用。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

- 一个 OpenShift 集群
- 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 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"
  # ...
```

其中 *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 节点不共享同一 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"
  # ...
```

其中 *CLUSTER-NAME* 是 Kafka 自定义资源的名称。

3. 创建或更新资源。

```
oc apply -f <kafka_configuration_file>
```

9.12.3. 在 Kafka 组件中配置 pod 反关联性

Pod 反关联性配置有助于 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，您要避免 Kafka 在与其他网络或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。

先决条件

- 一个 OpenShift 集群
- 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。使用标签指定不应在同一节点上调度的 pod。topologyKey 应设置为 kubernetes.io/hostname，以指定所选 pod 不应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。

   这可以通过 oc apply 来完成：

```
oc apply -f <kafka_configuration_file>
```

9.12.4. 在 Kafka 组件中配置节点关联性

先决条件

- 一个 OpenShift 集群
- 正在运行的 Cluster Operator

流程

1. 标记应该调度 Apache Kafka 组件的 Streams 节点。

   这可以通过 oc label 来完成：

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

   或者，也可以重复使用一些现有标签。

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。

   这可以通过 **oc apply** 来完成：

```
oc apply -f <kafka_configuration_file>
```

9.12.5. 设置专用节点并在其上调度 pod

先决条件

- 一个 OpenShift 集群
- 正在运行的 Cluster Operator

流程

1. 选择应用作专用的节点。
2. 确保没有在这些节点上调度工作负载。
3. 在所选节点上设置污点：

   这可以通过 **oc adm taint** 来完成：

```
oc adm taint node NAME-OF-NODE dedicated=Kafka:NoSchedule
```

4. 另外，还要为所选节点添加标签。

   这可以通过 oc label 来完成：

```
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。

   这可以通过 oc apply 来完成：

```
oc apply -f <kafka_configuration_file>
```

9.13. 配置日志记录级别

在 Kafka 组件和 Apache Kafka operator 的 Streams 的自定义资源中配置日志记录级别。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 的优点在于，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 为 Apache Kafka Operator 指定 Streams 的日志记录器，您也可以附加日志记录规格来添加过滤器。

您可以在日志记录规格中指定 日志类型 ：

- 直接指定日志记录级别时的 内联
- 引用 ConfigMap 时的外部

内联 日志记录配置示例

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

外部日志记录 配置示例

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

ConfigMap 的 name 和 key 的值是必需的。如果没有设置 name 或 key，则会使用默认日志记录。

9.13.1. Kafka 组件和 Operator 的日志记录选项

有关为特定 Kafka 组件或 Operator 配置日志记录的更多信息，请参阅以下部分。

Kafka 组件日志记录

- Kafka 日志记录
- ZooKeeper 日志记录
- Kafka Connect 和 MirrorMaker 2 日志记录
- MirrorMaker 日志记录
- Kafka Bridge 日志记录
- Cruise Control 日志记录

Operator 日志记录

- Cluster Operator 日志
- 主题 Operator 日志记录
- 用户 Operator 日志记录

9.13.2. 为日志创建 ConfigMap

要使用 ConfigMap 定义日志记录属性，您可以创建 ConfigMap，然后将其引用为资源 spec 中的日志记录定义的一部分。

ConfigMap 必须包含适当的日志记录配置。

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

配置必须放在这些属性下。

在此流程中，ConfigMap 为 Kafka 资源定义根日志记录器。

流程

1. 创建 ConfigMap。

   您可以将 ConfigMap 创建为 YAML 文件或从属性文件创建。

   带有 Kafka 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，并将 logging.valueFrom.configMapKeyRef.key 设置为此 ConfigMap 中的键。

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

3. 创建或更新资源。

```
oc apply -f <kafka_configuration_file>
```

9.13.3. 配置 Cluster Operator 日志

Cluster Operator 日志记录通过名为 strimzi-cluster-operator 的 ConfigMap 配置。安装 Cluster Operator 时会创建一个包含日志记录配置的 ConfigMap。这个 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.log4j2.properties 值来配置 Cluster Operator 日志记录。

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
```

使用这个 ConfigMap，您可以控制日志记录的各个方面，包括根日志记录器级别、日志输出格式和不同组件的日志级别。monitorInterval 设置决定了日志记录配置重新加载的频率。您还可以控制 Kafka AdminClient、ZooKTrustManager、Netty 和 OkHttp 客户端的日志级别。Netnetty 是 Apache Kafka 用于网络通信的框架，而 OkHttp 是用于发出 HTTP 请求的库。

如果部署 Cluster Operator 时缺少 ConfigMap，则使用默认的日志记录值。

如果在部署 Cluster Operator 后 ConfigMap 意外删除，则会使用最近载入的日志配置。创建新的 ConfigMap 以加载新的日志记录配置。

注意

不要从 ConfigMap 中删除 monitorInterval 选项。

9.13.4. 在 Apache Kafka operator 的 Streams 中添加日志记录过滤器

如果您使用 ConfigMap 为 Apache Kafka operator 配置流(log4j2)日志记录级别，您还可以定义日志记录过滤器来限制日志中返回的内容。

当您有大量日志信息时，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为日志记录器返回的日志数量，以便您可以专注于特定资源。当设置了过滤器时，只会记录与过滤器匹配的日志消息。

过滤器使用 *标记来指定* 日志中要包含的内容。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，并使用故障集群的命名空间和名称。

本例显示了一个名为 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 类型比较指定的过滤标记。

2

如果标记匹配，onMatch 属性接受日志。

3

如果标记不匹配，onMismatch 属性会拒绝日志。

4

用于过滤的标记的格式是 *KIND (NAMESPACE/NAME-OF-RESOURCE)*。

您可以创建一个或多个过滤器。在这里，为两个 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)
```

在 Cluster Operator 中添加过滤器

要将过滤器添加到 Cluster Operator 中，请更新其日志记录 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，过滤器属性只返回 my-kafka-cluster Kafka 集群的日志：

```
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
```

在主题 Operator 或 User Operator 中添加过滤器

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

在此过程中，使用 Topic Operator 的过滤器创建日志记录 ConfigMap。用户 Operator 使用相同的方法。

流程

1. 创建 ConfigMap。

您可以将 ConfigMap 创建为 YAML 文件或从属性文件创建。

在本例中，过滤器属性只返回 my-topic 主题的日志：

```
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，并将 logging.valueFrom.configMapKeyRef.key 设置为此 ConfigMap 中的键。

   对于主题 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
```

其他资源

- 配置 Kafka
- Cluster Operator 日志
- 主题 Operator 日志记录
- 用户 Operator 日志记录

9.14. 使用 CONFIGMAP 添加配置

使用 ConfigMap 资源，将特定的配置添加到 Apache Kafka 部署的 Streams 中。ConfigMap 使用键值对来存储非机密数据。添加到 ConfigMap 的配置数据在一个位置维护，并可在组件间重复使用。

ConfigMap 只能存储以下类型的配置数据：

- 日志记录配置
- 指标配置
- Kafka 连接连接器的外部配置

您不能将 ConfigMap 用于其他配置区域。

在配置组件时，您可以使用 configMapKeyRef 属性添加对 ConfigMap 的引用。

例如，您可以使用 configMapKeyRef 引用为日志记录提供配置的 ConfigMap。您可以使用 ConfigMap 传递一个 Log4j 配置文件。您可以添加对 日志记录配置 的引用。

日志的 ConfigMap 示例

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

要将 ConfigMap 用于指标配置，您可以以同样的方式添加对组件的 metricsConfig 配置的引用。

externalConfiguration 属性从挂载到 pod 的 ConfigMap （或 Secret）中的数据作为环境变量或卷提供。您可以将外部配置数据用于 Kafka Connect 使用的连接器。数据可能与外部数据源相关，提供连接器与该数据源通信所需的值。

例如，您可以使用 configMapKeyRef 属性将 ConfigMap 中的配置数据作为环境变量传递。

提供环境变量值的 ConfigMap 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
```

如果您使用外部管理的 ConfigMap，请使用配置供应商加载 ConfigMap 中的数据。

9.14.1. 命名自定义 ConfigMap

Apache Kafka 的流会在部署到 OpenShift 时创建自己的 ConfigMap 和其他资源。ConfigMap 包含运行组件所需的数据。不可编辑为 Apache Kafka 的 Streams 创建的 ConfigMap。

确保您创建的任何自定义 ConfigMap 的名称都与这些默认 ConfigMap 的名称相同。如果它们具有相同的名称，则它们将被覆盖。例如，如果您的 ConfigMap 与 Kafka 集群的 ConfigMap 的名称相同，则在对 Kafka 集群有更新时会覆盖它。

其他资源

- Kafka 集群资源列表 （包括 ConfigMap）
- 日志记录配置
- metricsConfig
- ExternalConfiguration 模式参考
- 从外部来源加载配置值

9.15. 从外部来源加载配置值

使用配置提供程序从外部来源加载配置数据。供应商独立于 Apache Kafka 的 Streams 操作。您可以使用它们为所有 Kafka 组件加载配置数据，包括生成者和消费者。您可以在组件的配置中引用外部源并提供访问权限。供应商在不重启 Kafka 组件或提取文件的情况下加载数据，即使引用新的外部源也是如此。例如，使用供应商提供 Kafka Connect 连接器配置的凭证。配置必须包含对外部源的任何访问权限。

9.15.1. 启用配置供应商

您可以使用组件的 spec 配置中的 config.providers 属性启用一个或多个配置供应商。

启用配置供应商的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  # ...
  config:
    # ...
    config.providers: env
    config.providers.env.class:
org.apache.kafka.common.config.provider.EnvVarConfigProvider
  # ...
```

KubernetesSecretConfigProvider

从 OpenShift 机密加载配置数据。您可以在存储配置数据的 secret 中指定 secret 和密钥。此提供程序可用于存储敏感配置数据，如密码或其他用户凭据。

KubernetesConfigMapConfigProvider

从 OpenShift 配置映射加载配置数据。您可以在存储配置数据的配置映射中指定配置映射的名称和键。此提供程序可用于存储非敏感配置数据。

EnvVarConfigProvider

从环境变量加载配置数据。您可以指定保存配置数据的环境变量的名称。此提供程序可用于配置容器中运行的应用，例如从机密映射的环境变量加载证书或 JAAS 配置。

FileConfigProvider

从文件加载配置数据。您可以指定保存配置数据的文件的路径。此提供程序可用于从挂载到容器中的文件中加载配置数据。

DirectoryConfigProvider

从目录中的文件加载配置数据。您可以指定存储配置文件的目录的路径。此提供程序可用于加载多个配置文件和将配置数据组织到单独的文件中。

要使用作为 OpenShift Configuration Provider 插件一部分的 KubernetesSecretConfigProvider 和 KubernetesConfigMapConfigProvider，您必须为包含配置文件的命名空间设置访问权限。

您可以在不设置访问权限的情况下使用其他供应商。您可以通过执行以下操作来为 Kafka Connect 或 MirrorMaker 2 提供连接器配置：

- 将配置映射或 secret 挂载到 Kafka Connect pod 中作为环境变量或卷中
- 在 Kafka Connect 或 MirrorMaker 2 配置中启用 EnvVarConfigProvider,FileConfigProvider, 或 DirectoryConfigProvider
- 使用 KafkaConnect 或 KafkaMirrorMaker2 资源的 spec 中的 externalConfiguration 属性传递连接器配置

使用供应商有助于防止通过 Kafka Connect REST 接口传递受限信息。在以下情况下可以使用这个方法：

- 使用连接器用来连接和数据源通信的值挂载环境变量
- 使用配置 Kafka Connect 连接器的值挂载属性文件
- 在目录中挂载文件，其中包含连接器使用的 TLS 信任存储和密钥存储的值

注意

将新的 Secret 或 ConfigMap 用于连接器时需要重启，这可能会破坏其他连接器。

其他资源

ExternalConfiguration 模式参考

9.15.2. 从 secret 或配置映射加载配置值

使用 KubernetesSecretConfigProvider 从 secret 或 KubernetesConfigMapConfigProvider 提供配置属性，从配置映射提供配置属性。

在此过程中，配置映射为连接器提供配置属性。属性指定为配置映射的键值。配置映射作为卷挂载到 Kafka Connect pod 中。

先决条件

- Kafka 集群正在运行。
- Cluster Operator 正在运行。
- 您有一个包含连接器配置的配置映射。

使用连接器属性的配置映射示例

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

流程

1. 配置 KafkaConnect 资源。
   - 启用 KubernetesConfigMapConfigProvider

   此处显示的规格支持从配置映射和 secret 加载值。

   使用配置映射和 secret 的 Kafka 连接配置示例

```
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.configmaps.class:
io.strimzi.kafka.KubernetesConfigMapConfigProvider (2)
  config.providers.secrets.class: io.strimzi.kafka.KubernetesSecretConfigProvider (3)
# ...
```

(1) 配置提供程序的别名用于定义其他配置参数。provider 参数使用来自 config.providers 的别名，格式为 config.providers.${alias}.class。

(2) KubernetesConfigMapConfigProvider 提供配置映射的值。

(3) KubernetesSecretConfigProvider 提供来自 secret 的值。

2. 创建或更新资源以启用提供程序。

```
oc apply -f <kafka_connect_configuration_file>
```

3. 创建一个允许访问外部配置映射中值的角色。

要从配置映射中访问值的角色示例

```
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 配置映射。

4. 创建一个角色绑定，允许访问包含配置映射的命名空间。

访问包含配置映射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
# ...
```

角色绑定授予角色访问 my-project 命名空间的权限。

服务帐户必须是 Kafka Connect 部署使用的相同。服务帐户名称是 < cluster_name>-connect，其中 < cluster_name > 是 KafkaConnect 自定义资源的名称。

5. 在连接器配置中引用配置映射。

引用配置映射的连接器配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-connector
```

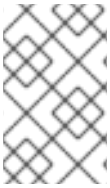
```
  labels:
    strimzi.io/cluster: my-connect
spec:
  # ...
  config:
    option: ${configmaps:my-project/my-connector-configuration:option1}
    # ...
# ...
```

占位符结构是 configmaps:<path_and_file_name>:<property>。KubernetesConfigMapConfigProvider 从外部配置映射读取和提取 option1 属性值。

9.15.3. 从环境变量加载配置值

使用 EnvVarConfigProvider 提供配置属性作为环境变量。环境变量可以包含来自配置映射或 secret 的值。

在此过程中，环境变量为连接器提供配置属性，以便与 Amazon AWS 通信。连接器必须能够读取 AWS_ACCESS_KEY_ID 和 AWS_SECRET_ACCESS_KEY。环境变量的值派生自挂载到 Kafka Connect pod 的 secret。



注意

用户定义的环境变量的名称不能以 KAFKA_ 或 STRIMZI_ 开头。

先决条件

- Kafka 集群正在运行。
- Cluster Operator 正在运行。
- 您有一个包含连接器配置的 secret。

带有环境变量值的 secret 示例

```
apiVersion: v1
kind: Secret
metadata:
  name: aws-creds
type: Opaque
data:
  awsAccessKey: QUtJQVhYWFhYWFhYWFhYWFg=
  awsSecretAccessKey: Ylhsd1lYTnpkMjl5WkE=
```

流程

1. 配置 KafkaConnect 资源。

   - 启用 EnvVarConfigProvider
   - 使用 externalConfiguration 属性指定环境变量。

使用外部环境变量的 Kafka 连接配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  # ...
  config:
    # ...
    config.providers: env (1)
    config.providers.env.class:
org.apache.kafka.common.config.provider.EnvVarConfigProvider (2)
  # ...
  externalConfiguration:
    env:
      - name: AWS_ACCESS_KEY_ID (3)
        valueFrom:
          secretKeyRef:
            name: aws-creds (4)
            key: awsAccessKey (5)
      - name: AWS_SECRET_ACCESS_KEY
        valueFrom:
```

```
      secretKeyRef:
        name: aws-creds
        key: awsSecretAccessKey
# ...
```

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

2 EnvVarConfigProvider 提供来自环境变量的值。

3 环境变量从 secret 中获取一个值。

4 包含环境变量的 secret 的名称。

5 secret 中存储的密钥名称。

注意

secretKeyRef 属性引用 secret 中的键。如果您使用配置映射而不是 secret，请使用 configMapKeyRef 属性。

2. 创建或更新资源以启用提供程序。

```
oc apply -f <kafka_connect_configuration_file>
```

3. 在连接器配置中引用环境变量。

引用环境变量的连接器配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-connector
  labels:
    strimzi.io/cluster: my-connect
spec:
  # ...
  config:
    option: ${env:AWS_ACCESS_KEY_ID}
    option: ${env:AWS_SECRET_ACCESS_KEY}
    # ...
# ...
```

占位符结构是 env:<environment_variable_name>。EnvVarConfigProvider 从挂载的 secret 中读取并提取环境变量值。

9.15.4. 从目录中的文件载入配置值

使用 FileConfigProvider 从目录中的文件提供配置属性。文件可以是配置映射或 secret。

在此过程中，文件为连接器提供配置属性。数据库名称和密码被指定为 secret 的属性。secret 作为一个卷挂载到 Kafka Connect pod。卷挂载到路径 /opt/kafka/external-configuration/<volume-name > 上。

先决条件

- Kafka 集群正在运行。
- Cluster Operator 正在运行。
- 您有一个包含连接器配置的 secret。

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

连接器以属性文件格式配置。

2

配置中使用的数据库用户名和密码属性。

流程

1. 配置 KafkaConnect 资源。

   - 启用 FileConfigProvider
   - 使用 externalConfiguration 属性指定文件。

使用外部属性文件的 Kafka 连接配置示例

```
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 提供来自属性文件的值。参数使用 config.providers 的别名，格式为 config.providers.${alias}.class。

3 包含 secret 的卷名称。

4 secret 的名称。

2. 创建或更新资源以启用提供程序。

```
oc apply -f <kafka_connect_configuration_file>
```

3. 将连接器配置中的文件属性引用为占位符。

引用该文件的连接器配置示例

```
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"
    #...
```

占位符结构是 file:<path_and_file_name>:<property>。FileConfigProvider 从挂载的 secret 读取并提取数据库 username 和 password 属性值。

9.15.5. 从目录中的多个文件载入配置值

使用 DirectoryConfigProvider 从目录中的多个文件提供配置属性。文件可以是配置映射或 secret。

在此过程中，secret 为连接器提供 TLS 密钥存储和信任存储用户凭证。凭据位于单独的文件中。secret 作为卷挂载到 Kafka Connect pod 中。卷挂载到路径 /opt/kafka/external-configuration/<volume-name > 上。

先决条件

- Kafka 集群正在运行。
- Cluster Operator 正在运行。
- 您有一个包含用户凭证的 secret。

使用用户凭证的 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> # Public key of the clients CA
  user.crt: <user_certificate> # Public key of the user
  user.key: <user_private_key> # Private key of the user
  user.p12: <store> # PKCS #12 store for user certificates and keys
  user.password: <password_for_store> # Protects the PKCS #12 store
```

my-user secret 为连接器提供密钥存储凭证(user.crt 和 user.key)。

在部署 Kafka 集群时生成的 <cluster_name>-cluster-ca-cert secret 将集群 CA 证书提供为信任存储凭证(ca.crt)。

流程

1. 配置 KafkaConnect 资源。

   - 启用 DirectoryConfigProvider
   - 使用 externalConfiguration 属性指定文件。

使用外部属性文件的 Kafka 连接配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    config.providers: directory 1
    config.providers.directory.class:
org.apache.kafka.common.config.provider.DirectoryConfigProvider 2
  #...
  externalConfiguration:
    volumes: 3
      - name: cluster-ca 4
        secret:
```

```
      secretName: my-cluster-cluster-ca-cert 5
  - name: my-user
    secret:
      secretName: my-user 6
```

1 配置提供程序的别名用于定义其他配置参数。

2 DirectoryConfigProvider 从目录中的文件提供值。参数使用 config.providers 的别名，格式为 config.providers.${alias}.class。

3 包含 secret 的卷名称。

4 集群 CA 证书的 secret 名称，以提供信任存储配置。

5 提供密钥存储配置的用户的机密名称。

2. 创建或更新资源以启用提供程序。

```
oc apply -f <kafka_connect_configuration_file>
```

3. 将连接器配置中的文件属性引用为占位符。

引用文件的连接器配置示例

```
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.history.producer.security.protocol: SSL
    database.history.producer.ssl.truststore.type: PEM
    database.history.producer.ssl.truststore.certificates:
"${directory:/opt/kafka/external-configuration/cluster-ca:ca.crt}"
    database.history.producer.ssl.keystore.type: PEM
    database.history.producer.ssl.keystore.certificate.chain:
"${directory:/opt/kafka/external-configuration/my-user:user.crt}"
    database.history.producer.ssl.keystore.key: "${directory:/opt/kafka/external-
configuration/my-user:user.key}"
    #...
```

占位符结构是 directory:<path>:<file_name>。DirectoryConfigProvider 从挂载的 secret 读取并提取凭证。

9.16. 自定义 OPENSHIFT 资源

Apache Kafka 部署的流会创建 OpenShift 资源，如 Deployment、Pod 和 Service 资源。这些资源由 Apache Kafka operator 的 Streams 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操作器管理的 OpenShift 资源，Operator 将还原您的更改。

如果要执行某些任务，更改 Operator 管理的 OpenShift 资源会很有用，例如：

- 添加自定义标签或注解，以控制 Istio 或其他服务如何处理 Pod
- 管理如何为集群创建 Loadbalancer-type 服务

要更改 OpenShift 资源，您可以使用 Apache Kafka 自定义资源的不同流的 spec 部分中的 template 属性。

以下是您可以应用更改的自定义资源列表：

- Kafka.spec.kafka
- Kafka.spec.zookeeper
- Kafka.spec.entityOperator
- Kafka.spec.kafkaExporter
- Kafka.spec.cruiseControl
- KafkaNodePool.spec
- KafkaConnect.spec
- KafkaMirrorMaker.spec
- KafkaMirrorMaker2.spec
- KafkaBridge.spec
- KafkaUser.spec

有关这些属性的更多信息，请参阅 Apache Kafka 自定义资源 API 参考流。

Apache Kafka 自定义资源 API 参考流提供了有关可自定义字段的更多详情。

在以下示例中，template 属性用于修改 Kafka 代理的 pod 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:
      pod:
        metadata:
          labels:
            mylabel: myvalue
    # ...
```

9.16.1. 自定义镜像拉取策略

通过 Apache Kafka 的流，您可以自定义 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 环境变量可以设置为三个不同的值：

Always

每次 pod 启动或重启时，容器镜像都会从 registry 中拉取。

IfNotPresent

只有在容器镜像之前没有拉取时才从 registry 中拉取容器镜像。

Never

容器镜像从 registry 中拉取。

目前，镜像拉取策略一次只能为所有 Kafka、Kafka Connect 和 Kafka MirrorMaker 集群自定义。更改策略将导致对所有 Kafka、Kafka Connect 和 Kafka MirrorMaker 集群进行滚动更新。

其他资源

- 中断.

9.16.2. 应用终止宽限期

应用终止宽限期，为 Kafka 集群提供足够时间来完全关闭。

使用 terminationGracePeriodSeconds 属性指定时间。将属性添加到 Kafka 自定义资源的 template.pod 配置中。

您添加的时间将取决于 Kafka 集群的大小。终止宽限期的 OpenShift 默认为 30 秒。如果您发现集群没有完全关闭，您可以提高终止宽限期。

每次 pod 重启时都会应用终止宽限期。当 OpenShift 发送一个 *术语* (termination)信号到容器集中运行的进程时，周期开始。周期应反映在终止 pod 停止前将终止 pod 的进程传输到另一个 pod 所需的时间。在周期结束后，*kill* 信号将停止 pod 中仍在运行的任何进程。

以下示例为 Kafka 自定义资源添加 120 秒的终止宽限期。您还可以在其他 Kafka 组件的自定义资源中指定配置。

终止宽限期配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    template:
      pod:
        terminationGracePeriodSeconds: 120
        # ...
    # ...
```

第 10 章 使用主题 OPERATOR 管理 KAFKA 主题

KafkaTopic 资源配置主题，包括分区和复制因素设置。当使用 KafkaTopic 创建、修改或删除主题时，主题 Operator 可确保这些更改反映在 Kafka 集群中。

如需有关 KafkaTopic 资源的更多信息，请参阅 KafkaTopic 模式参考。

10.1. 主题管理模式

KafkaTopic 资源负责管理 Kafka 集群中的单个主题。Topic Operator 提供了两种管理 KafkaTopic 资源和 Kafka 主题的模式：

单向模式（默认）

单向模式不需要 ZooKeeper 进行集群管理。它以 KRaft 模式与使用 Apache Kafka 的 Streams 兼容。

双向模式

双向模式需要 ZooKeeper 用于集群管理。在 KRaft 模式中，它与使用 Apache Kafka 的 Streams 不兼容。

注意

当功能门使主题 Operator 在单向模式下运行时，双向模式将分阶段化。这个转换旨在增强用户体验，特别是在 KRaft 模式中支持 Kafka。

10.1.1. 单向主题管理

在单向模式中，主题 Operator 运行如下：

- 当创建、删除或更改 KafkaTopic 时，主题 Operator 会在 Kafka 主题上执行对应的操作。

如果在 Kafka 集群中直接创建、删除或修改一个主题，且没有对应的 KafkaTopic 资源，则 Topic Operator 不会管理该主题。Topic Operator 只管理与 KafkaTopic 资源关联的 Kafka 主题，不会影响在 Kafka 集群中独立管理的主题。如果 Kafka 主题存在 KafkaTopic，则会恢复资源外所做的任何配置更改。

主题 Operator 可以检测多个 KafkaTopic 资源试图使用相同的 .spec.topicName 管理 Kafka 主题的情况。只有最旧的资源会被协调，而其他资源会失败并显示资源冲突错误。

10.1.2. 双向主题管理

在双向模式中，主题 Operator 运行如下：

- 当创建、删除或更改 KafkaTopic 时，主题 Operator 会在 Kafka 主题上执行对应的操作。
- 同样，当在 Kafka 集群中创建、删除或更改主题时，主题 Operator 会在 KafkaTopic 资源上执行对应的操作。

提示

尝试坚持管理主题的方法，可以通过 KafkaTopic 资源或在 Kafka 中直接进行。避免在给定主题的两个方法之间定期切换。

10.2. 主题命名约定

KafkaTopic 资源包括主题的名称，以及一个标识它所属的 Kafka 集群名称的标签。

标识 Kafka 集群用于主题处理的标签

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

该标签提供 Kafka 资源的集群名称。主题 Operator 使用该标签作为确定要管理的 KafkaTopic 资源的机制。如果标签与 Kafka 集群不匹配，则 Topic Operator 无法看到 KafkaTopic，且不会创建主题。

Kafka 和 OpenShift 都有自己的命名验证规则，Kafka 主题名称在 OpenShift 中可能不是有效的资源名称。如果可能，请尝试并坚持适合两者的命名约定。

请考虑以下指南：

- 使用反映主题性质的主题名称
- 很简洁，并在 63 个字符下保留名称
- 使用所有小写和连字符
- 避免特殊字符、空格或符号

KafkaTopic 资源允许您使用 metadata.name 字段指定 Kafka 主题名称。但是，如果所需的 Kafka 主题名称不是有效的 OpenShift 资源名称，您可以使用 spec.topicName 属性来指定实际名称。spec.topicName 字段是可选的，当不存在时，Kafka 主题名称默认为主题的 metadata.name。创建主题时，无法稍后更改主题名称。

提供有效 Kafka 主题名称的示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: my-topic-1 1
spec:
  topicName: My.Topic.1 2
  # ...
```

1 在 OpenShift 中工作的有效主题名称。

2

如果多个 KafkaTopic 资源引用同一 Kafka 主题，则首先创建的资源被视为管理该主题的资源。更新较新的资源的状态以指示冲突，并且它们的 Ready 状态更改为 False。

如果 Kafka 客户端应用程序（如 Kafka Streams）自动创建带有无效 OpenShift 资源名称的主题，则主题 Operator 会在双向模式中使用时生成一个有效的 metadata.name。它替换无效字符，并将哈希附加到名称中。但是，此行为不适用于单向模式。

替换无效主题名称的示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: my-topic---c55e57fe2546a33f9e603caf57165db4072e827e
  # ...
```

注意

如需有关集群中标识符和名称要求的更多信息，请参阅 OpenShift 文档 对象名称和 ID。

10.3. 处理对主题的更改

主题 Operator 如何处理对主题的更改取决于 主题管理的模式。

- 对于单向主题管理，配置更改仅以一个方向：从 KafkaTopic 资源到 Kafka 主题。对 KafkaTopic 资源外管理的 Kafka 主题的任何更改都会被恢复。
- 对于双向主题管理，配置更改会在 Kafka 主题和 KafkaTopic 资源之间同步。不兼容的更改会优先选择 Kafka 配置，并相应地调整 KafkaTopic 资源。

10.3.1. 双向主题管理的主题存储

对于双向主题管理，当没有单一数据源时，主题 Operator 能够处理对主题的更改。KafkaTopic 资源和 Kafka 主题可以独立修改，其中实时观察更改可能并不总是可行，特别是在 Topic Operator 无法正常工作时。为了解决这个问题，主题 Operator 维护一个主题存储，用于存储有关每个主题的主题配置信息。它将 Kafka 集群和 OpenShift 的状态与主题存储进行比较，以确定同步所需的更改。此评估在启动期间和定期的间隔发生，而主题 Operator 处于活跃状态时。

例如，如果主题 Operator 不活跃，并且在重启后会创建一个名为 *my-topic* 的新 KafkaTopic，则主题 Operator 会识别主题存储中没有 *my-topic*。它识别在最后一次操作后创建的 KafkaTopic。因此，主题 Operator 生成对应的 Kafka 主题，并将元数据保存在主题存储中。

主题存储可让 Topic Operator 管理在 Kafka 主题和 KafkaTopic 资源中更改主题配置的情况，只要更改兼容。当对 KafkaTopic 自定义资源更新或更改 Kafka 主题配置时，主题存储会在与 Kafka 集群协调后更新，只要更改兼容。

主题存储 基于 Kafka Streams 键-值机制，它使用 Kafka 主题来持久保留状态。主题元数据缓存在内存中，并在主题 Operator 本地访问。应用到本地内存缓存的操作更新会被保留到磁盘上的备份主题存储中。主题存储会与 Kafka 主题或 OpenShift KafkaTopic 自定义资源的更新同步。通过以这种方式设置主题存储快速处理操作，但内存中缓存崩溃，它会自动从持久性存储中重新填充。

内部主题支持处理主题存储中的主题元数据。

__strimzi_store_topic

用于存储主题元数据的输入主题

__strimzi-topic-operator-kstreams-topic-store-changelog

保留紧凑主题存储值的日志

警告

不要删除这些主题，因为它们对于 Topic Operator 的运行至关重要。

10.3.2. 将主题元数据从 ZooKeeper 迁移到主题存储

在以前的 Apache Kafka Streams 版本中，主题元数据存储在 ZooKeeper 中。主题存储会删除这个要求，将元数据带到 Kafka 集群，并受 Topic Operator 控制。

当升级到 Apache Kafka 2.7 的 Streams 时，对主题存储的转换到 Topic Operator 控制是无缝的。元数据从 ZooKeeper 找到并迁移，旧存储会被删除。

10.3.3. 降级到使用 ZooKeeper 存储主题元数据的 Apache Kafka 版本的 Streams

如果您要恢复到 1.7 之前的 Apache Kafka 的 Streams 版本，其使用 ZooKeeper 进行主题元数据存储，您仍将 Cluster Operator 降级到之前的版本，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，还必须使用 kafka-topics 命令删除为主题存储创建的主题，指定 Kafka 集群的 bootstrap 地址。例如：

```
oc run kafka-admin -ti --image=registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0 --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
```

该命令必须与监听程序的类型以及用于访问 Kafka 集群的身份验证对应。

主题 Operator 将从 Kafka 中的主题状态重建 ZooKeeper 主题元数据。

10.3.4. 自动创建主题

应用程序可以在 Kafka 集群中触发自动创建主题。默认情况下，Kafka 代理配置 auto.create.topics.enable 被设置为 true，允许代理在应用程序尝试从不存在的主题生成或消耗时自动创建主题。应用程序也可能使用 Kafka AdminClient 自动创建主题。当应用程序与 KafkaTopic 资源一起部署时，在主题 Operator 可以响应 KafkaTopic 前，集群中的自动主题创建可能会发生。

对于双向主题管理，主题 Operator 会同步主题和 KafkaTopic 资源之间的更改。

如果您使用单向主题管理，这可能意味着最初为应用部署创建的主题最初使用默认主题配置创建。如果主题 Operator 会尝试根据应用程序部署中包含的 KafkaTopic 资源规格重新配置主题，则操作可能会失败，因为不允许对配置进行所需的更改。例如，如果更改意味着减少主题分区的数量。因此，在使用单向主题管理时，建议在 Kafka 集群配置中禁用 auto.create.topics.enable。

10.4. 配置 KAFKA 主题

使用 KafkaTopic 资源的属性来配置 Kafka 主题。对 KafkaTopic 中主题配置所做的更改会被传播到 Kafka。

您可以使用 oc apply 创建或修改主题，并使用 oc delete 删除现有主题。

例如：

- oc apply -f <topic_config_file>
- oc delete KafkaTopic <topic_name>

要能够删除主题，必须在 Kafka 资源的 spec.kafka.config 中将 delete.topic.enable 设置为 true （默认）。

此流程演示了如何创建带有 10 个分区和 2 个副本的主题。

注意

对于主题管理的单向和双向模式，这个流程是相同的。

开始前

KafkaTopic 资源不允许以下更改：

- 重命名 spec.topicName 中定义的主题。将检测到 spec.topicName 和 status.topicName 不匹配。
- 使用 spec.partitions (Kafka 不支持)减少分区数量。
- 修改 spec.replicas 中指定的副本数量。

警告

增加带有键的主题的 spec.partitions 将更改记录分区，这可能会导致问题，特别是在主题使用语义分区时。

先决条件

- 正在运行的 Kafka 集群使用 mTLS 身份验证和 TLS 加密配置 Kafka 代理监听程序。
- 正在运行的主题 Operator （通常使用 Entity Operator 部署）。
- 要删除主题，在 Kafka 资源的 spec.kafka.config 中的 delete.topic.enable=true （默认）。

流程

1. 配置 KafkaTopic 资源。

Kafka 主题配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: my-topic-1
  labels:
    strimzi.io/cluster: my-cluster
spec:
  partitions: 10
  replicas: 2
```

提示

修改主题时，您可以使用 oc get kafkatopic my-topic-1 -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>
```

3. 等待主题的 ready 状态更改为 True ：

```
oc get kafkatopics -o wide -w -n <namespace>
```

Kafka 主题状态

```
NAME       CLUSTER    PARTITIONS  REPLICATION FACTOR READY
my-topic-1   my-cluster  10        3                 True
my-topic-2   my-cluster  10        3
my-topic-3   my-cluster  10        3                 True
```

当 READY 输出显示为 True 时，主题创建成功。

4. 如果 READY 列留空，请从资源 YAML 或 Topic Operator 日志中获取有关状态的更多详细信息。

状态信息提供有关当前状态原因的详细信息。

```
oc get kafkatopics my-topic-2 -o yaml
```

有关具有 NotReady 状态的主题详情

```
# ...
status:
  conditions:
```

```
  - lastTransitionTime: "2022-06-13T10:14:43.351550Z"
    message: Number of partitions cannot be decreased
    reason: PartitionDecreaseException
    status: "True"
    type: NotReady
```

在本例中，主题未就绪的原因是，原始分区数量在 KafkaTopic 配置中被减少。Kafka 不支持此功能。

重置主题配置后，状态会显示主题为 ready。

```
oc get kafkatopics my-topic-2 -o wide -w -n <namespace>
```

主题的状态更新

```
NAME        CLUSTER     PARTITIONS  REPLICATION FACTOR READY
my-topic-2  my-cluster  10          3                  True
```

获取详情不会显示任何信息

```
oc get kafkatopics my-topic-2 -o yaml
```

带有 READY 状态的主题详情

```
# ...
status:
  conditions:
  - lastTransitionTime: '2022-06-13T10:15:03.761084Z'
    status: 'True'
    type: Ready
```

10.5. 为复制和分区数量配置主题

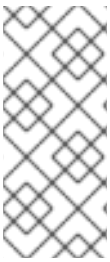
主题 Operator 管理的主题的推荐配置是 3 个主题复制因素，最小 2 个 in-sync 副本。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: my-topic
  labels:
    strimzi.io/cluster: my-cluster
spec:
  partitions: 10 (1)
  replicas: 3 (2)
  config:
    min.insync.replicas: 2 (3)
  #...
```

(1) 主题的分区数。

(2) 副本主题分区的数量。目前，无法在 KafkaTopic 资源中更改，但可以使用 kafka-reassign-partitions.sh 工具进行更改。

(3) 必须成功写入消息或引发异常的最小副本分区数量。



注意

in-sync 副本与生成者应用程序的 acks 配置结合使用。acks 配置决定了在确认消息被确认为成功收到的消息之前，必须复制消息的后续分区数量。双向主题 Operator 使用 acks=all 运行，用于其内部主题，其中消息必须被所有同步的副本确认。

当通过添加或删除代理扩展 Kafka 集群时，复制因素配置不会改变，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，并手动将副本重新分配给代理。

另外，虽然 Apache Kafka 的 Cruise Control for Streams 集成无法更改主题的复制因素，但它为重新平衡 Kafka 生成的优化提议包括传输分区副本和更改分区领导的命令。

其他资源

- 降级 Apache Kafka 的流
- 第 20.1 节 “分区重新分配工具概述”
- 第 19 章 *使用 Cruise Control 重新平衡集群*

10.6. 管理 KAFKATOPIC 资源，而不影响 KAFKA 主题

此流程描述了如何当前通过 KafkaTopic 资源管理的 Kafka 主题转换为非管理的主题。此功能在各种情况下非常有用。例如，您可能想要更新 KafkaTopic 资源的 metadata.name。您只能通过删除原始 KafkaTopic 资源并重新创建新资源来完成此操作。

通过使用 strimzi.io/managed=false 注解 KafkaTopic 资源，这表示 Topic Operator 应该不再管理该特定主题。这可让您在更改资源的配置或其他管理任务时保留 Kafka 主题。

如果您使用无方向性的主题管理，您可以执行此任务。

先决条件

- 必须部署 Cluster Operator。

流程

1. 在 OpenShift 中注解 KafkaTopic 资源，将 strimzi.io/managed 设置为 false ：

```
oc annotate kafkatopic my-topic-1 strimzi.io/managed="false"
```

在 KafkaTopic 资源中指定主题的 metadata.name，本例中为 my-topic-1。

2. 检查 KafkaTopic 资源的状态，以确保请求成功：

```
oc get kafkatopics my-topic-1 -o yaml
```

具有 Ready 状态的主题示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  generation: 124
  name: my-topic-1
  finalizer:
    strimzi.io/topic-operator
  labels:
    strimzi.io/cluster: my-cluster
spec:
  partitions: 10
  replicas: 2

# ...
status:
  observedGeneration: 124 1
  topicName: my-topic-1
  conditions:
  - type: Ready
    status: True
    lastTransitionTime: 20230301T103000Z
```

1

资源成功协调意味着主题不再管理。

metadata.generation （部署的当前版本）的 值必须与 status.observedGeneration （资源的最新协调）匹配。

3. 现在，您可以在不影响管理的 Kafka 主题的情况下更改 KafkaTopic 资源。

例如，要更改 metadata.name，请执行以下操作：

a. 删除原始 KafkTopic 资源：

```
oc delete kafkatopic <kafka_topic_name>
```

b.

重新创建具有不同 metadata.name 的 KafkTopic 资源，但使用 spec.topicName 来引用由原始管理的相同主题

4. 如果您没有删除原始 KafkaTopic 资源，并且您希望再次恢复管理 Kafka 主题，请将 strimzi.io/managed 注解设置为 true 或删除注解。

10.7. 为现有 KAFKA 主题启用主题管理

此流程描述了如何为当前没有通过 KafkaTopic 资源管理的主题启用主题管理。您可以通过创建匹配的 KafkaTopic 资源来实现此目的。

如果您使用无方向性的主题管理，您可以执行此任务。

先决条件

- 必须部署 Cluster Operator。

流程

1. 使用与 Kafka 主题相同的 metadata.name 创建 KafkaTopic 资源。

   或者，如果 Kafka 中的主题名称不是法律 OpenShift 资源名称，则使用 spec.topicName。

   Kafka 主题配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: my-topic-1
  labels:
    strimzi.io/cluster: my-cluster
spec:
  partitions: 10
  replicas: 2
```

在本例中，Kafka 主题名为 my-topic-1。

主题 Operator 检查主题是否由另一个 KafkaTopic 资源管理。如果是，旧的资源具有优先权，并在新资源状态中返回资源冲突错误。

2. 应用 KafkaTopic 资源：

```
oc apply -f <topic_configuration_file>
```

3. 等待 Operator 更新 Kafka 中的主题。

Operator 使用具有相同名称的 KafkaTopic 的 spec 更新 Kafka 主题。

4. 检查 KafkaTopic 资源的状态，以确保请求成功：

```
oc get kafkatopics my-topic-1 -o yaml
```

具有 Ready 状态的主题示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  generation: 1
  name: my-topic-1
  labels:
    strimzi.io/cluster: my-cluster
spec:
  partitions: 10
  replicas: 2
# ...
status:
  observedGeneration: 1 (1)
  topicName: my-topic-1
  conditions:
  - type: Ready
    status: True
    lastTransitionTime: 20230301T103000Z
```

1

对资源成功协调意味着主题现已被管理。

metadata.generation （部署的当前版本）的 值必须与 status.observedGeneration （资源的最新协调）匹配。

10.8. 删除受管主题

单向主题管理支持通过 KafkaTopic 资源删除通过 OpenShift 终结器管理的主题。这由 STRIMZI_USE_FINALIZERS Topic Operator 环境变量决定。默认情况下，这被设置为 true，但如果您不想 主题 Operator 来添加终结器，则可以在主题 Operator env 配置中将其设置为 false。

终结器(finalizers)可确保按顺序控制地删除 KafkaTopic 资源。Topic Operator 的终结器添加到 KafkaTopic 资源的元数据中：

控制删除主题的终结器

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  generation: 1
  name: my-topic-1
  finalizers:
    - strimzi.io/topic-operator
  labels:
    strimzi.io/cluster: my-cluster
```

在本例中，为主题 my-topic-1 添加了终结器。finalizer 会阻止主题被完全删除，直到最终化过程完成为止。如果使用 oc delete kafkatopic my-topic-1 删除主题，则会在元数据中添加时间戳：

删除时的终结器时间戳

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
```

```
generation: 1
name: my-topic-1
finalizers:
  - strimzi.io/topic-operator
labels:
  strimzi.io/cluster: my-cluster
deletionTimestamp: 20230301T000000.000
```

资源仍然存在。如果删除失败，则会在资源的状态中显示。

当成功执行最终任务时，终结器会从元数据中删除，并且资源会被完全删除。

终结器还可用于防止相关资源被删除。如果单向主题 Operator 未运行，则无法从 metadata.finalizers 中删除其终结器。并且任何尝试直接删除 KafkaTopic 资源，或者命名空间将失败或超时，使命名空间处于卡住终止状态。如果发生这种情况，您可以通过删除 主题上的终结器 来绕过最终化过程。

10.9. 在主题 OPERATOR 模式间切换

在升级或降级 Apache Kafka 的 Streams 时，或者在使用同一版本的 Apache Kafka 时切换主题管理模式，只要该版本支持模式。

从双向切换到单向主题管理模式

1. 启用 UnidirectionalTopicOperator 功能门。

   Cluster Operator 以单向主题管理模式使用主题 Operator 部署实体 Operator。

2. 支持以双向主题管理模式运行的 Topic Operator 的内部主题不再需要，以便您可以删除 KafkaTopic 资源来管理它们：

   ```
   oc delete $(oc get kt -n <namespace_name> -o name | grep strimzi-store-topic) \
     && oc delete $(oc get kt -n <namespace_name> -o name | grep strimzi-topic-operator)
   ```

   此命令删除内部主题，其名称为启动 strimzi-store-topic 和 strimzi-topic-operator。

3.

存储消费者偏移和事务状态的内部主题必须保留在 Kafka 中。因此，您必须先停用主题 Operator 管理，然后才能删除 KafkaTopic 资源。

a.

停用对主题的管理：

```
oc annotate $(oc get kt -n <namespace_name> -o name | grep consumer-offsets)
strimzi.io/managed="false" \
  && oc annotate $(oc get kt -n <namespace_name> -o name | grep transaction-
state) strimzi.io/managed="false"
```

通过使用 strimzi.io/managed="false" 注解 KafkaTopic 资源，这表示 Topic Operator 不再管理这些主题。在本例中，我们将注释添加到用于管理内部主题的资源，其名称为启动 consumer-offsets 和 transaction-state。

b.

当其管理停用时，删除 KafkaTopic 资源（不会删除 Kafka 中的主题）：

```
oc delete $(oc get kt -n <namespace_name> -o name | grep consumer-offsets) \
  && oc delete $(oc get kt -n <namespace_name> -o name | grep transaction-state)
```

从单向切换到双向主题管理模式

1.

禁用 UnidirectionalTopicOperator 功能门。

Cluster Operator 以双向主题管理模式部署带有主题 Operator 的 Entity Operator。

以双向主题管理模式运行的 Topic Operator 所需的内部主题会被创建。

2.

检查是否使用终结器来控制删除主题。如果 KafkaTopic 资源使用终结器，请确保在进行交换机后执行以下操作之一：

- 从主题中删除所有终结器。
- 通过在 Topic Operator env 配置中将 STRIMZI_USE_FINALIZERS 环境变量设置为 false 来禁用终结器。

对在 Apache Kafka 管理的集群的 Streams 中运行的主题 Operator 或独立部署使用相同的配置。

在 Apache Kafka 管理的集群的流中禁用主题终结器

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  entityOperator:
    topicOperator: {}
    userOperator: {}
    template:
      topicOperatorContainer:
        env:
          - name: STRIMZI_USE_FINALIZERS
            value: "false"
# ...
```

在独立部署中禁用主题终结器

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-topic-operator
spec:
  template:
    spec:
      containers:
        - name: STRIMZI_USE_FINALIZERS
          value: "false"
# ...
```

主题 Operator 在双向模式中不使用终结器。如果在从单向模式进行切换后保留它们，您将无法删除 KafkaTopic 和相关资源。

在主题 Operator 模式间切换后，尝试创建主题以确保 Operator 正确运行。如需更多信息，请参阅第 10.4 节 “配置 Kafka 主题”。

10.10. 删除主题的终结器

如果单向主题 Operator 未运行，并且您希望在删除受管主题时绕过最终化过程，则必须删除终结器。您可以通过直接编辑资源或使用 命令手动进行此操作。

要删除所有主题的终结器，请使用以下命令：

删除主题的终结器

```
oc get kt -o=json | jq '.items[].metadata.finalizers = null' | oc apply -f -
```

命令使用 jq 命令行 JSON 解析器工具 通过将终结器设置为 null 来修改 KafkaTopic (kt)资源。您还可以将命令用于特定主题：

删除特定主题上的终结器

```
oc get kt <topic_name> -o=json | jq '.metadata.finalizers = null' | oc apply -f -
```

运行此命令后，您可以继续并删除主题。或者，如果主题已被删除，但因为未完成的终结器而被阻止，则其删除应该完成。

警告

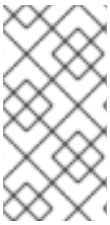
删除终结器时要小心，因为如果主题 Operator 未运行，则不会执行与最终化进程关联的清理操作。例如，如果您从 KafkaTopic 资源中删除终结器，然后删除该资源，则不会删除相关的 Kafka 主题。

10.11. 禁用主题删除时的注意事项

当 Kafka 中的 delete.topic.enable 配置设置为 false 时，无法删除主题。在某些情况下，这可能是必需的，但在单向模式中使用 Topic Operator 时引入了考虑。

由于无法删除主题，添加到 KafkaTopic 资源的元数据中以控制主题删除的终结器不会被主题 Operator 删除（但可以手动删除它们）。同样，任何与主题关联的自定义资源定义(CRD)或命名空间都不能被删除。

在配置 delete.topic.enable=false 前，评估这些影响以确保其与您的特定要求保持一致。



注意

为了避免使用终结器，您可以将 STRIMZI_USE_FINALIZERS Topic Operator 环境变量设置为 false。

10.12. 为主题操作调整请求批处理

在单向模式中，主题 Operator 使用 Kafka Admin API 的请求批处理功能来对主题资源执行操作。您可以使用以下操作器配置属性微调批处理机制：

- STRIMZI_MAX_QUEUE_SIZE 来设置主题事件队列的最大大小。默认值为 1024。
- STRIMZI_MAX_BATCH_SIZE 设置单个批处理中允许的最大主题事件数。默认值为 100。
- max_BATCH_LINGER_MS 指定在处理前批处理要积累项目的最长时间。默认值为 100 毫秒。

如果超过请求批处理队列的最大大小，则主题 Operator 会关闭并重启。要防止频繁重启，请考虑调整 STRIMZI_MAX_QUEUE_SIZE 属性以适应典型的负载。

第 11 章 使用 USER OPERATOR 管理 KAFKA 用户

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。

有关 KafkaUser 资源的更多信息，请参阅 KafkaUser 模式参考。

11.1. 配置 KAFKA 用户

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，并使用 oc delete 删除现有用户。

例如：

- oc apply -f *<user_config_file>*
- oc delete KafkaUser *<user_name>*

用户代表 Kafka 客户端。配置 Kafka 用户时，您可以启用客户端访问 Kafka 所需的用户身份验证和授权机制。使用的机制必须与等效的 Kafka 配置匹配。有关使用 Kafka 和 KafkaUser 资源保护对 Kafka 代理的访问的更多信息，请参阅 保护对 Kafka 代理的访问。

先决条件

- 正在运行的 Kafka 集群使用 mTLS 身份验证和 TLS 加密配置 Kafka 代理监听程序。
- 正在运行的用户 Operator （通常使用 Entity Operator 部署）。

流程

1. 配置 KafkaUser 资源。

这个示例使用 ACL 指定 mTLS 身份验证和简单授权。

Kafka 用户配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user-1
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: tls
  authorization:
    type: simple
    acls:
      # Example consumer Acls for topic my-topic using consumer group my-group
      - resource:
          type: topic
          name: my-topic
          patternType: literal
        operations:
          - Describe
          - Read
        host: "*"
      - resource:
          type: group
          name: my-group
          patternType: literal
        operations:
          - Read
        host: "*"
      # Example Producer Acls for topic my-topic
      - resource:
          type: topic
          name: my-topic
          patternType: literal
        operations:
          - Create
          - Describe
          - Write
        host: "*"
```

2. 在 OpenShift 中创建 KafkaUser 资源。

```
oc apply -f <user_config_file>
```

3. 等待用户的就绪状态更改为 True ：

```
oc get kafkausers -o wide -w -n <namespace>
```

Kafka 用户状态

```
NAME      CLUSTER    AUTHENTICATION  AUTHORIZATION READY
my-user-1  my-cluster  tls           simple       True
my-user-2  my-cluster  tls           simple
my-user-3  my-cluster  tls           simple       True
```

当 READY 输出显示为 True 时，用户创建成功。

4. 如果 READY 列留空，请从资源 YAML 或 User Operator 日志中获取有关状态的更多详细信息。

消息提供有关当前状态原因的详细信息。

```
oc get kafkausers my-user-2 -o yaml
```

具有 NotReady 状态的用户详情

```
# ...
status:
  conditions:
  - lastTransitionTime: "2022-06-10T10:07:37.238065Z"
    message: Simple authorization ACL rules are configured but not supported in the
      Kafka cluster configuration.
    reason: InvalidResourceException
    status: "True"
    type: NotReady
```

在本例中，用户未就绪的原因是在 Kafka 配置中没有启用简单的授权。

用于简单授权的 Kafka 配置

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

更新 Kafka 配置后，状态会显示用户就绪。

```
oc get kafkausers my-user-2 -o wide -w -n <namespace>
```

用户的状态更新

```
NAME       CLUSTER     AUTHENTICATION  AUTHORIZATION READY
my-user-2  my-cluster  tls             simple        True
```

获取详细信息不会显示任何消息。

```
oc get kafkausers my-user-2 -o yaml
```

具有 READY 状态的用户的详情

```
# ...
status:
  conditions:
```

```
- lastTransitionTime: "2022-06-10T10:33:40.166846Z"
  status: "True"
  type: Ready
```

第 12 章 使用红帽构建的 APICURIO REGISTRY 验证模式

您可以将红帽构建的 Apicurio Registry 与 Streams for Apache Kafka 搭配使用。

Apicurio Registry 是一个数据存储，用于在 API 和事件驱动的构架中共享标准事件模式和 API 设计。您可以使用 Apicurio Registry 将数据的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您的数据类型和 API 描述。

Apicurio Registry 存储用于序列化和反序列化消息的模式，然后可以从客户端应用程序引用这些消息，以确保它们发送和接收的信息与这些模式兼容。Apicurio Registry 为 Kafka 生成者和消费者应用程序提供 Kafka 客户端序列化器/反序列化器。Kafka 制作者应用程序使用 serializers 对符合特定事件模式的信息进行编码。Kafka 消费者应用程序使用反序列化器，它根据特定的模式 ID 验证信息是否被序列化使用正确的模式。

您可以使应用程序使用 registry 中的模式。这样可确保一致的模式使用，并有助于防止运行时出现数据错误。

其他资源

- Red Hat build of Apicurio Registry 产品文档
- 红帽构建的 Apicurio Registry 基于 GitHub: Apicurio/apicurio-registry上提供的 Apicurio Registry 构建

第 13 章 与红帽构建的 DEBEZIUM 集成以更改数据捕获

红帽构建的 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。您可以将红帽构建的 Debezium 与 Apache Kafka 的 Streams 一起部署并集成。在部署了 Apache Kafka 的 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给 OpenShift 上的 Apache Kafka 的流。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 Apache Kafka 的 Streams 部署红帽构建的 Debezium 的更多信息，请参阅产品文档。文档包括 *Debezium 入门指南*，指导您完成设置数据库更新事件记录所需的服务和连接器。

第 14 章 设置 KAFKA 集群的客户端访问权限

为 Apache Kafka 部署 Streams 后，您可以设置对 Kafka 集群的客户端访问。要验证部署，您可以部署示例制作者和消费者客户端。否则，创建在 OpenShift 集群内或外部提供客户端访问的监听程序。

14.1. 部署示例客户端

部署示例生成者和消费者客户端以发送和接收消息。您可以使用这些客户端验证 Apache Kafka 的 Streams 部署。

先决条件

- Kafka 集群可用于客户端。

流程

1. 部署 Kafka producer。

```
oc run kafka-producer -ti --image=registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0 --rm=true --restart=Never -- bin/kafka-console-producer.sh --bootstrap-server cluster-name-kafka-bootstrap:9092 --topic my-topic
```

2. 在运行制作者的控制台中输入消息。

3. 按 *Enter* 来发送邮件。

4. 部署 Kafka 使用者。

```
oc run kafka-consumer -ti --image=registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0 --rm=true --restart=Never -- bin/kafka-console-consumer.sh --bootstrap-server cluster-name-kafka-bootstrap:9092 --topic my-topic --from-beginning
```

5. 确认您在使用者控制台中看到传入的信息。

14.2. 配置监听程序以连接到 KAFKA 代理

使用监听程序进行与 Kafka 代理的客户端连接。Apache Kafka 的流提供了一个通用 GenericKafkaListener 模式，它带有通过 Kafka 资源配置监听程序的属性。GenericKafkaListener 提供了灵活的监听程序配置方法。您可以指定属性来配置 *内部监听程序*，以便在 OpenShift 集群内或 *外部监听程序* 连接 OpenShift 集群外部监听程序。

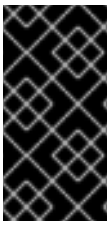
指定在监听器配置中公开 Kafka 的连接类型。所选的类型取决于您的要求，以及您的环境和基础架构。支持以下监听程序类型：

内部监听程序

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

外部监听程序

- NodePort 使用 OpenShift 节点上的端口
- LoadBalancer 使用负载均衡器服务
- ingress 使用 Kubernetes Ingress 和 Ingress NGINX Controller for Kubernetes（仅限 Kubernetes）
- route 使用 OpenShift Route 和默认的 HAProxy 路由器（仅限 OpenShift）



重要

不要在 OpenShift 上使用 ingress，改为使用 route 类型。Ingress NGINX Controller 仅适用于 Kubernetes。路由类型 只在 OpenShift 中被支持。

内部 类型监听程序配置使用无头服务和提供给代理 pod 的 DNS 名称。您可能希望将 OpenShift 网络加入外部网络。在这种情况下，您可以配置 内部 类型监听程序（使用 useServiceDnsDomain 属性），以便不使用 OpenShift 服务 DNS 域（通常为 .cluster.local）。您还可以配置 cluster-ip 类型，它根据每个代理的 ClusterIP 服务公开 Kafka 集群。当您无法通过无头服务路由或者您希望纳入自定义访问机制时，这个选项是一个实用的选项。例如，您可以在为特定 Ingress 控制器或 OpenShift 网关 API 构建自己的类型的外部监听程序时使用此监听程序。

外部监听程序处理从需要不同身份验证机制的网络访问 Kafka 集群。您可以使用指定的连接机制（如 loadbalancer 或 route）为 OpenShift 环境以外的客户端配置外部监听程序。例如，loadbalancers 可能不适用于某些基础架构，如裸机，节点端口提供了更好的选择。

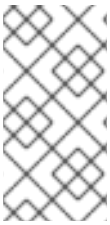
每个监听程序在 Kafka 资源中被定义为一个数组。

侦听器配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
        configuration:
          useServiceDnsDomain: true
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
      - name: external1
        port: 9094
        type: route
        tls: true
        configuration:
          brokerCertChainAndKey:
            secretName: my-secret
            certificate: my-certificate.crt
            key: my-key.key
    # ...
```

您可以根据需要配置多个监听程序，只要它们的名称和端口是唯一的。您还可以使用身份验证为安全连接配置监听程序。

如果要了解更多有关每个连接类型的 pros 和 conss 的信息，请参阅 在 Strimzi 中访问 Apache Kafka。



注意

如果您在使用外部监听程序时需要扩展 Kafka 集群，可能会触发所有 Kafka 代理的滚动更新。这取决于配置。

其他资源

- GenericKafkaListener 模式参考

14.3. 侦听器命名约定

在监听器配置中，生成的监听程序 bootstrap 和每个代理服务名称会根据以下命名约定进行构建：

表 14.1. 侦听器命名约定

侦听器类型	bootstrap 服务名称	per-Broker 服务名称
internal	<cluster_name>-kafka-bootstrap	*Not applicable*
LoadBalancer nodeport ingress route cluster-ip	<cluster_name>-kafka-<listener-name>-bootstrap	<cluster_name>-kafka-<listener-name>-<idx>

例如，my-cluster-kafka-bootstrap、my-cluster-kafka-external1-bootstrap 和 my-cluster-kafka-external1-0。名称分配给通过监听程序配置创建的服务、路由、负载均衡器和入口。

您可以使用某些向后兼容名称和端口号来转换监听程序最初在停用的 KafkaListeners schema 下配置。生成的外部监听程序命名规则稍有不同。这些特定监听程序名称和端口配置值组合向后兼容：

表 14.2. 后向兼容的监听程序名称和端口组合

侦听器名称	端口	bootstrap 服务名称	per-Broker 服务名称
plain	**9092**	<cluster_name>-kafka-bootstrap	*Not applicable*
tls	**9093**	<cluster-name>-kafka-bootstrap	*Not applicable*

侦听器名称	端口	bootstrap 服务名称	per-Broker 服务名称
external	**9094**	<cluster_name>-kafka-bootstrap	<cluster_name>-kafka-bootstrap-<idx>

14.4. 使用监听程序设置对 KAFKA 集群的客户端访问

使用 Kafka 集群的地址，您可以提供对同一 OpenShift 集群中的客户端的访问权限；或者，提供对不同 OpenShift 命名空间或 OpenShift 外部客户端的外部访问权限。此流程演示了如何配置从 OpenShift 外部或从另一个 OpenShift 集群配置对 Kafka 集群的客户端访问。

Kafka 侦听器提供对 Kafka 集群的访问。客户端访问是通过以下配置进行保护的：

1. 为 Kafka 集群配置了外部监听程序，并启用了 TLS 加密和 mTLS 身份验证，并启用了 Kafka 简单 授权。
2. 为客户端创建一个 KafkaUser，带有 mTLS 身份验证，并为 简单 授权定义访问控制列表 (ACL)。

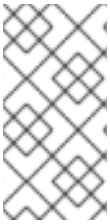
您可以将监听程序配置为使用 mutual tls、scram-sha-512 或 oauth 身份验证。mTLS 始终使用加密，但在使用 SCRAM-SHA-512 和 OAuth 2.0 身份验证时也建议使用加密。

您可以为 Kafka 代理配置简单、oauth、opa 或 自定义授权。启用后，授权将应用到所有已启用的监听程序。

当您配置 KafkaUser 身份验证和授权机制时，请确保它们与对应的 Kafka 配置匹配：

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

您应该至少有一个监听程序支持您要用于 KafkaUser 的身份验证。



注意

Kafka 用户和 Kafka 代理之间的身份验证取决于每个代理的身份验证设置。例如，如果在 Kafka 配置中也未启用，则无法验证带有 mTLS 的用户。

Apache Kafka operator 的 Streams 会自动配置过程并创建身份验证所需的证书：

- 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，以便使用 Kafka 集群启用身份验证。
- User Operator 根据所选的验证类型，创建代表客户端的用户以及用于客户端身份验证的安全凭证。

您可以将证书添加到客户端配置中。

在此过程中，会使用 Cluster Operator 生成的 CA 证书，但 您可以通过安装自己的证书 来替换它们。您还可以将侦听器 配置为使用由外部 CA （证书颁发机构）管理的 Kafka 侦听器证书。

证书以 PEM (.crt) 和 PKCS #12 (.p12) 格式提供。此流程使用 PEM 证书。使用带有 X.509 格式的证书的 PEM 证书。

注意

对于同一 OpenShift 集群和命名空间中的内部客户端，您可以在 Pod 规格中挂载集群 CA 证书。如需更多信息，请参阅配置内部客户端以信任集群 CA。

先决条件

- Kafka 集群可供在 OpenShift 集群外部运行的客户端进行连接
- Cluster Operator 和 User Operator 在集群中运行

流程

1. 使用 Kafka 侦听器配置 Kafka 集群。

- 定义通过监听程序访问 Kafka 代理所需的身份验证。
- 在 Kafka 代理上启用授权。

侦听器配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    listeners: 1
    - name: external1 2
      port: 9094 3
      type: <listener_type> 4
      tls: true 5
      authentication:
        type: tls 6
      configuration: 7
        #...
    authorization: 8
      type: simple
      superUsers:
        - super-user-name 9
  # ...
```

1 在 Generic Kafka listener schema reference 中描述了启用外部监听程序的配置选项。

2 用于标识监听程序的名称。在 Kafka 集群中必须是唯一的。

3

4

外部监听器类型指定为 route (只限 OpenShift), loadbalancer, nodeport 或 ingress (只限 Kubernetes)。内部监听程序指定为 internal 或 cluster-ip。

5

必需。侦听器上的 TLS 加密。对于 route 和 ingress 类型监听程序，必须设置为 true。对于 mTLS 身份验证，也使用 authentication 属性。

6

侦听器上的客户端身份验证机制。对于使用 mTLS 的服务器和客户端身份验证，您可以指定 tls: true 和 authentication.type: tls。

7

（可选）根据监听器类型的要求，您可以指定额外的 监听程序配置。

8

授权指定为 simple，它使用 AclAuthorizer 和 StandardAuthorizer Kafka 插件。

9

（可选）无论 ACL 中定义的任何访问限制，Super 用户可以访问所有代理。

警告

OpenShift 路由地址由 Kafka 集群名称、侦听器名称、项目名称和路由器的域组成。例如，my-cluster-kafka-external1-bootstrap-my-project.domain.com (<cluster_name>-kafka-<listener_name>-bootstrap-<namespace>.<domain>)。每个 DNS 标签(between period ".")不得超过 63 个字符，地址的总长度不得超过 255 个字符。

2. 创建或更新 Kafka 资源。

-

```
oc apply -f <kafka_configuration_file>
```

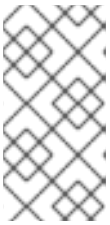
Kafka 集群使用 mTLS 身份验证配置 Kafka 代理监听程序。

为每个 Kafka 代理 pod 创建一个服务。

创建一个服务作为与 Kafka 集群连接的 *bootstrap 地址*。

一个服务也作为*外部 bootstrap 地址*为到 Kafka 集群的外部连接创建，使用 nodeport 监听器。

在 secret <cluster *_name> -cluster*-ca-cert 中还创建了用于验证 kafka 代理的身份的集群 CA 证书。



注意

如果您在使用外部监听程序时需要扩展 Kafka 集群，可能会触发所有 Kafka 代理的滚动更新。这取决于配置。

3. 从 Kafka 资源的状态中检索可用于访问 Kafka 集群的 bootstrap 地址。

```
oc get kafka <kafka_cluster_name> -o=jsonpath='{.status.listeners[?
(@.name=="<listener_name>")].bootstrapServers}{"\n"}'
```

例如：

```
oc get kafka my-cluster -o=jsonpath='{.status.listeners[?
(@.name=="external")].bootstrapServers}{"\n"}'
```

使用 Kafka 客户端中的 bootstrap 地址连接到 Kafka 集群。

4. 创建或修改代表需要访问 Kafka 集群的客户端的用户。

- 指定与 Kafka 侦听器相同的身份验证类型。

- 指定用于 简单 授权的授权 ACL。

用户配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster 1
spec:
  authentication:
    type: tls 2
  authorization:
    type: simple
    acls: 3
      - resource:
          type: topic
          name: my-topic
          patternType: literal
        operations:
          - Describe
          - Read
      - resource:
          type: group
          name: my-group
          patternType: literal
        operations:
          - Read
```

1 标签必须与 Kafka 集群的标签匹配。

2 将身份验证指定为 mutual tls。

3

5. 创建或修改 KafkaUser 资源。

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

创建了用户，以及名称与 KafkaUser 资源相同的 secret。secret 包含 mTLS 身份验证的公钥和私钥。

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> # Public key of the clients CA
  user.crt: <user_certificate> # Public key of the user
  user.key: <user_private_key> # Private key of the user
  user.p12: <store> # PKCS #12 store for user certificates and keys
  user.password: <password_for_store> # Protects the PKCS #12 store
```

6. 从 Kafka 集群的 &lt ;*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
```

7. 从 < *user_name&gt*; secret 中提取用户 CA 证书。

```
oc get secret <user_name> -o jsonpath='{.data.user\.crt}' | base64 -d > user.crt
```

8. 从 < *user_name&gt*; secret 中提取用户的私钥。

```
oc get secret <user_name> -o jsonpath='{.data.user\.key}' | base64 -d > user.key
```

9. 使用 bootstrap 地址主机名和端口配置客户端，以连接到 Kafka 集群：

```
props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "<hostname>:<port>");
```

10. 使用 truststore 凭证配置客户端，以验证 Kafka 集群的身份。

指定公共集群 CA 证书。

truststore 配置示例

```
props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SSL");
props.put(SslConfigs.SSL_TRUSTSTORE_TYPE_CONFIG, "PEM");
props.put(SslConfigs.SSL_TRUSTSTORE_CERTIFICATES_CONFIG,
"<ca.crt_file_content>");
```

SSL 是 mTLS 验证的指定的安全协议。为通过 TLS 进行 SCRAM-SHA-512 验证指定 SASL_SSL。PEM 是信任存储的文件格式。

11. 使用密钥存储凭证配置客户端，以便在连接到 Kafka 集群时验证用户。

指定公钥和私钥。

keystore 配置示例

```
props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SSL");
props.put(SslConfigs.SSL_KEYSTORE_TYPE_CONFIG, "PEM");
props.put(SslConfigs.SSL_KEYSTORE_CERTIFICATE_CHAIN_CONFIG,
"<user.crt_file_content>");
props.put(SslConfigs.SSL_KEYSTORE_KEY_CONFIG, "<user.key_file_content>");
```

将密钥存储证书和私钥直接添加到配置中。以单行格式添加。在 BEGIN CERTIFICATE 和 END CERTIFICATE 分隔符之间，以换行符(\n)开始。使用 \n 结束原始证书中的每一行。

keystore 配置示例

```
props.put(SslConfigs.SSL_KEYSTORE_CERTIFICATE_CHAIN_CONFIG, "-----BEGIN CERTIFICATE-----
\n<user_certificate_content_line_1>\n<user_certificate_content_line_n>\n-----END CERTIFICATE---");
props.put(SslConfigs.SSL_KEYSTORE_KEY_CONFIG, "----BEGIN PRIVATE KEY-----
\n<user_key_content_line_1>\n<user_key_content_line_n>\n-----END PRIVATE KEY-----");
```

其他资源

- 第 15.1.1 节 “侦听器身份验证”
- 第 15.1.2 节 “Kafka 授权”
- 如果使用授权服务器，您可以使用基于令牌的身份验证和授权：
  - 第 15.4 节 “使用基于 OAuth 2.0 令牌的身份验证”
  - 第 15.5 节 “使用基于 OAuth 2.0 令牌的授权”

14.5. 使用节点端口访问 KAFKA

使用节点端口从 OpenShift 集群外部客户端访问 Apache Kafka 集群的流。

要连接到代理，您可以为 Kafka bootstrap 地址指定主机名和端口号，以及用于 TLS 加密的证书。

该流程显示基本的 nodeport 侦听器配置。您可以使用监听程序属性启用 TLS 加密(tls)并指定客户端

身份验证机制(身份验证)。使用 配置属性 添加额外的配置。例如，您可以在 nodeport 监听器中使用以下配置属性：

preferredNodePortAddressType

指定作为节点地址检查的第一个地址类型。

externalTrafficPolicy

指定服务是否将外部流量路由到节点本地端点还是集群范围的端点。

nodePort

覆盖 bootstrap 和代理服务的分配的节点端口号。

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

先决条件

- 正在运行的 Cluster Operator

在此过程中，Kafka 集群名称是 my-cluster。侦听器的名称是 external4。

流程

1. 配置一个 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
metadata:
  labels:
    app: my-cluster
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    listeners:
      - name: external4
        port: 9094
        type: nodeport
```

```
      tls: true
      authentication:
        type: tls
      # ...
  # ...
 zookeeper:
  # ...
```

2. 创建或更新资源。

```
oc apply -f <kafka_configuration_file>
```

在 secret my-cluster-cluster-ca-cert 中创建用于验证 kafka 代理身份的集群 CA 证书。

为每个 Kafka 代理以及外部 bootstrap 服务创建 NodePort 类型服务。

为 bootstrap 和代理创建的节点端口服务

```
NAME                                 TYPE      CLUSTER-IP      PORT(S)
my-cluster-kafka-external4-0         NodePort  172.30.55.13    9094:31789/TCP
my-cluster-kafka-external4-1         NodePort  172.30.250.248  9094:30028/TCP
my-cluster-kafka-external4-2         NodePort  172.30.115.81   9094:32650/TCP
my-cluster-kafka-external4-bootstrap  NodePort  172.30.30.23    9094:32650/TCP
```

用于客户端连接的 bootstrap 地址传播到 Kafka 资源的 status。

bootstrap 地址的状态示例

```
status:
 clusterId: Y_RJQDGKRXmNF7fEcWIdJQ
 conditions:
  - lastTransitionTime: '2023-01-31T14:59:37.113630Z'
    status: 'True'
    type: Ready
 kafkaVersion: 3.7.0
 listeners:
  # ...
  - addresses:
```

```
    - host: ip-10-0-224-199.us-west-2.compute.internal
      port: 32650
  bootstrapServers: 'ip-10-0-224-199.us-west-2.compute.internal:32650'
  certificates:
    - |
      -----BEGIN CERTIFICATE-----

      -----END CERTIFICATE-----
  name: external4
 observedGeneration: 2
 operatorLastSuccessfulVersion: 2.7
# ...
```

3. 从 Kafka 资源的状态中检索可用于访问 Kafka 集群的 bootstrap 地址。

```
oc get kafka my-cluster -o=jsonpath='{.status.listeners[?
(@.name=="external4")].bootstrapServers}{"\n"}'

ip-10-0-224-199.us-west-2.compute.internal:32650
```

4. 提取集群 CA 证书。

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

5. 将您的客户端配置为连接到代理。

   a. 指定 Kafka 客户端中的 bootstrap 主机和端口作为连接到 Kafka 集群的 bootstrap 地址。例如，ip-10-0-224-199.us-west-2.compute.internal:32650。

   b. 将提取的证书添加到 Kafka 客户端的信任存储中，以配置 TLS 连接。

      如果启用了客户端身份验证机制，您还需要在客户端中配置它。

注意

如果您使用自己的监听程序证书，请检查您是否需要将 CA 证书添加到客户端的信任存储配置中。如果它是一个公共（外部）CA，通常不需要添加它。

14.6. 使用 LOADBALANCERS 访问 KAFKA

使用 loadbalancers 从 OpenShift 集群外部客户端访问 Apache Kafka 集群的流。

要连接到代理，您可以为 Kafka bootstrap 地址指定主机名和端口号，以及用于 TLS 加密的证书。

该流程显示基本的 loadbalancer 侦听器配置。您可以使用监听程序属性启用 TLS 加密(tls)并指定客户端身份验证机制(身份验证)。使用 配置属性 添加额外的配置。例如，您可以在 loadbalancer 监听器中使用以下配置属性：

loadBalancerSourceRanges

将流量限制为指定的 CIDR 列表(Classless Inter-Domain Routing)范围。

externalTrafficPolicy

指定服务是否将外部流量路由到节点本地端点还是集群范围的端点。

loadBalancerIP

在创建负载均衡器时请求特定的 IP 地址。

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

先决条件

- 正在运行的 Cluster Operator

在此过程中，Kafka 集群名称是 my-cluster。侦听器的名称是 external3。

流程

1. 配置一个 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
```

```
metadata:
  labels:
    app: my-cluster
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    listeners:
      - name: external3
        port: 9094
        type: loadbalancer
        tls: true
        authentication:
          type: tls
        # ...
    # ...
  zookeeper:
    # ...
```

2. 创建或更新资源。

```
oc apply -f <kafka_configuration_file>
```

在 secret my-cluster-cluster-ca-cert 中还创建了用于验证 kafka 代理身份的集群 CA 证书。

为每个 Kafka 代理创建 loadbalancer 类型服务和负载均衡器，以及外部 bootstrap 服务。

为 bootstrap 和代理创建的 LoadBalancer 服务和负载均衡器

```
NAME                                  TYPE           CLUSTER-IP      PORT(S)
my-cluster-kafka-external3-0          LoadBalancer   172.30.204.234  9094:30011/TCP
my-cluster-kafka-external3-1          LoadBalancer   172.30.164.89   9094:32544/TCP
my-cluster-kafka-external3-2          LoadBalancer   172.30.73.151   9094:32504/TCP
my-cluster-kafka-external3-bootstrap  LoadBalancer   172.30.30.228   9094:30371/TCP

NAME                                  EXTERNAL-IP (loadbalancer)
my-cluster-kafka-external3-0          a8a519e464b924000b6c0f0a05e19f0d-
1132975133.us-west-2.elb.amazonaws.com
my-cluster-kafka-external3-1          ab6adc22b556343afb0db5ea05d07347-
611832211.us-west-2.elb.amazonaws.com
my-cluster-kafka-external3-2          a9173e8ccb1914778aeb17eca98713c0-
777597560.us-west-2.elb.amazonaws.com
my-cluster-kafka-external3-bootstrap  a8d4a6fb363bf447fb6e475fc3040176-
36312313.us-west-2.elb.amazonaws.com
```

用于客户端连接的 bootstrap 地址传播到 Kafka 资源的 status。

bootstrap 地址的状态示例

```
status:
  clusterId: Y_RJQDGKRXmNF7fEcWIdJQ
  conditions:
    - lastTransitionTime: '2023-01-31T14:59:37.113630Z'
      status: 'True'
      type: Ready
  kafkaVersion: 3.7.0
  listeners:
    # ...
    - addresses:
        - host: >-
            a8d4a6fb363bf447fb6e475fc3040176-36312313.us-west-2.elb.amazonaws.com
          port: 9094
      bootstrapServers: >-
        a8d4a6fb363bf447fb6e475fc3040176-36312313.us-west-
2.elb.amazonaws.com:9094
      certificates:
        - |
          -----BEGIN CERTIFICATE-----

          -----END CERTIFICATE-----
      name: external3
  observedGeneration: 2
  operatorLastSuccessfulVersion: 2.7
# ...
```

用于客户端连接的 DNS 地址传播到每个 loadbalancer 服务的状态。

bootstrap loadbalancer 的状态示例

```
status:
  loadBalancer:
    ingress:
```

```
    - hostname: >-
      a8d4a6fb363bf447fb6e475fc3040176-36312313.us-west-2.elb.amazonaws.com
# ...
```

3. 从 Kafka 资源的状态中检索可用于访问 Kafka 集群的 bootstrap 地址。

```
oc get kafka my-cluster -o=jsonpath='{.status.listeners[?
(@.name=="external3")].bootstrapServers}{"\n"}'

a8d4a6fb363bf447fb6e475fc3040176-36312313.us-west-2.elb.amazonaws.com:9094
```

4. 提取集群 CA 证书。

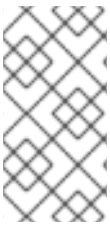
```
oc get secret my-cluster-cluster-ca-cert -o jsonpath='{.data.ca\.crt}' | base64 -d > ca.crt
```

5. 将您的客户端配置为连接到代理。

   a. 指定 Kafka 客户端中的 bootstrap 主机和端口作为连接到 Kafka 集群的 bootstrap 地址。例如，a8d4a6fb363bf447fb6e475fc3040176-36312313.us-west-2.elb.amazonaws.com:9094。

   b. 将提取的证书添加到 Kafka 客户端的信任存储中，以配置 TLS 连接。

      如果启用了客户端身份验证机制，您还需要在客户端中配置它。



注意

如果您使用自己的监听程序证书，请检查您是否需要将 CA 证书添加到客户端的信任存储配置中。如果它是一个公共（外部）CA，通常不需要添加它。

14.7. 使用 OPENSHIFT 路由访问 KAFKA

使用 OpenShift 路由从 OpenShift 集群外部的客户端访问 Apache Kafka 集群的流。

为了可以使用路由，请在 Kafka 自定义资源中为 route 类型监听程序添加配置。应用时，配置会为外部 bootstrap 和集群中的每个代理创建一个专用路由和服务。客户端连接到 bootstrap 路由，该路由通过 bootstrap 服务将其路由到代理。然后，使用 DNS 名称建立每个代理连接，该名称通过特定于代理的路由和服务将流量从客户端路由到代理。

要连接到代理，您可以为路由 bootstrap 地址指定一个主机名，以及用于 TLS 加密的证书。对于使用路由访问，端口始终为 443。

警告

OpenShift 路由地址由 Kafka 集群名称、侦听器名称、项目名称和路由器的域组成。例如，my-cluster-kafka-external1-bootstrap-my-project.domain.com (<cluster_name>-kafka-<listener_name>-bootstrap-<namespace>.<domain>)。每个 DNS 标签(between period ".")不得超过 63 个字符，地址的总长度不得超过 255 个字符。

该流程显示基本的监听程序配置。必须启用 TLS 加密(tls)。您还可以指定客户端身份验证机制(身份验证)。使用 配置属性 添加额外的配置。例如，您可以使用 host 配置属性 和路由 监听程序来指定 bootstrap 和 per-broker 服务使用的主机名。

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

TLS passthrough

对于 Apache Kafka 的 Streams 创建的路由，启用了 TLS 透传。Kafka 通过 TCP 使用二进制协议，但路由设计为使用 HTTP 协议。为了能够通过路由路由路由 TCP 流量，Apache Kafka 的 Streams 使用 Server Name Indication (SNI)的 TLS 透传。

SNI 有助于识别并传递到 Kafka 代理的连接。在 passthrough 模式中，始终使用 TLS 加密。由于连接传递给代理，因此监听程序使用由内部集群 CA 签名的 TLS 证书，而不是入口证书。要将监听程序配置为使用您自己的监听程序证书，请使用 brokerCertChainAndKey 属性。

先决条件

- 正在运行的 Cluster Operator

在此过程中，Kafka 集群名称是 my-cluster。侦听器的名称是 external1。

流程

1. 配置 Kafka 资源，将外部监听程序设置为 route 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: external1
        port: 9094
        type: route
        tls: true 1
        authentication:
          type: tls
        # ...
    # ...
  zookeeper:
    # ...
```

1. 对于 路由类型 监听程序，必须启用 TLS 加密(true)。

2. 创建或更新资源。

```
oc apply -f <kafka_configuration_file>
```

   在 secret my-cluster-cluster-ca-cert 中创建用于验证 kafka 代理身份的集群 CA 证书。

   为每个 Kafka 代理以及外部 bootstrap 服务创建 ClusterIP 类型服务。

另外，还会为每个服务创建一个路由，其中包含一个 DNS 地址 (host/port)，以使用默认的 OpenShift HAProxy 路由器公开它们。

路由预先配置了 TLS 透传。

为 bootstrap 和代理创建的路由

```
NAME                                HOST/PORT                                                  SERVICES
PORT  TERMINATION
my-cluster-kafka-external1-0        my-cluster-kafka-external1-0-my-project.router.com
my-cluster-kafka-external1-0        9094  passthrough
my-cluster-kafka-external1-1        my-cluster-kafka-external1-1-my-project.router.com
my-cluster-kafka-external1-1        9094  passthrough
my-cluster-kafka-external1-2        my-cluster-kafka-external1-2-my-project.router.com
my-cluster-kafka-external1-2        9094  passthrough
my-cluster-kafka-external1-bootstrap  my-cluster-kafka-external1-bootstrap-my-
project.router.com my-cluster-kafka-external1-bootstrap  9094  passthrough
```

用于客户端连接的 DNS 地址传播到 每个路由的状态。

bootstrap 路由的状态示例

```
status:
  ingress:
    - host: >-
        my-cluster-kafka-external1-bootstrap-my-project.router.com
  # ...
```

3. 使用目标代理使用 OpenSSL s_client 检查端口 443 上的客户端-服务器 TLS 连接。

```
openssl s_client -connect my-cluster-kafka-external1-0-my-project.router.com:443 -
servername my-cluster-kafka-external1-0-my-project.router.com -showcerts
```

服务器名称是将连接传递给代理的 Server Name Indication (SNI)。

如果连接成功，则返回代理的证书。

代理的证书

```
Certificate chain
 0 s:O = io.strimzi, CN = my-cluster-kafka
   i:O = io.strimzi, CN = cluster-ca v0
```

4. 从 Kafka 资源的状态检索 bootstrap 服务的地址。

```
oc get kafka my-cluster -o=jsonpath='{.status.listeners[?
(@.name=="external1")].bootstrapServers}{"\n"}'

my-cluster-kafka-external1-bootstrap-my-project.router.com:443
```

该地址包含 Kafka 集群名称、侦听器名称、项目名称和路由器域（本例中为router.com ）。

5. 提取集群 CA 证书。

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

6. 将您的客户端配置为连接到代理。

   a. 指定 Kafka 客户端中的 bootstrap 服务和端口 443 的地址，作为连接到 Kafka 集群的 bootstrap 地址。

   b. 将提取的证书添加到 Kafka 客户端的信任存储中，以配置 TLS 连接。

      如果启用了客户端身份验证机制，您还需要在客户端中配置它。

注意

如果您使用自己的监听程序证书，请检查您是否需要将 CA 证书添加到客户端的信任存储配置中。如果它是一个公共（外部）CA，通常不需要添加它。

14.8. 返回服务的连接详情

通过服务发现，可以更轻松地在与 Apache Kafka 相同的 OpenShift 集群中运行的客户端应用程序，以便与 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
- Kafka Bridge 服务

该标签有助于使服务可以被发现，而注解则提供客户端应用程序的连接详情来建立连接。

Service 资源的服务发现标签 strimzi.io/discovery 设置为 true。服务发现注解具有相同的密钥，为每个服务提供 JSON 格式的连接详情。

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:
  #...
```

Kafka 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
```

在从命令行或对应的 API 调用获取服务时，通过指定发现标签来查找服务。

使用发现标签返回服务

```
oc get service -l strimzi.io/discovery=true
```

检索服务发现标签时返回连接详情。

第 15 章 保护对 KAFKA 的访问

通过管理客户端对 Kafka 代理的访问来保护 Kafka 集群。Kafka 代理和客户端之间的安全连接可以包括以下内容：

- 数据交换加密
- 证明身份的身份验证
- 允许或拒绝用户执行操作的授权

在 Apache Kafka 的 Streams 中，保护连接涉及配置监听程序和用户帐户：

侦听器配置

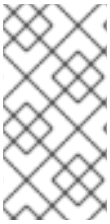
使用 Kafka 资源为到 Kafka 代理的客户端连接配置监听程序。侦听器定义客户端如何进行身份验证，如使用 mTLS、SCRAM-SHA-512、OAuth 2.0 或自定义身份验证方法。要提高安全性，请配置 TLS 加密以保护 Kafka 代理和客户端之间的通信。您可以通过在 Kafka 代理配置中指定支持的 TLS 版本和密码套件来进一步保护基于 TLS 的通信。

对于添加的保护层，请使用 Kafka 资源为 Kafka 集群指定授权方法，如简单、OAuth 2.0、OPA 或自定义授权。

用户帐户

使用 Apache Kafka Streams 中的 KafkaUser 资源设置用户帐户和凭证。用户代表您的客户端，并确定它们应该如何使用 Kafka 集群验证和授权。用户配置中指定的身份验证和授权机制必须与 Kafka 配置匹配。另外，定义访问控制列表(ACL)来控制用户对特定主题的访问以及更精细的授权。要进一步提高安全性，请指定用户配额，以根据字节率或 CPU 使用率限制客户端对 Kafka 代理的访问。

如果要限制其使用的 TLS 版本和密码套件，您还可以将制作者或消费者配置添加到客户端。客户端上的配置必须使用在代理上启用的协议和密码套件。



注意

如果您使用 OAuth 2.0 管理客户端访问，则用户身份验证和授权凭据将通过授权服务器进行管理。

Apache Kafka operator 的流自动化配置过程并创建身份验证所需的证书。Cluster Operator 会自动为集群中的数据加密和身份验证设置 TLS 证书。

15.1. KAFKA 的安全选项

使用 Kafka 资源来配置用于 Kafka 身份验证和授权的机制。

15.1.1. 侦听器身份验证

在创建监听程序时为 Kafka 代理配置客户端身份验证。使用 Kafka 资源中的 Kafka.spec.kafka.listeners.authentication 属性指定监听程序验证类型。

对于 OpenShift 集群中的客户端，您可以创建 plain （没有加密）或 tls *内部监听程序*。内部监听程序 类型使用无头服务，以及提供给代理 pod 的 DNS 名称。作为无头服务的替代选择，您还可以创建内部监听程序的 cluster-ip 类型，以使用每个代理的 ClusterIP 服务公开 Kafka。对于 OpenShift 集群外的客户端，您可以创建 *外部监听程序* 并指定连接机制，可以是 nodeport、loadbalancer、ingress （仅限 Kubernetes） 或路由 （仅限 OpenShift）。

有关连接外部客户端的配置选项的详情请参考 第 14 章 *设置 Kafka 集群的客户端访问权限*。

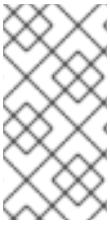
支持的身份验证选项：

1. mTLS 身份验证（只在启用了 TLS 的监听程序中）
2. SCRAM-SHA-512 身份验证
3. 基于 OAuth 2.0 令牌的身份验证

4. 自定义身份验证

5. TLS 版本和密码套件

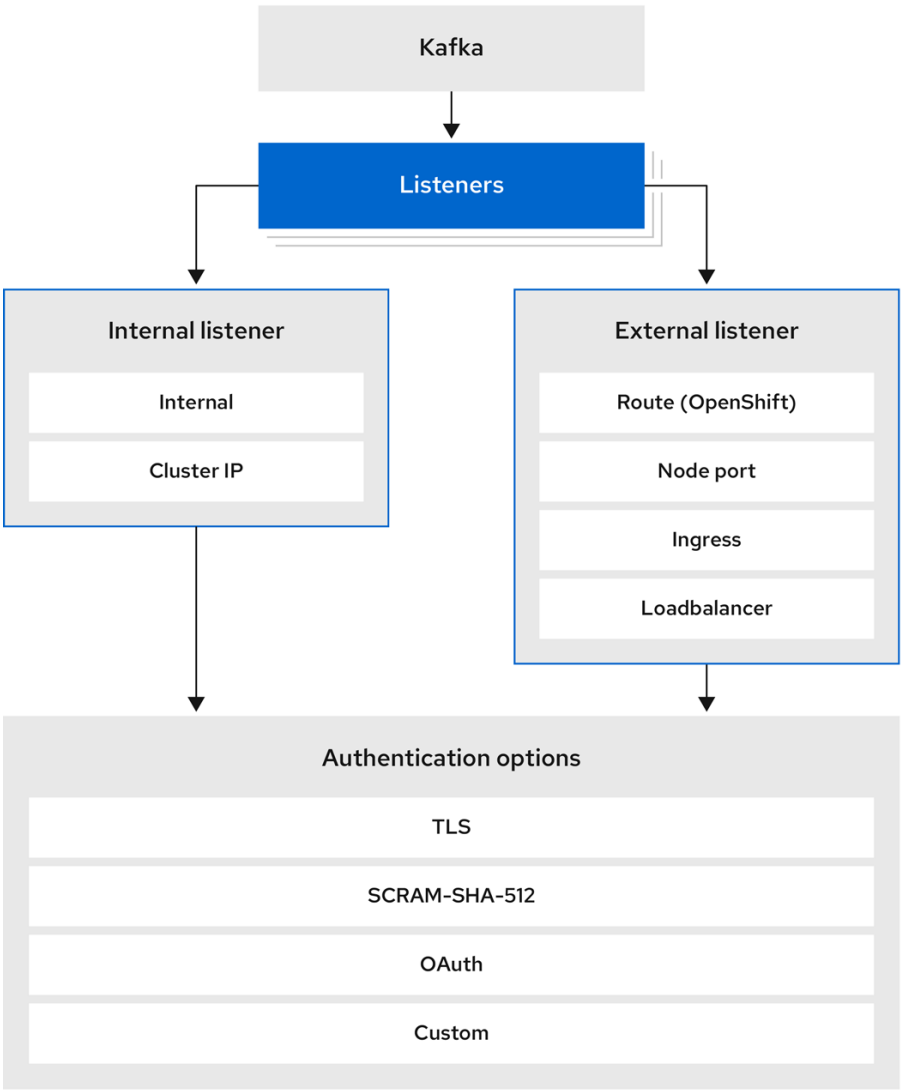
您选择的身份验证选项取决于您如何验证客户端对 Kafka 代理的访问。



注意

在使用自定义身份验证前，尝试了解标准身份验证选项。自定义身份验证允许任何类型的 Kafka 支持的身份验证。它可以提供更大的灵活性，但也增加了复杂性。

图 15.1. Kafka 侦听器身份验证选项



listener authentication 属性用于指定特定于该侦听器的身份验证机制。

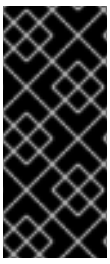
如果没有指定 身份验证 属性，则监听器不会验证通过该监听程序连接的客户端。侦听器将在没有身份验证的情况下接受所有连接。

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

以下示例显示了：

- 为 SCRAM-SHA-512 身份验证配置 纯文本
- 带有 mTLS 验证的 tls 侦听器
- 带有 mTLS 验证的 外部监听程序

每个监听程序都配置在 Kafka 集群中具有唯一的名称和端口。



重要

当为客户端配置监听程序对代理的访问时，您可以使用端口 9092 或更高版本(9093、9094 等)，但有一些例外。侦听程序无法配置为使用为 interbroker 通信(9090 和 9091)、Prometheus 指标(9404)和 JMX (Java 管理扩展)监控(9999)保留的端口。

侦听器身份验证配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    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: external3
      port: 9094
      type: loadbalancer
      tls: true
      authentication:
        type: tls
# ...
```

15.1.1.1. mTLS 身份验证

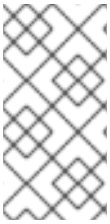
mTLS 身份验证总是用于 Kafka 代理和 ZooKeeper pod 之间的通信。

Apache Kafka 的流可将 Kafka 配置为使用 TLS （传输层安全）来提供 Kafka 代理和客户端之间的加密通信，而无需 mutual 身份验证。对于 mutual，或双向验证，服务器和客户端都存在证书。当您配置 mTLS 身份验证时，代理会验证客户端（客户端身份验证），客户端会验证代理（服务器身份验证）。

Kafka 资源中的 mTLS 侦听器配置需要以下内容：

- tls: true 指定 TLS 加密和服务器身份验证
- authentication.type: tls 来指定客户端身份验证

当 Cluster Operator 创建 Kafka 集群时，它会创建一个名为 < cluster_name>-cluster-ca-cert 的新 secret。secret 包含 CA 证书。CA 证书采用 PEM 和 PKCS #12 格式。要验证 Kafka 集群，请将 CA 证书添加到客户端配置中的信任存储中。若要验证客户端，请在客户端配置中添加用户证书和密钥到密钥存储。有关为 mTLS 配置客户端的详情请参考 第 15.2.2 节 “用户身份验证”。



注意

TLS 身份验证更为常见的单向，另一方验证另一方的身份。例如，当在 Web 浏览器和 Web 服务器之间使用 HTTPS 时，浏览器获取 Web 服务器身份的证明。

15.1.1.2. SCRAM-SHA-512 身份验证

SCRAM (Salted Challenge Response Authentication Mechanism)是一个身份验证协议，可以使用密码建立 mutual 身份验证。Apache Kafka 的流可将 Kafka 配置为使用 SASL (Simple Authentication and Security Layer) SCRAM-SHA-512，以便在未加密的和加密的客户端连接上提供身份验证。

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

以下 SCRAM 属性使得在未加密的连接中也安全地使用 SCRAM-SHA-512 ：

- 密码不会通过通信通道以明文形式发送。取而代之，客户端和服务器都是由另一个挑战的挑战，以提供对其了解验证用户的密码的证明。
- 服务器和客户端各自为每个身份验证交换生成一个新的挑战。这意味着，交换具有弹性地应对重播攻击。

当使用 scram-sha-512 配置 KafkaUser.spec.authentication.type 时，User Operator 将生成一个由大写和小写 ASCII 字母和数字组成的随机 12 个字符密码。

15.1.1.3. 网络策略

默认情况下，Apache Kafka 的 Streams 会自动为每个在 Kafka 代理上启用的监听程序创建一个 NetworkPolicy 资源。此 NetworkPolicy 允许应用程序连接到所有命名空间中的监听程序。使用网络策略作为监听器配置的一部分。

如果要将网络级别的监听程序的访问权限限制为所选应用程序或命名空间，请使用 networkPolicyPeers 属性。每个监听程序都可以有不同的 networkPolicyPeers 配置。有关网络策略对等点的更多信息，请参阅 NetworkPolicyPeer API 参考。

如果要使用自定义网络策略，您可以在 Cluster Operator 配置中将 STRIMZI_NETWORK_POLICY_GENERATION 环境变量设置为 false。如需更多信息，请参阅 第 9.5 节

“配置 Cluster Operator”。

注意

您的 OpenShift 配置必须支持 ingress NetworkPolicies，以便在 Apache Kafka 的 Streams 中使用网络策略。

15.1.1.4. 提供监听程序证书

您可以为启用了 TLS 加密的 TLS 监听程序或启用了 TLS 加密的外部监听程序提供您自己的服务器证书，称为 *Kafka 侦听器证书*。如需更多信息，请参阅 第 15.3.4 节 “为 TLS 加密提供自己的 Kafka 侦听器证书”。

其他资源

- GenericKafkaListener 模式参考

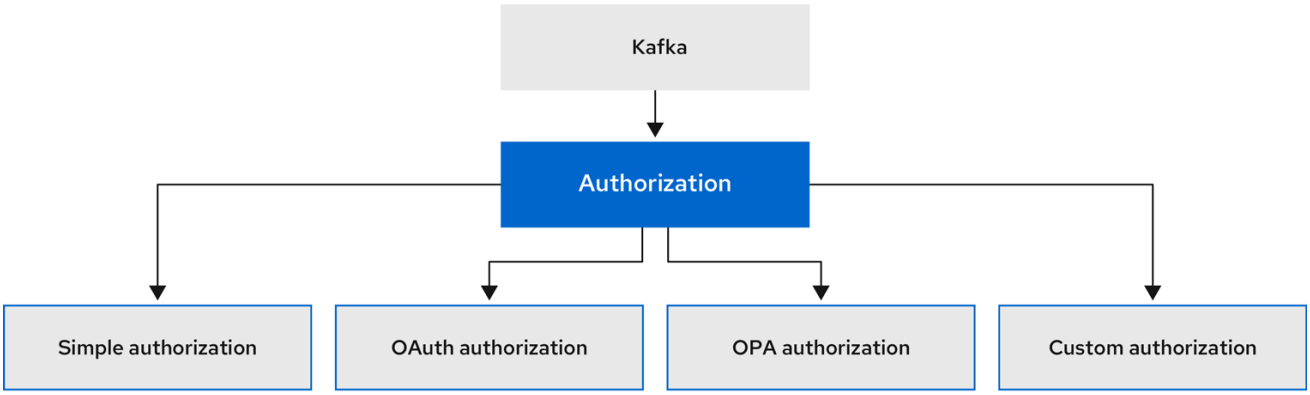
15.1.2. Kafka 授权

使用 Kafka 资源中的 Kafka.spec.kafka.authorization 属性为 Kafka 代理配置授权。如果缺少 authorization 属性，则不会启用授权，并且客户端没有限制。启用后，授权将应用到所有已启用的监听程序。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）
- 开放策略代理 (OPA) 授权
- 自定义授权

图 15.2. Kafka 集群授权选项



15.1.2.1. 超级用户

超级用户都可以访问 Kafka 集群中的所有资源，无论访问限制如何，且都由所有授权机制支持。

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 mTLS 身份验证，则用户名是 TLS 证书主题中的通用名称，前缀为 CN=。如果您不使用 User Operator 并将您自己的证书用于 mTLS，则用户名是完整的证书主题。一个完整的证书主题可以有以下字段：CN=user,OU=my_ou,O=my_org,L=my_location,ST=my_state,C=my_country_code.省略不存在的任何字段。

具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
        - CN=client_4,OU=my_ou,O=my_org,L=my_location,ST=my_state,C=US
        - CN=client_5,OU=my_ou,O=my_org,C=GB
        - CN=client_6,O=my_org
    # ...
```

15.2. 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，客户端以用户表示。

您可以验证和授权用户对 Kafka 代理的访问。身份验证允许访问，授权限制访问允许的操作。

您还可以创建对 Kafka 代理没有限制的访问权限的 *超级用户*。

身份验证和授权机制必须与 用于访问 Kafka 代理 的监听程序的规格 匹配。

有关配置 KafkaUser 资源以安全地访问 Kafka 代理的更多信息，请参阅 第 14.4 节 “使用监听程序设置对 Kafka 集群的客户端访问”。

15.2.1. 为用户处理识别 Kafka 集群

KafkaUser 资源包含一个标签，用于定义相应 Kafka 集群的名称（从其所属的 Kafka 资源的名称派生）。

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

User Operator 使用该标签来标识 KafkaUser 资源并创建新用户，并在以后的用户处理中。

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

如果 KafkaUser 资源的状态保持为空，请检查您的标签。

15.2.2. 用户身份验证

使用 KafkaUser 自定义资源为需要访问 Kafka 集群的用户（客户端）配置身份验证凭据。使用 KafkaUser.spec 中的 authentication 属性配置凭证。通过指定 类型，您可以控制生成的凭证。

支持的验证类型：

- 用于 mTLS 验证的 TLS
- 使用外部证书的 mTLS 验证的 tls-external
- 用于 SCRAM -SHA-512 验证的 SCRAM-sha -512

如果指定了 tls 或 scram-sha-512，则 User Operator 会在创建用户时创建身份验证凭据。如果指定了 tls-external，用户仍然使用 mTLS，但不会创建身份验证凭证。当您提供自己的证书时，使用这个选项。如果没有指定身份验证类型，User Operator 不会创建用户或其凭证。

您可以使用 tls-external 使用 User Operator 外部发布的证书与 mTLS 进行身份验证。User Operator 不生成 TLS 证书或 secret。您仍然可以通过 User Operator 管理 ACL 规则和配额，其方式与使用 tls 机制时相同。这意味着，您可以在指定 ACL 规则和配额时使用 CN=*USER-NAME* 格式。*USER-NAME* 是 TLS 证书中提供的通用名称。

15.2.2.1. mTLS 身份验证

要使用 mTLS 身份验证，您可以将 KafkaUser 资源中的 type 字段设置为 tls。

启用 mTLS 验证的用户示例

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

身份验证类型必须与用于访问 Kafka 集群的 Kafka 侦听器等效的配置匹配。

当用户由 User Operator 创建时，它会创建一个与 KafkaUser 资源名称相同的新 secret。secret 包含 mTLS 的私钥和公钥。公钥包含在用户证书中，该证书由创建时由客户端 CA （证书颁发机构）签名。所有密钥都是 X.509 格式。

注意

如果您使用 Cluster Operator 生成的客户端 CA，则当 Cluster Operator 更新客户端 CA 时，由 User Operator 生成的用户证书也会更新。

用户 secret 以 PEM 和 PKCS #12 的形式提供密钥和证书。

使用用户凭证的 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> # Public key of the clients CA
  user.crt: <user_certificate> # Public key of the user
  user.key: <user_private_key> # Private key of the user
  user.p12: <store> # PKCS #12 store for user certificates and keys
  user.password: <password_for_store> # Protects the PKCS #12 store
```

配置客户端时，您可以指定以下内容：

- 公共集群 CA 证书的 Truststore 属性用于验证 Kafka 集群的身份

- 用户身份验证凭证的 Keystore 属性用于验证客户端

配置取决于文件格式 (PEM 或 PKCS #12)。这个示例使用 PKCS #12 格式存储，以及访问存储中凭证所需的密码。

以 PKCS #12 格式使用 mTLS 的客户端配置示例

```
bootstrap.servers=<kafka_cluster_name>-kafka-bootstrap:9093 1
security.protocol=SSL 2
ssl.truststore.location=/tmp/ca.p12 3
ssl.truststore.password=<truststore_password> 4
ssl.keystore.location=/tmp/user.p12 5
ssl.keystore.password=<keystore_password> 6
```

1. 连接到 Kafka 集群的 bootstrap 服务器地址。
2. 使用 TLS 加密时的安全协议选项。
3. truststore 位置包含了 Kafka 集群的公钥证书 (ca.p12)。在创建 Kafka 集群时，Cluster Operator 在 < cluster_name>-cluster-ca-cert secret 中生成集群 CA 证书和密钥。
4. 用于访问信任存储的密码(ca.password)。
5. 密钥存储位置包含 Kafka 用户的公钥证书(user.p12)。
6. 用于访问密钥存储的密码(user.password)。

15.2.2.2. 使用 User Operator 外部发布的证书进行 mTLS 验证

要使用 User Operator 外部发布的证书使用 mTLS 身份验证，您可以将 KafkaUser 资源中的 type 字段设置为 tls-external。不会为用户创建 secret 和凭证。

使用 User Operator 外部发布的带有 mTLS 验证的用户示例

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

15.2.2.3. SCRAM-SHA-512 身份验证

要使用 SCRAM-SHA-512 身份验证机制，您可以将 KafkaUser 资源中的 type 字段设置为 scram-sha-512。

启用 SCRAM-SHA-512 验证的用户示例

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

当用户由 User Operator 创建时，它会创建一个与 KafkaUser 资源名称相同的新 secret。secret 包

含在 password 键中生成的密码，该密钥使用 base64 编码。要使用密码，必须对其进行解码。

使用用户凭证的 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) 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
```

15.2.2.3.1. 自定义密码配置

创建用户时，Apache Kafka 的 Streams 会生成一个随机密码。您可以使用自己的密码而不是 Streams for Apache Kafka 生成的密码。要做到这一点，使用密码创建一个 secret，并在 KafkaUser 资源中引用它。

为 SCRAM-SHA-512 验证设置了密码的用户示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: scram-sha-512
    password:
      valueFrom:
        secretKeyRef:
          name: my-secret 1
          key: my-password 2
  # ...
```

1 包含预定义密码的 secret 名称。

2 存储在机密中的密码的密钥。

15.2.3. 用户授权

使用 KafkaUser 自定义资源为需要访问 Kafka 集群的用户（客户端）配置授权规则。使用 KafkaUser.spec 中的 authorization 属性配置规则。通过指定 类型，您可以控制使用哪些规则。

要使用简单授权，您可以在 KafkaUser.spec.authorization 中将 type 属性设置为 simple。简单的授权使用 Kafka Admin API 管理 Kafka 集群中的 ACL 规则。是否启用 User Operator 中的 ACL 管理，是否取决于您的 Kafka 集群中的授权配置。

- 对于简单的授权，始终启用 ACL 管理。
- 对于 OPA 授权，始终禁用 ACL 管理。授权规则在 OPA 服务器中配置。
- 对于 Red Hat Single Sign-On 授权，您可以在 Red Hat Single Sign-On 中直接管理 ACL 规则。您还可以在配置中将授权委托给简单授权器作为回退选项。启用到简单授权器时，User

Operator 也会启用 ACL 规则的管理。

- 要使用自定义授权插件进行自定义授权，请使用 Kafka 自定义资源的 .spec.kafka.authorization 配置中的 supportsAdminApi 属性来启用或禁用支持。

授权是集群范围的。授权类型必须与 Kafka 自定义资源中的等效配置匹配。

如果没有启用 ACL 管理，如果包含任何 ACL 规则，则 Apache Kafka 的流会拒绝资源。

如果您使用 User Operator 的独立部署，则默认启用 ACL 管理。您可以使用 STRIMZI_ACLS_ADMIN_API_SUPPORTED 环境变量禁用它。

如果没有指定授权，User Operator 不会为用户置备任何访问权限。此类 KafkaUser 仍然可以访问资源取决于所使用的授权器。例如，对于 简单 授权，这由 Kafka 集群中的 allow.everyone.if.no.acl.found 配置决定。

15.2.3.1. ACL 规则

简单 授权使用 ACL 规则来管理 Kafka 代理的访问。

ACL 规则授予您在 acls 属性中指定的用户访问权限。

有关 AclRule 对象的更多信息，请参阅 AclRule 模式参考。

15.2.3.2. 超级用户对 Kafka 代理的访问

如果用户被添加到 Kafka 代理配置中的超级用户列表中，无论 KafkaUser 中的 ACL 中定义的任何授权限制是什么，用户都可以无限访问集群。

有关配置超级用户访问代理的更多信息，请参阅 Kafka 授权。

15.2.3.3. 用户配额

您可以为 KafkaUser 资源配置 spec 以强制配额，以便用户不会超过配置对 Kafka 代理的访问级别。

您可以设置基于大小的网络使用量和基于时间的 CPU 使用率阈值。您还可以添加分区修改配额来控制更改分区请求的速率。

带有用户配额的 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) 用户可推送到 Kafka 代理的数据量的字节每秒配额

(2) 用户可以从 Kafka 代理获取的数据量上的字节每秒配额

(3) CPU 使用率限制作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

有关这些属性的更多信息，请参阅 KafkaUserQuotas 模式参考。

15.3. 保护对 KAFKA 代理的访问

要建立对 Kafka 代理的安全访问，请配置并应用：

- 一个 Kafka 资源：
  - 使用指定验证类型创建监听程序
  - 为整个 Kafka 集群配置授权
- 通过监听程序安全地访问 Kafka 代理的 KafkaUser 资源

配置 Kafka 资源来设置：

- 侦听器身份验证
- 限制访问 Kafka 监听器的网络策略
- Kafka 授权
- 超级用户对代理的访问没有限制

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。

您可以通过安装自己的证书来替换 Cluster Operator 生成的证书。

您还可以为启用了 TLS 加密的任何监听程序提供自己的服务器证书和私钥。这些用户提供的证书称为 *Kafka 侦听器证书*。通过提供 Kafka 侦听器证书，您可以利用现有的安全基础架构，如机构的私有 CA 或

公共 CA。Kafka 客户端需要信任用于为监听器证书签名的 CA。在需要时，您必须手动续订 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 资源来设置：

- 与启用的监听程序身份验证匹配的身份验证
- 与启用的 Kafka 授权匹配的授权
- 客户端控制资源使用的配额

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

有关访问配置属性的更多信息，请参阅 schema 参考：

- Kafka 模式参考
- KafkaUser 模式参考
- GenericKafkaListener 模式参考

15.3.1. 保护 Kafka 代理

此流程显示在运行 Apache Kafka 的 Streams 时保护 Kafka 代理的步骤。

为 Kafka 代理实现的安全性必须与需要访问权限的客户端实现的安全兼容。

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

这些步骤显示简单授权的配置以及使用 mTLS 身份验证的监听程序。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，并将 OAuth 2.0 或 OPA 用于 Kafka 授权。

流程

1. 配置 Kafka 资源。
   a. 配置授权的 authorization 属性。
   b. 配置 listeners 属性，以创建具有身份验证的监听程序。

   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 在使用 AclAuthorizer 和 StandardAuthorizer Kafka 插件的 Kafka 代理上启用 simple 授权。

2. 对 Kafka 有无限访问权限的用户主体列表。当使用 mTLS 身份验证时，*CN* 是客户端证书的通用名称。

3. 可以为每个监听程序配置监听程序身份验证机制，并指定为 mTLS、SCRAM-SHA-512 或基于令牌的 OAuth 2.0。

如果要配置外部监听程序，则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_configuration_file>
```

Kafka 集群使用 mTLS 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。

创建一个服务作为与 Kafka 集群连接的 *bootstrap 地址*。

在 secret <cluster *_name*> *-cluster*-ca-cert 中还创建了用于验证 kafka 代理的身份的集群 CA 证书。

15.3.2. 保护用户对 Kafka 的访问

创建或修改 KafkaUser 以代表需要对 Kafka 集群进行安全访问的客户端。

当您配置 KafkaUser 身份验证和授权机制时，请确保它们与对应的 Kafka 配置匹配：

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

此流程演示了如何使用 mTLS 身份验证创建用户。您还可以创建带有 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代理的身份验证设置。例如，如果在 Kafka 配置中也未启用，则无法验证带有 mTLS 的用户。

先决条件

- 正在运行的 Kafka 集群使用 mTLS 身份验证和 TLS 加密配置 Kafka 代理监听程序。
- 正在运行的用户 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
        operations:
          - Describe
          - Read
      - resource:
          type: group
          name: my-group
          patternType: literal
        operations:
          - Read
```

1 用户身份验证机制，定义为 mutual tls 或 scram-sha-512。

2 简单的授权，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>
```

创建了用户，以及名称与 KafkaUser 资源相同的 Secret。Secret 包含 mTLS 身份验证的私钥和公钥。

有关配置带有与 Kafka 代理安全连接的属性的 Kafka 客户端的详情，请参考 第 14.4 节 “使用监听程序设置对 Kafka 集群的客户端访问”。

15.3.3. 使用网络策略限制 Kafka 侦听程序的访问

您可以使用 networkPolicyPeers 属性限制对监听程序的访问。

先决条件

- 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 tls 侦听器配置为只允许来自应用程序 pod 的连接，标签 app 设置为 kafka-client：

```
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
```

其他资源

- networkPolicyPeers 配置
- NetworkPolicyPeer API 参考

15.3.4. 为 TLS 加密提供自己的 Kafka 侦听器证书

监听器提供对 Kafka 代理的客户端访问。在 Kafka 资源中配置监听程序，包括使用 TLS 进行客户端访问所需的配置。

默认情况下，监听程序使用由 Apache Kafka 的 Streams 生成的内部 CA （证书颁发机构）证书签名的证书。Cluster Operator 在创建 Kafka 集群时生成 CA 证书。当您为 TLS 配置客户端时，您可以将 CA 证书添加到其信任存储配置中以验证 Kafka 集群。您还可以安装和使用自己的 CA 证书。或者，您可以使用 brokerCertChainAndKey 属性配置监听程序，并使用自定义服务器证书。

brokerCertChainAndKey 属性允许您使用在监听器级别自己的自定义证书访问 Kafka 代理。您可以使用自己的私钥和服务器证书创建 secret，然后在监听器的 brokerCertChainAndKey 配置中指定密钥和证书。您可以使用由公共（外部）CA 或私有 CA 签名的证书。如果由公共 CA 签名，通常不需要将其添加到客户端的信任存储配置中。自定义证书不由 Apache Kafka 的 Streams 管理，因此您需要手动更新它们。

注意

侦听器证书仅用于 TLS 加密和服务器身份验证。它们不用于 TLS 客户端身份验证。如果还想将自己的证书用于 TLS 客户端身份验证，您必须安装和使用自己的客户端 CA。

先决条件

- Cluster Operator 正在运行。
- 每个监听程序都需要以下内容：
  - 由外部 CA 签名的兼容服务器证书。（提供 PEM 格式的 X.509 证书。）

    您可以将一个监听程序证书用于多个监听程序。
  - 主题备用名称(SAN)在每个监听器的证书中指定。更多信息请参阅 第 15.3.5 节 “Kafka

监听器的服务器证书中的替代主题”。

如果您不使用自签名证书，您可以提供一个证书中包含整个 CA 链的证书。

如果为监听程序配置了 TLS 加密(tls: true)，则只能使用 brokerCertChainAndKey 属性。

注意

Apache Kafka 的流不支持将加密私钥用于 TLS。存储在 secret 中的私钥必须未加密的才能正常工作。

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。

将监听程序配置为在 configuration.brokerCertChainAndKey 属性中使用您的 Secret、证书文件和私钥文件。

启用 TLS 加密的 loadbalancer 外部监听程序配置示例

```
# ...
listeners:
  - name: plain
    port: 9092
    type: internal
    tls: false
  - name: external3
    port: 9094
    type: loadbalancer
    tls: true
    configuration:
      brokerCertChainAndKey:
        secretName: my-secret
```

```
      certificate: my-listener-certificate.crt
      key: my-listener-key.key
# ...
```

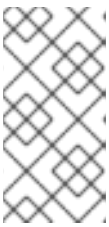
TLS 侦听器配置示例

```
# ...
listeners:
  - name: plain
    port: 9092
    type: internal
    tls: false
  - name: tls
    port: 9093
    type: internal
    tls: true
    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 启动 Kafka 集群的滚动更新，该集群更新监听程序的配置。



注意

如果您在被监听程序使用的 Secret 中更新 Kafka 侦听器证书，也会启动滚动更新。

15.3.5. Kafka 监听器的服务器证书中的替代主题

要将 TLS 主机名验证与您自己的 Kafka 侦听器证书 搭配使用，您必须为每个监听程序使用正确的主

题备用名称(SAN)。证书 SAN 必须为以下内容指定主机名：

- 集群中的所有 Kafka 代理
- Kafka 集群 bootstrap 服务

如果您的 CA 支持，您可以使用通配符证书。

15.3.5.1. 内部监听程序的 SAN 示例

使用以下示例帮助您为内部监听程序在证书中指定 SAN 的主机名。

将 <cluster-name > 替换为 Kafka 集群的名称，将 <namespace > 替换为运行集群的 OpenShift 命名空间。

类型的通配符示例：内部监听程序

```
//Kafka brokers
*.<cluster-name>-kafka-brokers
*.<cluster-name>-kafka-brokers.<namespace>.svc

// Bootstrap service
<cluster-name>-kafka-bootstrap
<cluster-name>-kafka-bootstrap.<namespace>.svc
```

类型：内部监听程序 的非通配符示例

```
// 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
```

类型的非通配符示例：cluster-ip listener

```
// Kafka brokers
<cluster-name>-kafka-<listener-name>-0
<cluster-name>-kafka-<listener-name>-0.<namespace>.svc
<cluster-name>-kafka-<listener-name>-1
<cluster-name>-kafka-<listener-name>-1.<namespace>.svc
# ...

// Bootstrap service
<cluster-name>-kafka-<listener-name>-bootstrap
<cluster-name>-kafka-<listener-name>-bootstrap.<namespace>.svc
```

15.3.5.2. 外部监听程序的 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 15.1. 每个类型的外部监听程序的 sans

外部监听程序类型	在 SANs 中，指定...
Ingress	所有 Kafka 代理 **Ingress** 资源的地址以及 bootstrap **Ingress** 的地址。 您可以使用匹配的通配符名称。
route	所有 Kafka 代理 **路由和** bootstrap 路由地址 **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**。 您可以使用匹配的通配符名称。

外部监听程序类型	在 SANs 中，指定...
nodeport	Kafka 代理 pod 可能调度到的所有 OpenShift worker 节点的地址。 您可以使用匹配的通配符名称。

其他资源

- 第 15.3.4 节 “为 TLS 加密提供自己的 Kafka 侦听器证书”

15.4. 使用基于 OAUTH 2.0 令牌的身份验证

Apache Kafka 的流支持使用 *OAUTHBEARER* 和 *PLAIN* 机制使用 OAuth 2.0 身份验证。

OAuth 2.0 启用应用程序之间的基于令牌的标准化身份验证和授权，使用中央授权服务器签发对资源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。

Kafka 代理和客户端都需要配置为使用 OAuth 2.0。OAuth 2.0 身份验证也可以与基于 simple 或 OPA 的 Kafka 授权 一起使用。

使用基于 OAuth 2.0 令牌的身份验证时，应用程序客户端可以在不公开帐户凭据的情况下访问应用服务器（称为 *资源服务器*）上的资源。

应用程序客户端通过访问令牌作为身份验证方法传递，应用服务器也可以用来决定要授予的访问权限级别。授权服务器处理访问权限的授予和查询有关访问权限的查询。

在 Apache Kafka 的 Streams 中：

- Kafka 代理作为 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 客户端在 Kafka 代理验证。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 Apache Kafka 的 Streams 部署，OAuth 2.0 集成提供：

- Kafka 代理的服务器端 OAuth 2.0 支持
- Kafka MirrorMaker、Kafka Connect 和 Kafka Bridge 的客户端 OAuth 2.0 支持

15.4.1. OAuth 2.0 身份验证机制

Apache Kafka 的流支持 OAUTHBEARER 和 PLAIN 机制进行 OAuth 2.0 身份验证。这两种机制都允许 Kafka 客户端与 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因每种机制而异。

我们建议您将客户端配置为尽可能使用 OAUTHBEARER。OAUTHBEARER 提供比 PLAIN 更高的安全性，因为客户端凭证*不会*与 Kafka 代理共享。考虑仅在不支持 OAUTHBEARER 的 Kafka 客户端中使用 PLAIN。

您可以将 Kafka 代理监听程序配置为使用 OAuth 2.0 身份验证来连接客户端。如果需要，您可以在同一 oauth 侦听器上使用 OAUTHBEARER 和 PLAIN 机制。支持每个机制的属性必须在 oauth 侦听器配置中明确指定。

OAUTHBEARER 概述

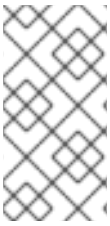
OAUTHBEARER 在 oauth 侦听器配置中自动启用用于 Kafka 代理。您可以将 enableOauthBearer 属性设置为 true，但这不是强制要求。

```
# ...
authentication:
  type: oauth
  # ...
  enableOauthBearer: true
```

许多 Kafka 客户端工具使用在协议级别为 OAUTHBEARER 提供基本支持的库。为了支持应用程序开发，Apache Kafka 的 Streams 为上游 Kafka 客户端 Java 库（但不适用于其他库）提供了一个 *OAuth 回调处理器*。因此，您不需要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。使用 Go 等其他语言编写的客户端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，客户端发起带有 Kafka 代理进行凭证交换的会话，其中凭证采用由回调处理器提供的 bearer 令牌的形式。使用回调，您可以以三种方式之一配置令牌置备：

- 客户端 ID 和 Secret （使用 *OAuth 2.0 客户端凭证* 机制）
- 一个长期的访问令牌，在配置时手动获取
- 一个长期的刷新令牌，在配置时手动获取



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

要使用 PLAIN，您必须在 oauth 侦听器配置中为 Kafka 代理启用它。

在以下示例中，除了 OAUTHBEARER 外，PLAIN 会被启用，这默认是启用的。如果只使用 PLAIN，您可以通过将 enableOauthBearer 设置为 false 来禁用 OAUTHBEARER。

```
# ...
authentication:
  type: oauth
  # ...
  enablePlain: true
  tokenEndpointUri: https://OAUTH-SERVER-ADDRESS/auth/realms/external/protocol/openid-connect/token
```

PLAIN 是所有 Kafka 客户端工具使用的简单身份验证机制。要启用 PLAIN 用于 OAuth 2.0 身份验证，Apache Kafka 的流 *通过 PLAIN 服务器端回调提供 OAuth 2.0*。

随着 Apache Kafka 的 Streams 实现的 PLAIN，客户端凭证不会存储在 ZooKeeper 中。相反，客户端凭证会在兼容授权服务器后进行集中处理，这与使用 OAUTHBEARER 身份验证类似。

当与 OAuth 2.0 over PLAIN 回调一起使用时，Kafka 客户端使用以下方法之一与 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）
- 一个长期的访问令牌，在配置时手动获取

对于这两种方法，客户端必须提供 PLAIN username 和 password 属性，将凭证传递给 Kafka 代理。客户端使用这些属性传递客户端 ID 和 secret 或用户名和访问令牌。

客户端 ID 和 secret 用于获取访问令牌。

访问令牌作为 password 属性值传递。您可以使用或没有 $accessToken: 前缀来传递访问令牌。

- 如果您在监听器配置中配置令牌端点(tokenEndpointUri)，则需要前缀。
- 如果您没有在监听器配置中配置令牌端点(tokenEndpointUri)，则不需要前缀。Kafka 代理将密码解释为原始访问令牌。

如果将密码设置为访问令牌，则必须将用户名设置为 Kafka 代理从访问令牌获取的相同的主体名称。您可以使用 userNameClaim,fallbackUserNameClaim,fallbackUsernamePrefix, 和 userInfoEndpointUri 属性在监听器中指定用户名提取选项。用户名提取过程还取决于您的授权服务器；特别是，它将客户端 ID 映射到帐户名称。

注意

OAuth over PLAIN 不支持 密码授权机制。您只能通过 SASL PLAIN 机制 'proxy' 来包括客户端凭证 (clientId + secret)或访问令牌，如上所述。

其他资源

- 第 15.4.6.2 节 “为 Kafka 代理配置 OAuth 2.0 支持”

15.4.2. 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

与授权服务器的关系，Kafka 代理和 Kafka 客户端都被视为 OAuth 2.0 客户端。

15.4.2.1. 授权服务器上的 OAuth 2.0 客户端配置

要配置 Kafka 代理以验证会话启动期间收到的令牌，建议的做法是在授权服务器中创建一个 OAuth 2.0 *client* 定义（配置为 *confidential*），并启用了以下客户端凭证：

- 客户端 ID kafka （例如）
- 客户端 ID 和 Secret 作为身份验证机制

注意

在使用授权服务器的非公共内省端点时，您只需要使用客户端 ID 和 secret。在使用公共授权服务器端点时，通常不需要凭据，如快速本地 JWT 令牌验证一样。

15.4.2.2. 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，使用验证方法 oauth 指定 Kafka 集群自定义资源的 tls 侦听器配置：

评估 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
    #...
```

您可以在监听器中配置 OAuth 2.0 身份验证。我们建议将 OAuth 2.0 身份验证与 TLS 加密一起使用 (tls: true)。如果没有加密，连接容易通过令牌保护和未经授权的访问。

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。

使用带有外部监听程序的 OAuth 2.0

```
# ...
listeners:
  - name: external3
    port: 9094
    type: loadbalancer
    tls: true
    authentication:
      type: oauth
    #...
```

tls 属性默认为 *false*，因此必须启用它。

当您将验证类型定义为 OAuth 2.0 时，您可以根据验证类型添加配置，可以是 fast local JWT validation 或 token validation using an introspection endpoint。

为监听器配置 OAuth 2.0 的步骤带有描述和示例，请参考为 Kafka 代理配置 OAuth 2.0 支持。

15.4.2.3. 快速的本地 JWT 令牌验证配置

快速本地 JWT 令牌验证会在本地检查 JWT 令牌签名。

本地检查可确保令牌：

- 使用一个访问令牌的 Bearer 的(*typ*)声明值符合类型
- 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 validIssuerURI 属性，以便任何未由授权服务器发布的令牌都会被拒绝。

授权服务器不需要在快速本地 JWT 令牌验证过程中联系。您可以通过指定 jwksEndpointUri 属性（由 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。端点包含验证已签名的 JWT 令牌的公钥，这些令牌由 Kafka 客户端作为凭证发送。

注意

所有与授权服务器通信都应使用 TLS 加密来执行。

您可以将证书信任存储配置为 Apache Kafka 项目命名空间的 Streams 中的 OpenShift Secret，并使用 tlsTrustedCertificates 属性指向包含信任存储文件的 OpenShift Secret。

您可能希望配置 userNameClaim 以正确提取 JWT 令牌中的用户名。如果需要，您可以使用 JsonPath 表达式，如 "['user.info'].['user.id']"，从令牌中的嵌套 JSON 属性检索用户名。

如果要使用 Kafka ACL 授权，则需要在身份验证过程中使用用户名来识别用户。(JWT 令牌中的 子 声明通常是唯一 ID，而不是用户名。)

快速本地 JWT 令牌验证配置示例

```
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
    #...
```

15.4.2.4. OAuth 2.0 内省端点配置

使用 OAuth 2.0 内省端点进行令牌验证会将接收的访问令牌视为不透明。Kafka 代理向内省端点发送访问令牌，该端点使用验证所需的令牌信息做出响应。最重要的是，如果特定访问令牌有效，它会返回最新的信息，以及令牌何时过期的信息。

要配置基于 OAuth 2.0 内省的验证，您可以指定一个 introspectionEndpointUri 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，通常必须指定 clientId 和 clientSecret，因为内省端点通常受到保护。

内省端点配置示例

```
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
```

15.4.3. Kafka 代理的会话重新身份验证

您可以将 oauth 监听程序配置为对 Kafka 客户端和 Kafka 代理之间的 OAuth 2.0 会话使用 Kafka *会话重新身份验证*。这个机制在定义的时间后强制实施客户端和代理之间经过身份验证的会话的过期。当会话过期时，客户端会立即通过重复使用现有连接而不是丢弃它来启动新的会话。

会话重新身份验证默认为禁用。要启用它，您可以在 oauth 侦听器配置中为 maxSecondsWithoutReauthentication 设置时间值。相同的属性用于为 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 15.4.6.2 节 “为 Kafka 代理配置 OAuth 2.0 支持”。

会话重新身份验证必须由客户端使用的 Kafka 客户端库支持。

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

当代理的经过身份验证的会话过期时，客户端必须通过向代理发送一个新的有效访问令牌来重新验证到现有会话，而无需丢弃连接。

如果令牌验证成功，则使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。如果代理上启用了重新身份验证机制，则使用 Kafka 客户端库 2.2 或更高版本的 Java 客户端会自动重新验证。

如果使用，会话重新身份验证也适用于刷新令牌。当会话过期时，客户端会使用其刷新令牌来刷新访问令牌。然后，客户端使用新的访问令牌重新验证现有会话。

OAUTHBEARER 和 PLAIN 的会话到期

配置会话重新身份验证后，会话到期对于 OAUTHBEARER 和 PLAIN 身份验证是不同的。

对于 OAUTHBEARER 和 PLAIN，使用客户端 ID 和 secret 方法：

- 代理的经过身份验证的用户会话将在配置的 maxSeconds withoutReauthentication 下过期。
- 如果访问令牌在配置的时间前过期，会话将提前过期。

对于 PLAIN，使用长期访问令牌方法：

- 代理的经过身份验证的用户会话将在配置的 maxSeconds withoutReauthentication 下过期。
- 如果访问令牌在配置的时间前过期，则重新身份验证将失败。虽然尝试尝试会话重新身份验证，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连接到代理，而无需重新验证。经过身份验证的会话不会因为访问令牌到期而终止。但是，这可在配置授权时考虑，例如，使用 keycloak 授权或安装自定义授权器。

其他资源

- 第 15.4.2 节 “OAuth 2.0 Kafka 代理配置”
- 第 15.4.6.2 节 “为 Kafka 代理配置 OAuth 2.0 支持”
- KafkaListenerAuthenticationOAuth 模式参考

- KIP-368

15.4.4. OAuth 2.0 Kafka 客户端配置

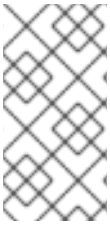
Kafka 客户端被配置为：

- 从授权服务器获取有效访问令牌（客户端 ID 和 Secret）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的凭证从不会发送到代理。

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 Secret 进行身份验证。使用长期的访问令牌或长期的刷新令牌会增加复杂性，因为对授权服务器工具还有额外的依赖。



注意

如果您使用长期的访问令牌，您可能需要在授权服务器中配置客户端来提高令牌的最大生命周期。

如果 Kafka 客户端没有直接配置访问令牌，客户端会在 Kafka 会话发起授权服务器的过程中交换访问令牌的凭证。Kafka 客户端交换：

- 客户端 ID 和 Secret
- 客户端 ID、刷新令牌和（可选）secret
- 使用客户端 ID 和（可选）secret 的用户名和密码

15.4.5. OAuth 2.0 客户端身份验证流

OAuth 2.0 身份验证流程取决于底层 Kafka 客户端和 Kafka 代理配置。流还必须由使用的授权服务器支持。

Kafka 代理监听程序配置决定客户端如何使用访问令牌进行身份验证。客户端可以传递客户端 ID 和机密来请求访问令牌。

如果侦听器配置为使用 PLAIN 身份验证，客户端可以通过客户端 ID 和 secret 或用户名和访问令牌进行身份验证。这些值作为 PLAIN 机制 username 和 password 属性传递。

侦听器配置支持以下令牌验证选项：

- 您可以根据 JWT 签名检查和本地令牌内省使用快速本地令牌验证，而无需联系授权服务器。授权服务器提供带有公共证书的 JWKS 端点，用于验证令牌中的签名。
- 您可以使用调用授权服务器提供的令牌内省端点。每次建立新的 Kafka 代理连接时，代理会将从客户端接收的访问令牌传递给授权服务器。Kafka 代理检查响应，以确认令牌是否有效。

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。

也可以为以下类型的身份验证配置 Kafka 客户端凭证：

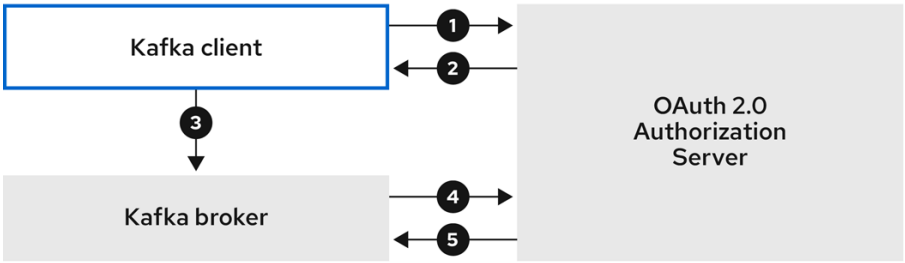
- 使用之前生成的长期访问令牌直接进行本地访问
- 与授权服务器联系，以获取要发布的新访问令牌（使用客户端 ID 和 secret，或刷新令牌，或者用户名和密码）

15.4.5.1. 使用 SASL OAUTHBEARER 机制的客户端身份验证流示例

您可以使用 SASL OAUTHBEARER 机制为 Kafka 身份验证使用以下通信流。

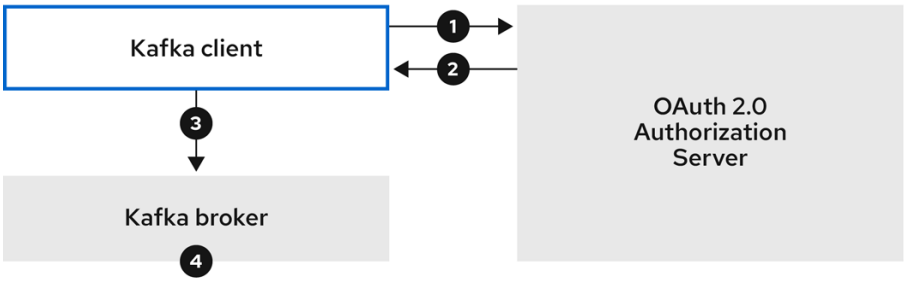
- 使用客户端 ID 和 secret 的客户端以及代理委派到授权服务器的验证
- 使用客户端 ID 和 secret 的客户端，代理执行快速本地令牌验证
- 使用长期访问令牌的客户端，带有代理委派验证到授权服务器
- 使用长期访问令牌的客户端，代理执行快速本地验证

使用客户端 ID 和 secret 的客户端以及代理委派到授权服务器的验证



1. Kafka 客户端使用客户端 ID 和 secret 从授权服务器请求访问令牌，以及可选的刷新令牌。或者，客户端也可以使用用户名和密码进行身份验证。
2. 授权服务器生成新的访问令牌。
3. Kafka 客户端使用 SASL OAUTHBEARER 机制通过 Kafka 代理进行身份验证，以传递访问令牌。
4. Kafka 代理通过使用自己的客户端 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，则会建立 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



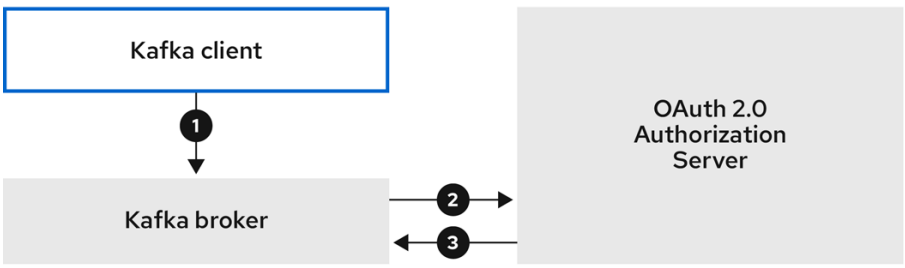
1. Kafka 客户端使用令牌端点、使用客户端 ID 和 secret 以及刷新令牌（可选）从令牌端点验证。或者，客户端也可以使用用户名和密码进行身份验证。

2. 授权服务器生成新的访问令牌。

3. Kafka 客户端使用 SASL OAUTHBEARER 机制通过 Kafka 代理进行身份验证，以传递访问令牌。

4. Kafka 代理使用 JWT 令牌签名检查和本地令牌内省验证访问令牌。

使用长期访问令牌的客户端，带有代理委派验证到授权服务器

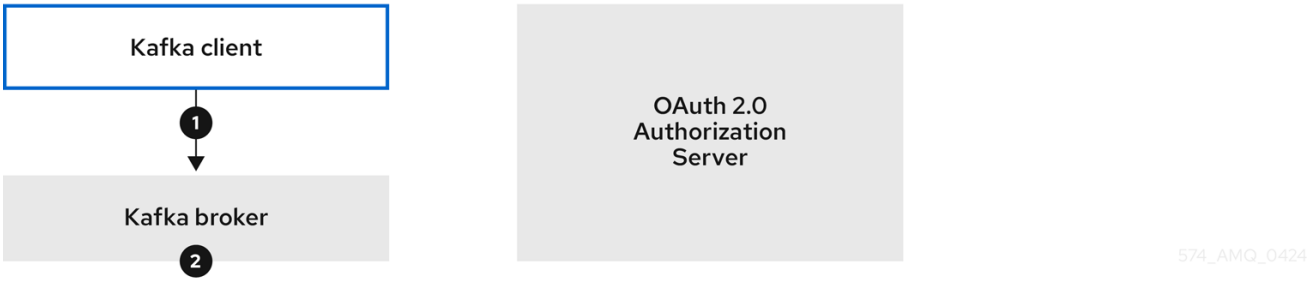


1. Kafka 客户端使用 SASL OAUTHBEARER 机制通过 Kafka 代理进行身份验证，以传递长期访问令牌。

2. Kafka 代理通过使用自己的客户端 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，则会建立 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 机制通过 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验证访问令牌。

警告

快速的本地 JWT 令牌签名验证仅适用于短期的令牌，因为如果已撤销令牌，就不会通过授权服务器检查该授权服务器。令牌到期时间写入到令牌，但可以随时进行撤销，因此不能在不联系授权服务器的情况下被考虑。任何发布的令牌都将被视为有效，直到过期为止。

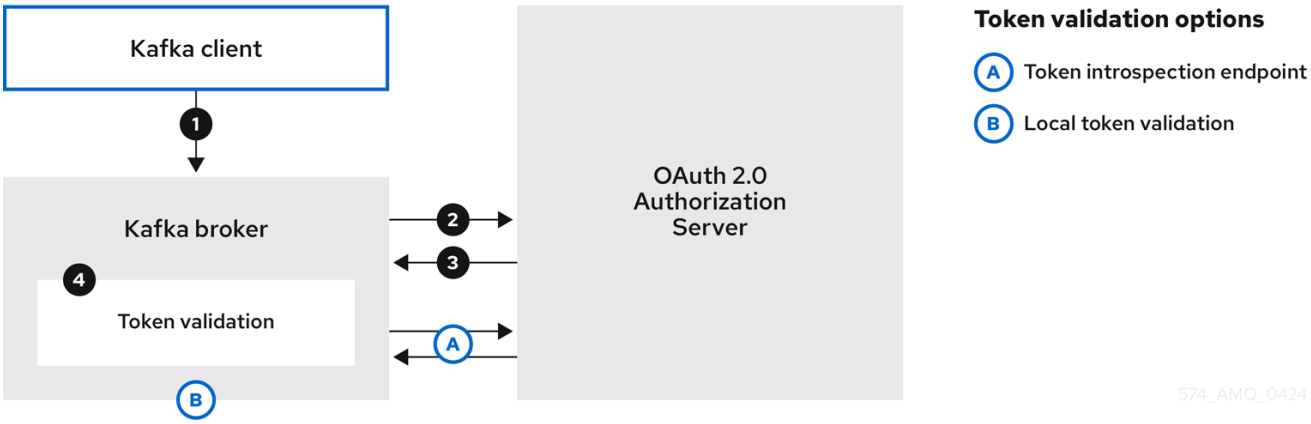
15.4.5.2. 使用 SASL PLAIN 机制的客户端身份验证流示例

您可以使用 OAuth PLAIN 机制对 Kafka 身份验证使用以下通信流。

- 使用客户端 ID 和 secret 的客户端以及代理获取客户端的访问令牌

- 使用没有客户端 ID 和 secret 的长期访问令牌的客户端

使用客户端 ID 和 secret 的客户端以及代理获取客户端的访问令牌



1. Kafka 客户端或传递一个 clientId 作为用户名，以及一个 secret 作为密码。

2. Kafka 代理使用令牌端点将 clientId 和 secret 传递给授权服务器。

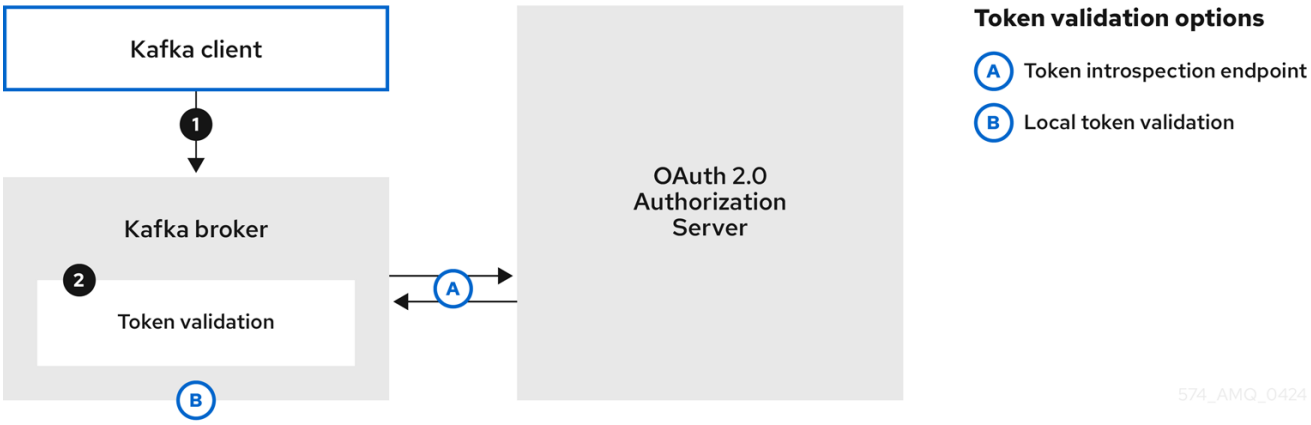
3. 如果客户端凭据无效，授权服务器会返回一个新的访问令牌或错误。

4. Kafka 代理使用以下方法之一验证令牌：

   a. 如果指定了令牌内省端点，Kafka 代理会通过调用授权服务器上的端点来验证访问令牌。如果令牌验证成功，则会建立会话。

   b. 如果使用本地令牌内省，则不会向授权服务器发出请求。Kafka 代理使用 JWT 令牌签名检查在本地验证访问令牌。

使用没有客户端 ID 和 secret 的长期访问令牌的客户端



1. Kafka 客户端会传递用户名和密码。密码提供在运行客户端前手动配置的访问令牌值。

2. 密码通过或不使用 $accessToken: 字符串前缀来传递，具体取决于 Kafka 代理侦听程序是否配置了令牌端点来进行身份验证。

   a. 如果配置了令牌端点，则密码应加上前缀 $accessToken:，以便代理知道 password 参数包含访问令牌，而不是客户端 secret。Kafka 代理将用户名解释为帐户用户名。

   b. 如果没有在 Kafka 代理监听程序上配置令牌端点（强制 no-client-credentials 模式），则密码应在没有前缀的情况下提供访问令牌。Kafka 代理将用户名解释为帐户用户名。在这个模式中，客户端不使用客户端 ID 和 secret，password 参数始终解释为原始访问令牌。

3. Kafka 代理使用以下方法之一验证令牌：

   a. 如果指定了令牌内省端点，Kafka 代理会通过调用授权服务器上的端点来验证访问令牌。如果令牌验证成功，则会建立会话。

   b. 如果使用本地令牌内省，则不会向授权服务器发出请求。Kafka 代理使用 JWT 令牌签名检查在本地验证访问令牌。

15.4.6. 配置 OAuth 2.0 身份验证

OAuth 2.0 用于 Kafka 客户端和流用于 Apache Kafka 组件之间的交互。

要将 OAuth 2.0 用于 Apache Kafka，您必须：

1. 部署授权服务器并配置部署，使其与 Apache Kafka 的 Streams 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

15.4.6.1. 配置 OAuth 2.0 授权服务器

这个步骤描述了在一般情况下，您需要配置授权服务器以与 Apache Kafka 的 Streams 集成。

这些说明不特定于产品。

这些步骤取决于所选的授权服务器。有关如何设置 OAuth 2.0 访问的信息，请参阅产品文档的授权服务器。

注意

如果您已经部署了授权服务器，您可以跳过部署步骤并使用您的当前部署。

流程

1. 将授权服务器部署到集群中。
2. 访问授权服务器的 CLI 或管理控制台，为 Apache Kafka 配置 OAuth 2.0。

现在，准备授权服务器以用于 Apache Kafka 的流。

3. 配置 kafka-broker 客户端。

4. 为应用程序的每个 Kafka 客户端组件配置客户端。

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。

15.4.6.2. 为 Kafka 代理配置 OAuth 2.0 支持

此流程描述了如何配置 Kafka 代理，以便代理监听程序可以使用授权服务器使用 OAuth 2.0 身份验证。

我们建议通过一个带有 tls: true 的监听程序在加密接口上使用 OAuth 2.0。不建议使用 plain 监听程序。

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，您可能需要使用正确的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，有两个选项用于在新连接的 Kafka 客户端的 OAuth 2.0 身份验证过程中验证访问令牌：

- 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
- OAuth 2.0 身份验证机制

先决条件

- Apache Kafka 和 Kafka 的流正在运行
- 部署 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必相同，因为它们相互独立。

   此处的示例显示了为外部监听器配置的配置选项。

示例 1：配置快速本地 JWT 令牌验证

```
#...
- name: external3
  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 证书端点的 URI。

4

包含用于识别用户的实际用户名的令牌声明（或密钥）。其值取决于授权服务器。如有必要，可以使用类似 "['user.info'].['user.id']" 的 JsonPath 表达式，从令牌内的嵌套 JSON 属性检索用户名。

5

（可选）激活 Kafka 重新验证机制，该机制强制会话到期时间与访问令牌相同的长度。如果指定的值小于访问令牌保留的时间，则客户端必须在实际令牌到期前重新验证。默认情况下，当访问令牌过期时，会话不会过期，客户端也不会尝试重新身份验证。

6

（可选）与授权服务器进行 TLS 连接的可信证书。

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。默认为 360 秒。如果您指定了较长的时间，请考虑允许访问撤销的证书的风险。

9

JWKS 证书刷新之间的周期。间隔必须至少超过到期间隔的 60 秒。默认值为 300 秒。

10

示例 2：使用内省端点配置令牌验证

```
- name: external3
  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 Secret 和客户端 ID 用于验证。

(4) 包含用于识别用户的实际用户名的令牌声明（或密钥）。其值取决于授权服务器。如有必要，可以使用类似 "['user.info'].['user.id']" 的 JsonPath 表达式，从令牌内的嵌套 JSON 属性检索用户名。

(5) （可选）激活 Kafka 重新验证机制，该机制强制会话到期时间与访问令牌相同的长度。如果指定的值小于访问令牌保留的时间，则客户端必须在实际令牌到期前重新验证。默认情况下，当访问令牌过期时，会话不会过期，客户端也不会尝试重新身份验证。

根据您如何应用 OAuth 2.0 身份验证和授权服务器类型，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://<auth_server_address>/auth/realms/external/protocol/openid-connect/userinfo
(6)
  enableOauthBearer: false (7)
  enablePlain: true (8)
  tokenEndpointUri:
https://<auth_server_address>/auth/realms/external/protocol/openid-connect/token
(9)
  customClaimCheck: "@.custom == 'custom-value'" (10)
  clientAudience: audience (11)
  clientScope: scope (12)
  connectTimeoutSeconds: 60 (13)
  readTimeoutSeconds: 60 (14)
  httpRetries: 2 (15)
  httpRetryPauseMs: 300 (16)
  groupsClaim: "$.groups" (17)
  groupsClaimDelimiter: "," (18)
  includeAcceptHeader: false (19)
```

(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，将 checkIssuer 设置为 false，且不指定 validIssuerUri。默认为 true。

(2)

如果您的授权服务器提供 aud (audience)声明，并且希望强制进行受众检查，请将 checkAudience 设置为 true。Audience 检查标识令牌的预期接收者。因此，Kafka 代理将拒绝在其 aud 声明中没有 clientId 的令牌。默认为 false。

(3)

授权服务器可能无法提供单个属性来识别常规用户和客户端。当客户端以自己的名称进行身份验证时，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来

获取刷新令牌或访问令牌时，除了客户端 ID 外，服务器可能会提供一个 *username* 属性。如果主用户 ID 属性不可用，则使用这个 fallback 选项指定要使用的用户名声明(attribute)。如有必要，可以使用 JsonPath 表达式（如 "['client.info'].['client.id']" 来检索回退用户名，以从令牌内嵌套的 JSON 属性检索用户名。

4

在适用 fallbackUserNameClaim 时，可能还需要防止用户名声明的值和回退用户名声明之间的名称冲突。请考虑存在名为 producer 的客户端存在的情况，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 introspectionEndpointUri时适用）取决于您使用的授权服务器，内省端点可能会或不返回 *令牌类型* 属性，或者可以包含不同的值。您可以指定来自内省端点的响应必须包含有效的令牌类型值。

6

（仅在使用 introspectionEndpointUri时适用）授权服务器可以配置或实施，以便在 Introspection Endpoint 响应中提供任何可识别的信息。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

把它设置为 false，以禁用侦听器上的 OAUTHBEARER 机制。至少需要启用 PLAIN 或 OAUTHBEARER 之一。默认为 true。

8

设置为 true，以便在侦听器上启用 PLAIN 身份验证，该监听程序支持所有平台上的客户端。

9

PLAIN 机制的其他配置。如果指定，客户端可以通过在使用 $accessToken: 前缀将访问令牌 作为密码 传递来通过 PLAIN 进行身份验证。对于生产环境，请始终使用 https:// urls。

10

可以通过将其设置为 JsonPath 过滤器查询，在验证过程中对 JWT 访问令牌实施其他自定义规则。如果访问令牌不包含必要的数据，它将被拒绝。使用 introspectionEndpointUri 时，自定义检查将应用到内省端点响应 JSON。

11

12

传递给令牌端点的 scope 参数。获取访问令牌进行代理身份验证时使用的 *scope*。它还在使用 clientId 和 secret 的 PLAIN 客户端验证中用于 OAuth 2.0 的客户端名称。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。它不会影响监听器的令牌验证规则。

13

连接到授权服务器时的连接超时（以秒为单位）。默认值为 60。

14

连接到授权服务器时读取超时（以秒为单位）。默认值为 60。

15

将失败的 HTTP 请求重试到授权服务器的次数上限。默认值为 0， 表示不会执行重试。要有效地使用这个选项，请考虑减少 connectTimeoutSeconds 和 readTimeoutSeconds 选项的超时时间。但请注意，重试可能会阻止当前 worker 线程可用于其他请求，如果太多请求停滞，则可能会导致 Kafka 代理无响应。

16

尝试另一个到授权服务器的 HTTP 请求重试前等待的时间。默认情况下，这个时间被设置为零，这意味着不会应用暂停。这是因为导致失败请求的许多问题是针对每个请求的网络粘合或代理问题，可以快速解决。但是，如果您的授权服务器处于压力下或高流量，您可能希望将此选项设置为值 100 ms 或更多，以减少服务器上的负载，并增加成功重试的可能性。

17

用于从 JWT 令牌或内省端点响应中提取组信息的 JsonPath 查询。默认不设置这个选项。通过配置此选项，自定义授权器可以根据用户组做出授权决策。

18

当作为单一分隔的字符串返回时，用于解析组信息的分隔符。默认值为 ','(comma)。

19

有些授权服务器在客户端发送 Accept: application/json 标头时遇到问题。通过设置 includeAcceptHeader: false，不会发送标头。默认为 true。

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，或通过监视 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

- 将您的 Kafka 客户端配置为使用 OAuth 2.0

15.4.6.3. 将 Kafka Java 客户端配置为使用 OAuth 2.0

配置 Kafka producer 和消费者 API，以使用 OAuth 2.0 与 Kafka 代理交互。在客户端 pom.xml 文件中添加回调插件，然后为 OAuth 2.0 配置您的客户端。

在客户端配置中指定以下内容：

- SASL （简单身份验证和安全层）安全协议：
  - SASL_SSL 用于通过 TLS 加密连接进行身份验证
  - SASL_PLAINTEXT 用于通过未加密的连接进行身份验证

  将 SASL_SSL 用于生产环境，SASL_PLAINTEXT 仅用于本地开发。使用 SASL_SSL 时，需要额外的 ssl.truststore 配置。安全连接(https://)需要 truststore 配置到 OAuth 2.0 授权服务器。要验证 OAuth 2.0 授权服务器，请将授权服务器的 CA 证书添加到客户端配置的信任存储中。您可以使用 PEM 或 PKCS #12 格式配置信任存储。

- Kafka SASL 机制：

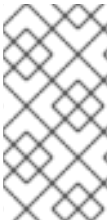
- OAUTHBEARER 用于使用 bearer 令牌交换的凭证
- PLAIN 传递客户端凭证(clientId + secret)或访问令牌

- 实现 SASL 机制的 JAAS (Java 身份验证和授权服务)模块：
  - org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule 实现 OAuthbearer 机制
  - org.apache.kafka.common.security.plain.PlainLoginModule 实现普通机制

为了可以使用 OAuthbearer 机制，还必须将自定义 io.strimzi.kafka.oauth.client.JaasClientOauthLoginCallbackHandler 类添加为回调处理程序。JaasClientOauthLoginCallbackHandler 处理 OAuth 回调到授权服务器，以便在客户端登录期间处理访问令牌。这可启用自动令牌续订，确保在用户不干预的情况下进行持续身份验证。另外，它使用 OAuth 2.0 密码授权方法为客户端处理登录凭证。

- SASL 验证方法，它支持以下验证方法：
  - OAuth 2.0 客户端凭证
  - OAuth 2.0 密码授权（已弃用）
  - 访问令牌
  - 刷新令牌

将 SASL 身份验证属性添加为 JAAS 配置 (sasl.jaas.config 和 sasl.login.callback.handler.class)。如何配置身份验证属性取决于您用来访问 OAuth 2.0 授权服务器的身份验证方法。在此过程中，属性在属性文件中指定，然后加载到客户端配置中。



注意

您还可以将身份验证属性指定为环境变量，或指定为 Java 系统属性。对于 Java 系统属性，您可以使用 setProperty 设置它们，并使用 -D 选项在命令行中传递它们。

先决条件

- Apache Kafka 和 Kafka 的流正在运行
- 部署和配置 OAuth 2.0 授权服务器以 OAuth 访问 Kafka 代理
- 为 OAuth 2.0 配置 Kafka 代理

流程

1. 将支持 OAuth 2.0 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>0.15.0.redhat-00007</version>
</dependency>
```

2. 通过在属性文件中指定以下配置来配置客户端属性：
   - 安全协议
   - SASL 机制
   - JAAS 模块和身份验证属性，具体取决于所使用的方法

   例如，我们可以将以下内容添加到 client.properties 文件中：

客户端凭证机制属性

```
security.protocol=SASL_SSL (1)
sasl.mechanism=OAUTHBEARER (2)
ssl.truststore.location=/tmp/truststore.p12 (3)
ssl.truststore.password=$STOREPASS
ssl.truststore.type=PKCS12
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLogi
nModule required \
  oauth.token.endpoint.uri="<token_endpoint_url>" \ (4)
  oauth.client.id="<client_id>" \ (5)
  oauth.client.secret="<client_secret>" \ (6)
  oauth.ssl.truststore.location="/tmp/oauth-truststore.p12" \ (7)
  oauth.ssl.truststore.password="$STOREPASS" \ (8)
  oauth.ssl.truststore.type="PKCS12" \ (9)
  oauth.scope="<scope>" \ (10)
  oauth.audience="<audience>" ; (11)
sasl.login.callback.handler.class=io.strimzi.kafka.oauth.client.JaasClientOauthLog
inCallbackHandler
```

(1) SASL_SSL 安全协议用于 TLS 加密连接。仅对本地开发使用 SASL_PLAINTEXT。

(2) 指定为 OAUTHBEARER 或 PLAIN 的 SASL 机制。

(3) 用于安全访问 Kafka 集群的 truststore 配置。

(4) 授权服务器令牌端点的 URI。

(5) 客户端 ID，这是在授权服务器中创建 *客户端*时使用的名称。

(6) 在授权服务器中创建客户端时创建的 *客户端* secret。

(7)

该位置包含授权服务器的公钥证书(truststore.p12)。

8

用于访问 truststore 的密码。

9

truststore 类型。

10

（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来指定范围。

11

（可选）从令牌端点请求令牌的听众。授权服务器可能需要客户端来指定受众。

密码授予机制属性

```
security.protocol=SASL_SSL
sasl.mechanism=OAUTHBEARER
ssl.truststore.location=/tmp/truststore.p12
ssl.truststore.password=$STOREPASS
ssl.truststore.type=PKCS12
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLogi
nModule required \
  oauth.token.endpoint.uri="<token_endpoint_url>" \
  oauth.client.id="<client_id>" \ 1
  oauth.client.secret="<client_secret>" \ 2
  oauth.password.grant.username="<username>" \ 3
  oauth.password.grant.password="<password>" \ 4
  oauth.ssl.truststore.location="/tmp/oauth-truststore.p12" \
  oauth.ssl.truststore.password="$STOREPASS" \
  oauth.ssl.truststore.type="PKCS12" \
  oauth.scope="<scope>" \
  oauth.audience="<audience>" ;
sasl.login.callback.handler.class=io.strimzi.kafka.oauth.client.JaasClientOauthLog
inCallbackHandler
```

1

2

（可选）在授权服务器中创建 *客户端*时创建的客户端 secret。

3

password 授权身份验证的用户名。OAuth 密码授权配置（用户名和密码）使用 OAuth 2.0 密码授权方法。要使用密码授权，请在您的授权服务器上为客户端创建一个有限权限的用户帐户。帐户应像服务帐户一样操作。在进行身份验证需要用户帐户的环境中使用，但首先考虑使用刷新令牌。

4

密码以授予密码身份验证。

注意

SASL PLAIN 不支持使用 OAuth 2.0 密码授权方法传递用户名和密码（密码授权）。

访问令牌属性

```
security.protocol=SASL_SSL
sasl.mechanism=OAUTHBEARER
ssl.truststore.location=/tmp/truststore.p12
ssl.truststore.password=$STOREPASS
ssl.truststore.type=PKCS12
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLogi
nModule required \
  oauth.token.endpoint.uri="<token_endpoint_url>" \
  oauth.access.token="<access_token>" \ 1
  oauth.ssl.truststore.location="/tmp/oauth-truststore.p12" \
  oauth.ssl.truststore.password="$STOREPASS" \
  oauth.ssl.truststore.type="PKCS12" ;
sasl.login.callback.handler.class=io.strimzi.kafka.oauth.client.JaasClientOauthLog
inCallbackHandler
```

1

刷新令牌属性

```
security.protocol=SASL_SSL
sasl.mechanism=OAUTHBEARER
ssl.truststore.location=/tmp/truststore.p12
ssl.truststore.password=$STOREPASS
ssl.truststore.type=PKCS12
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLogi
nModule required \
  oauth.token.endpoint.uri="<token_endpoint_url>" \
  oauth.client.id="<client_id>" \ 1
  oauth.client.secret="<client_secret>" \ 2
  oauth.refresh.token="<refresh_token>" \ 3
  oauth.ssl.truststore.location="/tmp/oauth-truststore.p12" \
  oauth.ssl.truststore.password="$STOREPASS" \
  oauth.ssl.truststore.type="PKCS12" ;
sasl.login.callback.handler.class=io.strimzi.kafka.oauth.client.JaasClientOauthLog
inCallbackHandler
```

1. 客户端 ID，这是在授权服务器中创建 *客户端*时使用的名称。
2. （可选）在授权服务器中创建 *客户端*时创建的客户端 secret。
3. Kafka 客户端长期刷新令牌。

3. 在 Java 客户端代码中输入 OAUTH 2.0 身份验证的客户端属性。

显示客户端属性输入示例

```
Properties props = new Properties();
try (FileReader reader = new FileReader("client.properties", StandardCharsets.UTF_8))
```

```
{
  props.load(reader);
}
```

4. 验证 Kafka 客户端是否可以访问 Kafka 代理。

15.4.6.4. 为 Kafka 组件配置 OAuth 2.0

此流程描述了如何将 Kafka 组件配置为使用授权服务器使用 OAuth 2.0 身份验证。

您可以为以下配置身份验证：

- 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。schema 引用包括配置选项示例。

先决条件

- Apache Kafka 和 Kafka 的流正在运行
- 部署和配置 OAuth 2.0 授权服务器以 OAuth 访问 Kafka 代理

- 为 OAuth 2.0 配置 Kafka 代理

流程

1. 创建客户端 secret，并将它作为环境变量挂载到组件。

   例如，这里为 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 密钥必须采用 base64 格式。

2. 创建或编辑 Kafka 组件的资源，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，您可以使用以下选项：

   - 客户端 ID 和 secret
   - 客户端 ID 和刷新令牌
   - 访问令牌
   - 用户名和密码
   - TLS

例如，以下 OAuth 2.0 使用客户端 ID 和 secret 和 TLS 分配给 Kafka Bridge 客户端：

```
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) 身份验证类型设置为 oauth。

(2) 用于身份验证的令牌端点的 URI。

(3) 用于 TLS 连接到授权服务器的可信证书。

根据您如何应用 OAuth 2.0 身份验证以及授权服务器类型，您可以使用其他配置选项：

```
# ...
spec:
  # ...
  authentication:
    # ...
    disableTlsHostnameVerification: true (1)
    checkAccessTokenType: false (2)
    accessTokenIsJwt: false (3)
    scope: any (4)
    audience: kafka (5)
    connectTimeoutSeconds: 60 (6)
    readTimeoutSeconds: 60 (7)
```

```
httpRetries: 2 8
httpRetryPauseMs: 300 9
includeAcceptHeader: false 10
```

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 令牌内返回 typ （类型）声明，您可以应用 checkAccessTokenType: false 来跳过令牌类型检查。默认为 true。

3

如果使用不透明令牌，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来指定范围。在这种情况下，它都是 any。

5

（可选）从令牌端点请求令牌的听众。授权服务器可能需要客户端来指定受众。在本例中，是 kafka。

6

（可选）连接到授权服务器时的连接超时（以秒为单位）。默认值为 60。

7

（可选）连接到授权服务器时读取超时（以秒为单位）。默认值为 60。

8

（可选）重试对授权服务器的失败 HTTP 请求的次数上限。默认值为 0， 表示不会执行重试。要有效地使用这个选项，请考虑减少 connectTimeoutSeconds 和 readTimeoutSeconds 选项的超时时间。但请注意，重试可能会阻止当前 worker 线程可用于其他请求，如果太多请求停滞，则可能会导致 Kafka 代理无响应。

9

（可选）尝试对授权服务器进行另一个重试失败的 HTTP 请求前等待的时间。默认情况下，这个时间被设置为零，这意味着不会应用暂停。这是因为导致失败请求的许多问题是针对每个请求的网络粘合或代理问题，可以快速解决。但是，如果您的授权服务器处于压力

下或高流量，您可能希望将此选项设置为值 100 ms 或更多，以减少服务器上的负载，并增加成功重试的可能性。

10 （可选）有些授权服务器在客户端发送 Accept: application/json 标头时遇到问题。通过设置 includeAcceptHeader: false，不会发送标头。默认为 true。

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，或通过监视 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。

15.5. 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 来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权限级别。

Apache Kafka 的流支持通过 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。

Kafka 允许所有用户默认对代理进行完全访问，并提供 AclAuthorizer 和 StandardAuthorizer 插件来配置基于访问控制列表(ACL)的授权。由这些插件管理的 ACL 规则用于根据 *用户名* 授予或拒绝对资源的访问，这些规则存储在 Kafka 集群本身中。但是，红帽单点登录基于 OAuth 2.0 令牌的授权在您希望实现对 Kafka 代理的访问控制方面具有更大的灵活性。另外，您可以将 Kafka 代理配置为使用 OAuth 2.0 授权和 ACL。

其他资源

- 使用基于 OAuth 2.0 令牌的身份验证
- Kafka 授权
- Red Hat Single Sign-On 文档

15.5.1. OAuth 2.0 授权机制

Apache Kafka 的 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 Red Hat Single Sign-On 扩展基于令牌的身份验证，通过在特定用户上应用定义的安全策略来扩展基于令牌的权限，并为该用户提供授予不同资源的权限列表。策略使用角色和组来匹配用户的权限。OAuth 2.0 授权根据从 Red Hat Single Sign-On Authorization Services 用户获得的授予者列表在本地强制实施权限。

15.5.1.1. Kafka 代理自定义授权器

Red Hat Single Sign-On *authorizer* (KeycloakAuthorizer)提供 Apache Kafka 的 Streams。为了可以使用 Red Hat Single Sign-On 提供的授权服务的 Red Hat Single Sign-On REST 端点，您可以在 Kafka 代理上配置自定义授权器。

授权程序根据需要从授权服务器获取授予权限的列表，并在 Kafka Broker 上本地强制实施授权，为每个客户端请求做出快速授权决策。

15.5.2. 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。

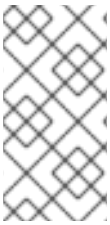
开始前

考虑某些用户所需的访问权限或希望限制某些用户。您可以使用 Red Hat Single Sign-On *组*、*角色*、*客户端* 和 *用户在* Red Hat Single Sign-On 中配置访问权限的组合。

通常，组用于根据机构部门或地理位置匹配用户。和 角色用于根据其功能匹配用户。

使用红帽单点登录，您可以在 LDAP 中存储用户和组，而客户端和角色不能以这种方式存储。存储和

对用户数据的访问可能是您选择配置授权策略的一个因素。



注意

无论在 Kafka 代理上实现的授权是什么，超级用户 始终对 Kafka 代理具有不受限制的访问权限。

先决条件

- Apache Kafka 的流必须配置为在 Red Hat Single Sign-On 中使用 OAuth 2.0 进行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 Authorization Services 为客户端定义资源、授权范围、策略和权限。
3. 通过分配角色和组，将权限绑定到用户和客户端。
4. 通过在编辑器中更新 Kafka 资源的 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，并能够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)
      grantsMaxIdleSeconds: 300 (10)
      grantsGcPeriodSeconds: 300 (11)
      grantsAlwaysLatest: false (12)
      connectTimeoutSeconds: 60 (13)
      readTimeoutSeconds: 60 (14)
      httpRetries: 2 (15)
      enableMetrics: false (16)
      includeAcceptHeader: false (17)
    #...
```

(1)

type keycloak 启用 Red Hat Single Sign-On 授权。

(2)

Red Hat Single Sign-On 令牌端点的 URI。对于生产环境，请始终使用 https:// urls。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)

在启用了授权服务的 Red Hat Single Sign-On 中 OAuth 2.0 客户端定义的客户端 ID。通常，kafka 被用作 ID。

(4)

（可选）如果 Red Hat Single Sign-On Authorization Services 策略无法访问，对 Kafka AclAuthorizer 和 StandardAuthorizer 的 Delegate 授权。默认为 false。

(5)

6

（可选）设计的超级用户。

7

（可选）与授权服务器进行 TLS 连接的可信证书。

8

（可选）两个连续之间的时间授予刷新运行。这是活动会话的最长时间，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）用于刷新（并行）活跃会话授予的线程数量。默认值为 5。

10

（可选）缓存中闲置授权可以被驱除的时间（以秒为单位）。默认值为 300。

11

（可选） 连续运行从缓存中清除过时的作业之间的时间（以秒为单位）。默认值为 300。

12

（可选）控制是否为新会话获取最新的授权。启用后，从 Red Hat Single Sign-On 检索并缓存该用户。默认值为 false。

13

（可选）连接到 Red Hat Single Sign-On 令牌端点时的连接超时（以秒为单位）。默认值为 60。

14

（可选）连接到 Red Hat Single Sign-On 令牌端点时读取超时（以秒为单位）。默认值为 60。

15

（可选）重试的最大次数（不会暂停）授权服务器的失败 HTTP 请求。默认值为 0，表示不会执行重试。要有效地使用这个选项，请考虑减少 connectTimeoutSeconds 和 readTimeoutSeconds 选项的超时时间。但请注意，重试可能会阻止当前 worker 线程可用

于其他请求，如果太多请求停滞，则可能会导致 Kafka 代理无响应。

16 （可选）启用或禁用 OAuth 指标。默认值为 false。

17 （可选）有些授权服务器在客户端发送 Accept: application/json 标头时遇到问题。通过设置 includeAcceptHeader: false，不会发送标头。默认为 true。

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，或通过监视 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. 通过以客户端或具有特定角色的用户访问 Kafka 代理来验证配置的权限，确保它们具有必要的访问权限，或者没有应该具有访问权限。

15.5.3. 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 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。

示例显示了常见 Kafka 操作所需的不同用户权限，如创建和列出主题。

15.5.3.1. Kafka 和 Red Hat Single Sign-On 授权模型概述

Kafka 和 Red Hat Single Sign-On Authorization Services 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 KeycloakAuthorizer 来根据操作和资源类型检查客户端的权限。

Kafka 使用五个资源类型来控制访问： Topic,Group,Cluster,TransactionalId, 和 DelegationToken。每个资源类型都有一组可用权限。

Topic

- 创建
- 写
- 读
- 删除
- Describe
- DescribeConfigs
- 更改
- AlterConfigs

组

- 读
- Describe
- 删除

集群

- 创建
- Describe
- 更改
- DescribeConfigs
- AlterConfigs
- IdempotentWrite
- ClusterAction

TransactionalId

- Describe
- 写

DelegationToken

- **Describe**

Red Hat Single Sign-On Authorization Services 模型

Red Hat Single Sign-On Authorization Services 模型有四个概念来定义和授予权限：*resources（资源）*, *authorization scopes（授权范围）*, *policies（政策）*, 和 *permissions（权限）*。

Resources

资源是一组资源定义，用于将资源与允许的操作匹配。资源可能是单个主题，例如，也可以是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，它代表了资源上可用的一组所有操作。通常，实际上只允许这些操作的子集。

授权范围

授权范围是一组对特定资源定义的所有可用操作。当您定义新资源时，您可以添加来自所有范围集的范围。

策略（policy）

策略是一种授权规则，它使用条件与帐户列表匹配。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

权限向一组用户授予对特定资源定义的授权范围的子集。

其他资源

- **Kafka 授权模型**

15.5.3.2. 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 Red Hat Single Sign-On 角色和资源的基础，以控制对 Kafka 的访问。

要为用户帐户或服务帐户授予 Kafka 权限，您首先在 Red Hat Single Sign-On 中为 Kafka 代理创建 *OAuth 客户端规格*。然后，在客户端上指定 Red Hat Single Sign-On Authorization Services 规则。通常，代表代理的 OAuth 客户端的客户端 id 是 kafka。由 Apache Kafka 的 Streams 提供 的示例配置文件 使用 kafka 作为 OAuth 客户端 ID。

注意

如果有多个 Kafka 集群，可以对所有集群使用单个 OAuth 客户端(kafka)。这为您提供了一个统一的空间，在其中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），并在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 Red Hat Single Sign-On Admin 控制台中启用 Authorization Enabled 选项。

kafka 客户端的范围中存在所有权限。如果您使用不同的 OAuth 客户端 ID 配置不同的 Kafka 集群，它们需要单独的权限集，即使它们是同一 Red Hat Single Sign-On 域的一部分。

当 Kafka 客户端使用 OAUTHBEARER 身份验证时，Red Hat Single Sign-On 授权器 (KeycloakAuthorizer)使用当前会话的访问令牌从 Red Hat Single Sign-On 服务器检索授权列表。要获得授权，授权者会评估 Red Hat Single Sign-On Authorization Services 策略和权限。

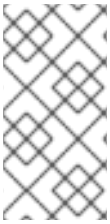
Kafka 权限的授权范围

初始 Red Hat Single Sign-On 配置通常涉及上传授权范围，以创建对每个 Kafka 资源类型执行的所有可能操作的列表。此步骤仅在定义任何权限前执行一次。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，无论资源类型是什么：

- 创建
- 写

- 读
- 删除
- Describe
- 更改
- DescribeConfig
- AlterConfig
- ClusterAction
- IdempotentWrite



注意

如果您确信不需要权限（例如，IdempotentWrite），您可以从授权范围列表中省略它。但是，这个权限不适用于 Kafka 资源上的目标。

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匹配的模式。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：

- 完全匹配（当模式不以 *结尾）

- 前缀匹配（当模式以 *结尾时）

资源模式示例

```
Topic:my-topic
Topic:orders-*
Group:orders-*
Cluster:*
```

另外，一般模式格式可以带有 kafka-cluster:*CLUSTER-NAME* 前缀，后跟逗号，*CLUSTER-NAME* 指的是 Kafka 自定义资源中的 metadata.name。

带有集群前缀的资源模式示例

```
kafka-cluster:my-cluster,Topic:*
kafka-cluster:*,Group:b_*
```

当缺少 kafka-cluster 前缀时，它被假定为 kafka-cluster:*。

在定义资源时，您可以将它与与资源相关的授权范围列表相关联。设置目标资源类型有意义的操作。

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考虑访问控制。

应用访问权限的策略

策略用于对一个或多个用户帐户或服务帐户的目标权限。目标可以参考：

- 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

一个策略被授予一个唯一名称，可以重复使用以多个资源为目标的多个权限。

授予访问权限的权限

使用细粒度权限将策略、资源和授权范围一起拉取，以授予用户访问权限。

每个权限的名称应该明确定义它授予哪些用户的权限。例如，Dev Team B 可以从以 x 开始的主题读取。

其他资源

- 有关如何通过 Red Hat Single Sign-On Authorization Services 配置权限的更多信息，请参阅 第 15.5.4 节 “Trying Red Hat Single Sign-On Authorization Services”。

15.5.3.3. 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，需要 Create 权限用于特定主题，或 Cluster:kafka-cluster。

```
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 Describe 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
```

显示主题详情

要显示主题的详情，需要主题的 Describe 和 DescribeConfigs 权限。

```
bin/kafka-topics.sh --describe --topic my-topic \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-
config=/tmp/config.properties
```

将消息生成到主题

要生成消息到主题，需要主题的 Describe 和 Write 权限。

如果主题还没有创建，并且启用了主题 auto-creation，则需要创建主题的权限。

```
bin/kafka-console-producer.sh  --topic my-topic \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --producer.config=/tmp/config.properties
```

使用主题的消息

为了消费主题中的消息，需要有主题的 Describe 和 Read 权限。在主题中使用通常依赖于将消费者偏移存储在消费者组中，这需要对消费者组需要额外的 Describe 和 Read 权限。

需要 两个资源 才能匹配。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
```

使用幂等制作者将消息生成到主题

除了生成主题的权限外，Cluster:kafka-cluster 资源还需要额外的 IdempotentWrite 权限。

需要 两个资源 才能匹配。例如：

```
Topic:my-topic
Cluster:kafka-cluster
```

```
bin/kafka-console-producer.sh  --topic my-topic \
  --bootstrap-server 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

当列出消费者组时，只有用户具有 Describe 权限的组才会返回。另外，如果用户对 Cluster:kafka-cluster 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情，在组以及与组关联的主题上需要 Describe 权限。

```
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，需要主题的 Describe 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 获取代理配置，Cluster:kafka-cluster 需要 DescribeConfigs 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 DescribeConfigs 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

要删除主题，主题需要 Describe 和 Delete 权限。

```
bin/kafka-topics.sh --delete --topic my-topic \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 --command-
config=/tmp/config.properties
```

选择引导分区

要为主题分区运行领导选择，Cluster:kafka-cluster 需要 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，对涉及的主题需要 Describe 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，Cluster:kafka-cluster 需要 Describe 和 Alter 权限。另外，有关涉及的主题还需要 Describe 权限。

```
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
```

对于 Cluster:kafka-cluster 以及涉及的每个主题，需要验证分区重新分配、Describe 和 AlterConfigs 权限。

```
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
```

15.5.4. Trying Red Hat Single Sign-On Authorization Services

本例解释了如何在 keycloak 授权中使用 Red Hat Single Sign-On Authorization Services。使用 Red Hat Single Sign-On Authorization Services 对 Kafka 客户端强制实施访问限制。Red Hat Single Sign-On Authorization Services 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。

Red Hat Single Sign-On Authorization Services REST 端点提供经过身份验证的用户授予资源的权限列表。在 Kafka 客户端建立经过身份验证的会话后，从 Red Hat Single Sign-On 服务器获取授权（权限）列表作为第一个操作。该列表在后台刷新，以便检测对授予的更改。为每个用户会话在 Kafka 代理上本地执行授权，以提供快速授权决策。

Apache Kafka 的流提供了 示例配置文件。这包括设置 Red Hat Single Sign-On 的以下示例文件：

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

配置了示例组、用户、角色和客户端的 Red Hat Single Sign-On 域示例。您可以将域导入到 Red Hat Single Sign-On 实例中，以设置精细的权限来访问 Kafka。

如果要尝试使用 Red Hat Single Sign-On 的示例，请使用这些文件执行本节中概述的任务，如显示的顺序。

1. 访问 Red Hat Single Sign-On 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

身份验证

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。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。

带有组或角色策略的目标权限

在 Red Hat Single Sign-On 中，启用了服务帐户的机密客户端可以使用客户端 ID 和 secret 在自己的名称中向服务器进行身份验证。这对通常充当自有名称的微服务来说很方便，而不适用于特定用户（如网站）的代理。服务帐户可以像常规用户一样分配角色。但是，不能为其分配组。因此，如果要使用服务帐户将权限目标到微服务，则无法使用组策略，而应使用角色策略。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，您可以将其作为使用组策略而不是角色策略的副作用实现。这是本例中以 ClusterManager 开头的权限。执行集群管理通常是使用 CLI 工具交互完成的。在使用生成的访问令牌向 Kafka 代理进行身份验证前，需要用户登录是合理的。在这种情况下，访问令牌代表特定用户，而不是客户端应用程序。

15.5.4.1. 访问 Red Hat Single Sign-On 管理控制台

设置 Red Hat Single Sign-On，然后连接到其管理控制台并添加预配置的域。使用示例 kafka-authz-realm.json 文件导入域。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 域的 Kafka 集群上资源的访问权限。

先决条件

- 一个正常运行的 OpenShift 集群。
- Apache Kafka 示例/security/keycloak-authorization/kafka-authz-realm.json 文件的 Streams，其中包含预配置的域。

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 Server Installation and configuration 所述。
2. 等待 Red Hat Single Sign-On 实例正在运行。
3. 获取可以访问管理控制台的外部主机名。

```
NS=sso
oc get ingress keycloak -n $NS
```

在本例中，我们假定 Red Hat Single Sign-On 服务器在 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，以识别 secret 的名称(secretKeyRef.name)。

5. 使用 secret 的名称来获取明文密码。

```
SECRET_NAME=credential-keycloak
oc get -n $NS secret $SECRET_NAME -o yaml | grep PASSWORD | awk '{print $2}' |
base64 -D
```

在本例中，我们假定 secret 的名称是 credential-keycloak。

6. 使用用户名 admin 和密码您获取的密码，登录管理控制台。

使用 https://*HOSTNAME* 访问 Kubernetes Ingress。

现在，您可以使用管理控制台将示例域上传到 Red Hat Single Sign-On。

7. 单击 Add Realm 以导入示例域。

8. 添加 examples/security/keycloak-authorization/kafka-authz-realm.json 文件，然后点 Create。

您现在具有 kafka-authz 作为管理控制台中的当前域。

默认视图显示 Master 域。

9. 在 Red Hat Single Sign-On Admin 控制台中，进入 Clients > kafka > Authorization > Settings，检查 Decision Strategy 是否已设置为 Affirmative。

高效策略意味着，必须至少满足一个策略才能访问 Kafka 集群。

10. 在 Red Hat Single Sign-On Admin Console 中，进入 Groups, Users, Roles 和 Clients 来查看 realm 配置。

组

组 用于创建用户组并设置用户权限。组是分配了名称的用户集合。它们用于将用户划分成地理、组织或部门单元。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 用户界面使用户成为组的成员，例如，授予对 Kafka 资源的权限。

用户

用户 用于创建用户。在本例中，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 身份提供程序中。

角色

Roles 代表用户或客户端具有特定权限。角色是一种类似于组的概念。它们通常 *用于标记* 具有组织角色的用户，并具有必要的权限。角色不能存储在 LDAP 身份提供程序中。如果 LDAP 是要求，您可以使用组，并将 Red Hat Single Sign-On 角色添加到组中，以便在为用户分配他们获得对应的角色时。

客户端

客户端可以 具有特定配置。在本例中，配置了 kafka, kafka-cli, team-a-client, 和 team-b-client 客户端。

- Kafka 代理使用 kafka 客户端来执行访问令牌验证所需的 OAuth 2.0 通信。此客户端还包含用于在 Kafka 代理上执行授权的授权服务资源定义、策略和授权范围。授权配置在 Authorization 选项卡的 kafka 客户端中定义，在从 Settings 标签页上切换 Authorization 时，它就会可见。

- 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由 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 客户端是代表服务的机密客户端，对某些 Kafka 主题具有部分访问权限。

11. 在 Red Hat Single Sign-On Admin 控制台中，进入 Authorization > Permissions 来查看

授予使用为域定义的资源和策略的权限。

例如，kafka 客户端具有以下权限：

```
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 Team A

Dev Team A realm 角色可以写入任何集群中以 x_ 开头的主题。它组合了名为 Topic:xuildDefaults、Describe 和 Write 范围的资源，以及 Dev Team A 策略。Dev Team A 策略与具有称为 Dev Team A 的 realm 角色的所有用户匹配。

Dev Team B

Dev Team B realm 角色可以从任何集群上的 x_ 开头的主题读取。这将合并 Topic:xuild Defaults , Group:x object resources, Describe 和 Read scopes, 和 Dev Team B 策略。Dev Team B 策略与具有称为 Dev Team B 的 realm 角色的所有用户匹配。匹配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消耗的偏移量。

15.5.4.2. 使用 Red Hat Single Sign-On 授权部署 Kafka 集群

部署配置为连接到 Red Hat Single Sign-On 服务器的 Kafka 集群。使用示例 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

- **Red Hat Single Sign-On 授权服务器部署到 OpenShift 集群，并使用 example 域加载。**
- **Cluster Operator 部署到 OpenShift 集群。**
- **Apache Kafka 示例/security/keycloak-authorization/kafka-ephemeral-oauth-single-keycloak-authz.yaml 自定义资源的 Streams。**

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 Red Hat Single Sign-On 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.pem
```

该证书是必需的，因为 Kubernetes Ingress 用于进行安全(HTTPS)连接。

通常没有单个证书，而是证书链。您只需要提供最顶层的签发者 CA，该 CA 在 /tmp/sso.pem 文件中最后列出。您可以手动提取它，也可以使用以下命令：

在证书链中提取顶级 CA 证书的命令示例

```
split -p "-----BEGIN CERTIFICATE-----" sso.pem sso-
for f in $(ls sso-*); do mv $f $f.pem; done
cp $(ls sso-* | sort -r | head -n 1) sso-ca.crt
```

注意

可信 CA 证书通常从可信来源获取，而不使用 openssl 命令。

2. 将证书作为机密部署到 OpenShift。

```
oc create secret generic oauth-server-cert --from-file=/tmp/sso-ca.crt -n $NS
```

3. 将主机名设置为环境变量

```
SSO_HOST=SSO-HOSTNAME
```

4. 创建和部署示例 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 -
```

15.5.4.3. 为 CLI Kafka 客户端会话准备 TLS 连接

为交互式 CLI 会话创建一个新 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

- Red Hat Single Sign-On 授权服务器部署到 OpenShift 集群，并使用 example 域加载。

  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 Apache Kafka 镜像的 Streams 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/amq-streams/kafka-37-rhel9:2.7.0 kafka-cli -n $NS -- /bin/sh
```

注意

如果 oc 超时等待镜像下载，后续的尝试可能会导致 *AlreadyExists* 错误。

2. 附加到 pod 容器。

```
oc attach -ti kafka-cli -n $NS
```

3. 使用 Red Hat Single Sign-On 实例的主机名，为使用 TLS 的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.pem
```

通常没有单个证书，而是证书链。您只需要提供最顶层的签发者 CA，该 CA 在 /tmp/sso.pem 文件中最后列出。您可以手动提取它，也可以使用以下命令：

在证书链中提取顶级 CA 证书的命令示例

```
split -p "-----BEGIN CERTIFICATE-----" sso.pem sso-
for f in $(ls sso-*); do mv $f $f.pem; done
cp $(ls sso-* | sort -r | head -n 1) sso-ca.crt
```

注意

可信 CA 证书通常从可信来源获取，而不使用 openssl 命令。

4. 为 TLS 连接到 Kafka 代理创建一个信任存储。

```
keytool -keystore /tmp/truststore.p12 -storetype pkcs12 -alias sso -storepass
$STOREPASS -import -file /tmp/sso-ca.crt -noprompt
```

5. 使用 Kafka bootstrap 地址作为 Kafka 代理的主机名和 tls 侦听器端口(9093)来为 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.pem
```

获取的 .pem 文件通常不是单个证书，而是一个证书链。您只需要提供最顶层的签发者 CA，该 CA 在 /tmp/my-cluster-kafka.pem 文件中最后列出。您可以手动提取它，也可以使用以下命令：

在证书链中提取顶级 CA 证书的命令示例

```
split -p "-----BEGIN CERTIFICATE-----" /tmp/my-cluster-kafka.pem kafka-
for f in $(ls kafka-*); do mv $f $f.pem; done
cp $(ls kafka-* | sort -r | head -n 1) my-cluster-kafka-ca.crt
```

注意

可信 CA 证书通常从可信来源获取，而不使用 openssl 命令。在本例中，我们假设客户端在部署了 Kafka 集群的同一命名空间中的 pod 中运行。如果客户端从 OpenShift 集群外部访问 Kafka 集群，您必须首先确定 bootstrap 地址。在这种情况下，您还可以从 OpenShift secret 直接获取集群证书，且不需要 openssl。如需更多信息，请参阅 第 14 章 *设置 Kafka 集群的客户端访问权限*。

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-ca.crt -noprompt
```

保持会话处于打开状态，以检查授权访问权限。

15.5.4.4. 使用 CLI Kafka 客户端会话检查对 Kafka 的授权访问

使用交互式 CLI 会话，检查通过 Red Hat Single Sign-On 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，以使用不同级别访问权限的用户和服务帐户创建主题。

使用 team-a-client 和 team-b-client 客户端来检查授权规则。使用 alice admin 用户对 Kafka 执行额外的管理任务。

本例中使用的 Apache Kafka 镜像的流包含 Kafka producer 和消费者二进制文件。

先决条件

- ZooKeeper 和 Kafka 在 OpenShift 集群中运行，以便能够发送和接收信息。

- 启动交互式 CLI Kafka 客户端会话。

  Apache Kafka 下载。

设置客户端和 admin 用户配置

1. 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。这个机制需要客户端 ID 和客户端 secret，这意味着客户端首先连接到 Red Hat Single Sign-On 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. 使用 curl 验证 admin 用户 alice，并执行密码授权身份验证来获取刷新令牌。

```
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，它长期且不会过期。

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经过身份验证的刷新令牌进行身份验证。刷新令牌请求 scenes 后面的访问令牌，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 配置检查您可以生成消息到以 a_ 或 x_ 开头的主题。

1. 写入主题 my-topic。

```
bin/kafka-console-producer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --
topic my-topic \
  --producer.config=/tmp/team-a-client.properties
First message
```

此请求会返回 Not authorized to access topics: [my-topic] 错误。

team-a-client 有一个 Dev Team A 角色，授予它对以 a_ 开头的主题执行任何受支持操作的权限，但只能写入以 x_ 开头的主题。名为 my-topic 的主题都匹配这些规则。

2. 写入主题 a_messages。

```
bin/kafka-console-producer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --
topic a_messages \
  --producer.config /tmp/team-a-client.properties
First message
Second message
```

消息被成功生成到 Kafka。

3. 按 CTRL+C 退出 CLI 应用程序。

4. 检查 Kafka 容器日志中 的授权 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消息

使用 team-a-client 配置来消耗来自主题 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
```

请求返回错误，因为 team-a-client 的 Dev Team A 角色只能访问名称以 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消费者组。

```
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
```

消费者从 a_messages 主题接收所有消息。

使用授权访问权限管理 Kafka

team-a-client 是一个帐户，没有集群级别的访问权限，但可用于一些管理操作。

1. 列出主题。

```
bin/kafka-topics.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --command-
config /tmp/team-a-client.properties --list
```

返回 a_messages 主题。

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
```

返回 a_consumer_group_1 使用者组。

获取集群配置的详情。

```
bin/kafka-configs.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --command-
config /tmp/team-a-client.properties \
 --entity-type brokers --describe --entity-default
```

请求返回错误，因为操作需要 team-a-client 没有的集群级别权限。

使用具有不同权限的客户端

使用 team-b-client 配置为以 b_ 开头的主题生成消息。

1. 写入主题 a_messages。

```
bin/kafka-console-producer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --
topic a_messages \
 --producer.config /tmp/team-b-client.properties
Message 1
```

此请求会返回 Not authorized to access topics: [a_messages] 错误。

2. 写入主题 b_messages。

```
bin/kafka-console-producer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --
topic b_messages \
 --producer.config /tmp/team-b-client.properties
Message 1
Message 2
Message 3
```

消息被成功生成到 Kafka。

3. 写入主题 x_messages。

```
bin/kafka-console-producer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --
topic x_messages \
```

```
  --producer.config /tmp/team-b-client.properties
Message 1
```

返回了 Not authorized access topics: [x_messages] 错误，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 --bootstrap-server my-cluster-kafka-bootstrap:9093 --
topic x_messages \
  --producer.config /tmp/team-a-client.properties
Message 1
```

此请求会返回 Not authorized to access topics: [x_messages] 错误。team-a-client 可以写入 x_messages 主题，但如果它尚不存在，则没有创建主题的权限。在 team-a-client 可以写入 x_messages 主题之前，admin *power 用户* 必须使用正确的配置创建它，如分区和副本数。

使用授权 admin 用户管理 Kafka

使用 admin 用户 alice 管理 Kafka。alice 具有管理任何 Kafka 集群上的所有内容的完整访问权限。

1. 以 alice 身份创建 x_messages 主题。

```
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
```

主题已创建成功。

2. 以 alice 身份列出所有主题。

```
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 Team A 和 Dev Team B 角色对以 x_ 开头的主题具有 Describe 权限，但它们无法看到其他团队的主题，因为它们没有 Describe 权限。

3. 使用 team-a-client 生成信息到 x_messages 主题：

```
bin/kafka-console-producer.sh --bootstrap-server 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，会在 Kafka 中成功生成信息。

4. 使用 team-b-client 生成消息到 x_messages 主题。

```
bin/kafka-console-producer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --
topic x_messages \
  --producer.config /tmp/team-b-client.properties
Message 4
Message 5
```

此请求会返回 Not authorized to access topics: [x_messages] 错误。

5. 使用 team-b-client 来消耗来自 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-b-client.properties --group
x_consumer_group_b
```

消费者接收来自 x_messages 主题的所有消息。

6. 使用 team-a-client 来消耗来自 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
```

此请求会返回 Not authorized to access topics: [x_messages] 错误。

7. 使用 team-a-client 来消耗来自以 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
```

此请求会返回 Not authorized to access topics: [x_messages] 错误。

Dev Team A 没有以 x_ 开始的主题的 Read 访问权限。

8. 使用 alice 生成消息到 x_messages 主题。

```
bin/kafka-console-consumer.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 --topic x_messages \
  --from-beginning --consumer.config /tmp/alice.properties
```

消息被成功生成到 Kafka。

alice 可以从任何主题读取或写入到任何主题。

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
```

本例中的集群配置为空。

其他资源

- 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 16 章 管理 TLS 证书

Streams for Apache Kafka 支持 TLS 用于 Apache Kafka 组件的 Kafka 和 Streams 之间的加密通信。

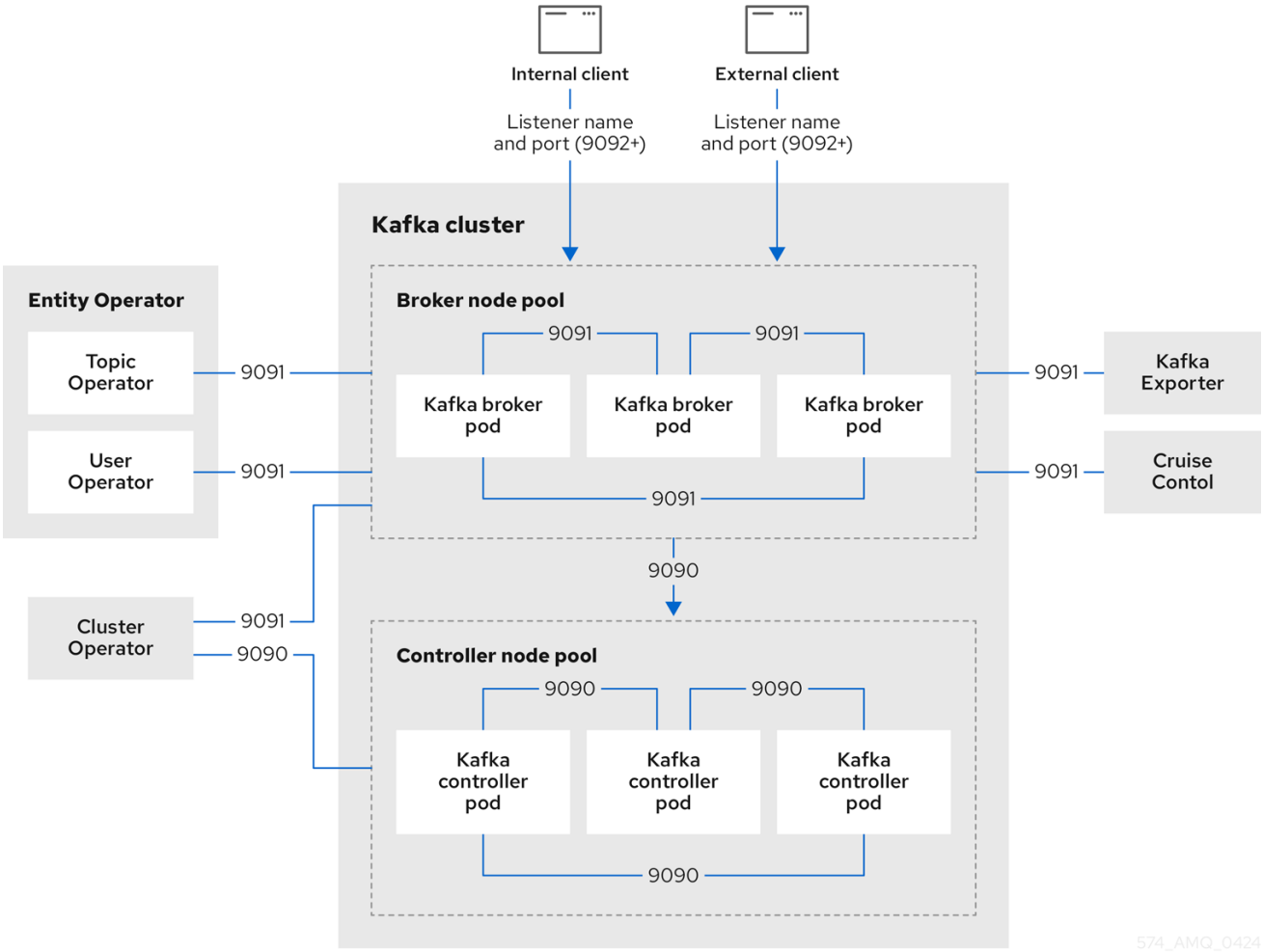
在 KRaft 模式下使用 Kafka 时，Apache Kafka 的流为以下组件之间的通信建立加密的 TLS 连接：

- Kafka 代理
- Kafka 控制器
- Kafka 代理和控制器
- Apache Kafka operator 和 Kafka 的流
- Cruise Control 和 Kafka 代理
- Kafka Exporter 和 Kafka 代理

客户端和 Kafka 代理之间的连接使用监听程序，您必须将它们配置为使用 TLS 加密通信。您可以在 Kafka 自定义资源中配置这些监听程序，每个监听程序名称和端口号必须在集群中唯一。Kafka 代理和 Kafka 客户端之间的通信会根据为监听程序配置 tls 属性的方式进行加密。如需更多信息，请参阅 第 14 章 *设置 Kafka 集群的客户端访问权限*。

下图显示了安全通信的连接。

图 16.1. KRaft 型 Kafka 通信由 TLS 加密保护



图中显示的端口如下：

control plane 侦听器(9090)

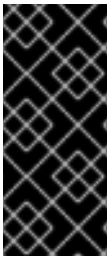
端口 9090 上的内部 control plane 侦听器有助于 Kafka 控制器和代理到控制器通信之间的 interbroker 通信。另外，Cluster Operator 通过监听程序与控制器通信。Kafka 客户端无法访问此监听程序。

复制监听程序(9091)

代理之间的数据复制，以及从 Apache Kafka operator、Cruise Control 和 Kafka Exporter 的 Streams 到代理的内部连接，使用端口 9091 上的复制监听程序。Kafka 客户端无法访问此监听程序。

客户端连接(9092 或更高版本)的监听程序

对于 TLS 加密的通信（通过配置监听程序），内部和外部客户端连接到 Kafka 代理。外部客户端 (producers 和 consumers)通过公告的监听程序端口连接到 Kafka 代理。



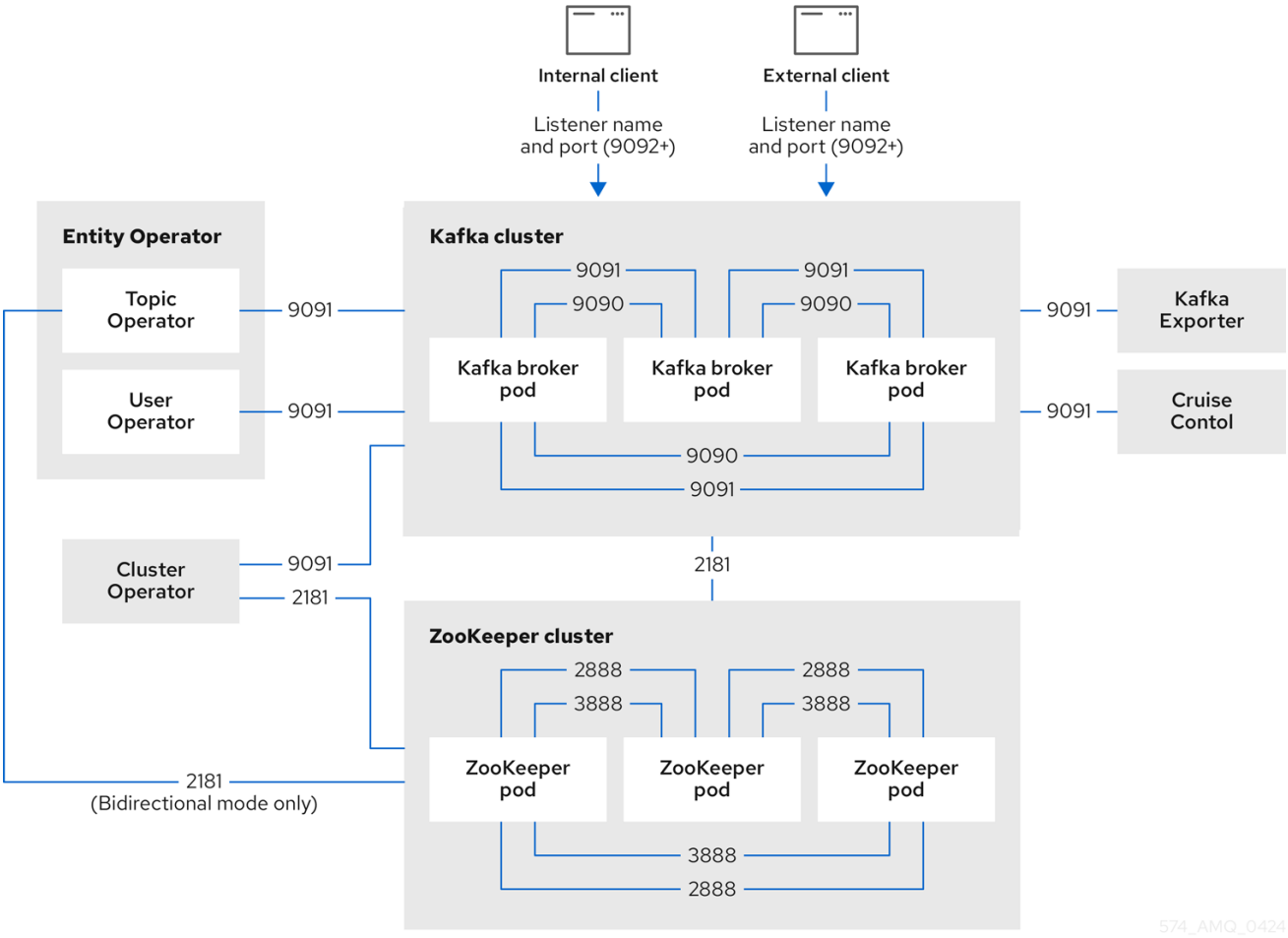
重要

当为客户端配置监听程序对代理的访问时，您可以使用端口 9092 或更高版本(9093、9094 等)，但有一些例外。侦听程序无法配置为使用为 interbroker 通信(9090 和 9091)、Prometheus 指标(9404)和 JMX (Java 管理扩展)监控(9999)保留的端口。

如果您使用 ZooKeeper 进行集群管理，则 ZooKeeper 和 Kafka 代理和 Apache Kafka operator 的 Streams 之间有 TLS 连接。

下图显示了使用 ZooKeeper 时安全通信的连接。

图 16.2. Kafka 和 ZooKeeper 通信由 TLS 加密保护



ZooKeeper 端口如下：

ZooKeeper Port (2181)

用于连接 Kafka 代理的 ZooKeeper 端口。另外，Cluster Operator 通过这个端口与 ZooKeeper 通信。如果您以双向模式使用 Topic Operator，它还通过这个端口与 ZooKeeper 通信。

ZooKeeper 间的通信端口(2888)

ZooKeeper 端口，用于在 ZooKeeper 节点之间进行干预。

ZooKeeper 领导选举端口(3888)

ZooKeeper 端口，用于 ZooKeeper 集群中的节点间的领导选举机制。

16.1. 内部集群 CA 和客户端 CA

要支持加密，Apache Kafka 组件的每个流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 CA （证书颁发机构）签名。

CA （证书授权机构）证书由 Cluster Operator 生成，以验证组件和客户端的身份。

同样，每个使用 mTLS 连接到 Apache Kafka 的 Kafka 客户端应用程序都需要使用私钥和证书。第二个名为 *客户端 CA* 的内部 CA 用于为 Kafka 客户端签名证书。

集群 CA 和客户端 CA 都有一个自签名公钥证书。

Kafka 代理被配置为信任由集群 CA 或客户端 CA 签名的证书。客户端不需要连接到的组件，如 ZooKeeper，只信任由集群 CA 签名的证书。除非外部监听程序的 TLS 加密被禁用，否则客户端应用程序必须信任由集群 CA 签名的证书。对于执行 mTLS 验证的客户端应用程序，这也也是如此。

默认情况下，Apache Kafka 的 Streams 会自动生成和更新集群 CA 或客户端 CA 发布的 CA 证书。您可以使用 Kafka.spec.clusterCa 和 Kafka.spec.clientsCa 属性来配置这些 CA 证书的管理。

注意

如果您不想使用 Cluster Operator 生成的 CA，您可以安装自己的集群和客户端 CA 证书。您提供的任何证书都不会由 Cluster Operator 更新。

16.2. 由 OPERATOR 生成的 SECRET

Cluster Operator 会自动设置和更新 TLS 证书，以便在集群中启用加密和身份验证。如果要启用 Kafka 代理和客户端之间的加密或 mTLS 身份验证，它还设置其他 TLS 证书。

部署自定义资源时会创建 secret，如 Kafka 和 KafkaUser。Apache Kafka 的 Streams 使用这些 secret 存储 Kafka 集群、客户端和用户的私钥证书。secret 用于在 Kafka 代理和代理和客户端之间建立 TLS 加密连接。它们也用于 mTLS 身份验证。

集群和客户端 secret 始终对：一个包含公钥，以及一个包含私钥。

集群 secret

集群 secret 包含用于为 Kafka 代理证书签名的 *集群 CA*。连接客户端使用证书建立与 Kafka 集群的 TLS 加密连接。证书会验证代理身份。

客户端 secret

客户端 secret *包含用户* 签署自己的客户端证书的客户端 CA。这允许对 Kafka 集群进行 mutual 身份验证。代理通过证书验证客户端的身份。

用户 secret

用户 secret 包含私钥和证书。secret 在创建新用户时由客户端 CA 创建并签名。密钥和证书用于在访问集群时验证和授权用户。

注意

您可以为 TLS *侦听程序或启用了 TLS 加密的外部监听程序提供 Kafka 侦听器证书*。使用 Kafka 侦听器证书融合您已有的安全基础架构。

16.2.1. 使用 PEM 或 PKCS #12 格式的密钥和证书进行 TLS 身份验证

由 Apache Kafka 的 Streams 创建的 secret 以 PEM (Privacy Enhanced Mail)和 PKCS TOTP (Public-Key Cryptography Standards)格式提供私钥和证书。PEM 和 PKCS #12 是用于使用 SSL 协议的 TLS 通信的 OpenSSL 生成的密钥格式。

您可以配置 mutual TLS (mTLS)身份验证，它使用为 Kafka 集群和用户生成的 secret 中包含的凭证。

要设置 mTLS，您必须首先执行以下操作：

- 使用使用 mTLS 的监听程序配置 Kafka 集群
- 创建一个 KafkaUser，为 mTLs 提供客户端凭证

当您部署 Kafka 集群时，会创建一个 &lt ;cluster_name>-cluster-ca-cert secret，并使用公钥验证集群。您可以使用公钥为客户端配置信任存储。

当您创建 KafkaUser 时，会使用 密钥和证书创建一个 <kafka_user_name > secret 来验证用户（客户端）。使用这些凭据为客户端配置密钥存储。

当 Kafka 集群和客户端设置为使用 mTLS 时，您可以从 secret 中提取凭证并将其添加到客户端配置中。

PEM 密钥和证书

对于 PEM，您可以在客户端配置中添加以下内容：

truststore

- 来自 < cluster_name>-cluster-ca-cert secret 的 ca.crt，这是集群的 CA 证书。

Keystore

- user.crt 来自 & lt;kafka_user_name > secret，这是用户的公共证书。
- < kafka_user_name> secret 中的 user.key，这是用户的私钥。

PKCS #12 密钥和证书

对于 PKCS #12，您可以在客户端配置中添加以下内容：

truststore

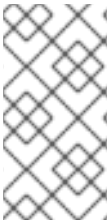
- 来自 < cluster_name>-cluster-ca-cert secret 的 ca.p12，这是集群的 CA 证书。
-

< cluster_name>-cluster-ca-cert secret 中的 ca.password，这是访问公共集群 CA 证书的密码。

Keystore

- 来自 < kafka_user_name> secret 的 user.p12，这是用户的公钥证书。
- < kafka_user_name> secret 中的 user.password，这是访问 Kafka 用户的公钥证书的密码。

Java 支持 PKCS #12，以便您可以将证书的值直接添加到 Java 客户端配置中。您还可以从安全存储位置引用证书。使用 PEM 文件时，您必须以单行格式直接将证书添加到客户端配置中。选择一个适合在 Kafka 集群和客户端之间建立 TLS 连接的格式。如果您对 PEM 不熟悉，可以使用 PKCS #12。



注意

所有密钥的大小都是 2048 位，默认情况下，对来自初始生成的 365 天有效。您可以更改有效期。

16.2.2. Cluster Operator 生成的 secret

Cluster Operator 生成以下证书，该证书保存为 OpenShift 集群中的 secret。Apache Kafka 的流默认使用这些 secret。

集群 CA 和客户端 CA 为私钥和公钥具有单独的 secret。

<cluster_name>-cluster-ca

包含集群 CA 的私钥。Apache Kafka 和 Kafka 组件的流使用私钥为服务器证书签名。

<cluster_name>-cluster-ca-cert

包含集群 CA 的公钥。Kafka 客户端使用公钥验证它们正在与 TLS 服务器身份验证连接的 Kafka 代理的身份。

<cluster_name>-clients-ca

包含客户端 CA 的私钥。Kafka 客户端在连接到 Kafka 代理时使用私钥为 mTLS 身份验证签署新的用户证书。

<cluster_name>-clients-ca-cert

包含客户端 CA 的公钥。Kafka 代理使用公钥验证使用 mTLS 身份验证时访问 Kafka 代理的客户端的身份。

用于 Apache Kafka 组件的流通信的 secret 包含由集群 CA 签名的私钥和公钥证书。

<cluster_name>-kafka-brokers

包含 Kafka 代理的私钥和公钥。

<cluster_name>-zookeeper-nodes

包含 ZooKeeper 节点的私钥和公钥。

<cluster_name>-cluster-operator-certs

包含用于加密 Cluster Operator 和 Kafka 或 ZooKeeper 间的通信的私钥和公钥。

<cluster_name>-entity-topic-operator-certs

包含用于加密主题 Operator 和 Kafka 或 ZooKeeper 之间通信的私钥和公钥。

<cluster_name>-entity-user-operator-certs

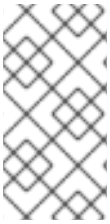
包含用于加密用户 Operator 和 Kafka 或 ZooKeeper 之间的通信的私钥和公钥。

<cluster_name>-cruise-control-certs

包含用于加密 Cruise 控制和 Kafka 或 ZooKeeper 之间的通信的私钥和公钥。

<cluster_name>-kafka-exporter-certs

包含用于加密 Kafka Exporter 和 Kafka 或 ZooKeeper 之间通信的私钥和公钥。



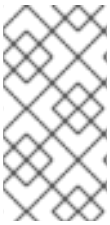
注意

您可以提供自己的服务器证书和私钥，以使用 Kafka *侦听器证书* 连接到 Kafka 代理，而不是由集群 CA 签名的证书。

16.2.3. 集群 CA secret

集群 CA secret 由 Kafka 集群中的 Cluster Operator 管理。

客户端只需要 *<cluster_name>* -cluster-ca-cert secret。所有其他集群 secret 都可通过 Streams for Apache Kafka 组件访问。如果需要，您可以使用 OpenShift 基于角色的访问控制来执行此操作。



注意

< *cluster_name>* -cluster-ca-cert 中的 CA 证书必须由 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 16.1. < *cluster_name>*-cluster-ca secret 中的字段

字段	Description
ca.key	集群 CA 的当前私钥。

表 16.2. < *cluster_name>*-cluster-ca-cert secret 中的字段

字段	Description
ca.p12	用于存储证书和密钥的 PKCS #12 存储。
ca.password	用于保护 PKCS #12 存储的密码。
ca.crt	集群 CA 的当前证书。

表 16.3. < *cluster_name>*-kafka-brokers secret 中的字段

字段	Description
***<cluster_name>*-kafka-*<num>*.p12**	用于存储证书和密钥的 PKCS #12 存储。
***<cluster_name>*-kafka-*<num>*.password**	用于保护 PKCS #12 存储的密码。
***<cluster_name>*-kafka-*<num>*.crt**	Kafka 代理 pod < *num*> 的证书。在 <cluster*_name> -cluster-ca* 中由当前或以前的集群 CA 私钥签名。
***<cluster_name>*-kafka-*<num>*.key**	Kafka 代理 pod < ***num> 的私钥***。

表 16.4. < *cluster_name>*-zookeeper-nodes secret 中的字段

字段	Description
***<cluster_name>*-zookeeper-*<num>*.p12**	用于存储证书和密钥的 PKCS #12 存储。
***<cluster_name>*-zookeeper-*<num>*.password**	用于保护 PKCS #12 存储的密码。
***<cluster_name>*-zookeeper-*<num>*.crt**	ZooKeeper 节点 *<num>* 的证书。在 <cluster_***name> -cluster*-ca** 中由当前或以前的集群 CA 私钥签名。
***<cluster_name>*-zookeeper-*<num>*.key**	ZooKeeper pod < ***num> 的私钥***。

表 16.5. < *cluster_name*>-cluster-operator-certs secret 中的字段

字段	Description
cluster-operator.p12	用于存储证书和密钥的 PKCS #12 存储。
cluster-operator.password	用于保护 PKCS #12 存储的密码。
cluster-operator.crt	Cluster Operator 和 Kafka 或 ZooKeeper 之间的 mTLS 通信的证书。在 <cluster_***name> -cluster*-ca** 中由当前或以前的集群 CA 私钥签名。
cluster-operator.key	Cluster Operator 和 Kafka 或 ZooKeeper 之间的 mTLS 通信的私钥。

表 16.6. < *cluster_name*>-entity-topic-operator-certs secret 中的字段

字段	Description
entity-operator.p12	用于存储证书和密钥的 PKCS #12 存储。
entity-operator.password	用于保护 PKCS #12 存储的密码。
entity-operator.crt	Topic Operator 和 Kafka 或 ZooKeeper 之间的 mTLS 通信的证书。在 <cluster_***name> -cluster*-ca** 中由当前或以前的集群 CA 私钥签名。
entity-operator.key	Topic Operator 和 Kafka 或 ZooKeeper 之间的 mTLS 通信的私钥。

表 16.7. < *cluster_name*>-entity-user-operator-certs secret 中的字段

字段	Description
entity-operator.p12	用于存储证书和密钥的 PKCS #12 存储。
entity-operator.password	用于保护 PKCS #12 存储的密码。
entity-operator.crt	User Operator 和 Kafka 或 ZooKeeper 之间的 mTLS 通信的证书。在 <cluster_***name> -cluster*-ca 中由当前或以前的集群 CA 私钥签名。
entity-operator.key	User Operator 和 Kafka 或 ZooKeeper 之间的 mTLS 通信的私钥。

表 16.8. < *cluster_name*>-cruise-control-certs secret 中的字段

字段	Description
cruise-control.p12	用于存储证书和密钥的 PKCS #12 存储。
cruise-control.password	用于保护 PKCS #12 存储的密码。
cruise-control.crt	用于 Cruise Control 和 Kafka 或 ZooKeeper 之间的 mTLS 通信的证书。在 <cluster_***name> -cluster*-ca 中由当前或以前的集群 CA 私钥签名。
cruise-control.key	Cruise Control 和 Kafka 或 ZooKeeper 之间的 mTLS 通信的私钥。

表 16.9. < *cluster_name*>-kafka-exporter-certs secret 中的字段

字段	Description
kafka-exporter.p12	用于存储证书和密钥的 PKCS #12 存储。
kafka-exporter.password	用于保护 PKCS #12 存储的密码。
kafka-exporter.crt	Kafka Exporter 和 Kafka 或 ZooKeeper 之间的 mTLS 通信的证书。在 <cluster_***name> -cluster*-ca 中由当前或以前的集群 CA 私钥签名。
kafka-exporter.key	Kafka Exporter 和 Kafka 或 ZooKeeper 之间的 mTLS 通信的私钥。

16.2.4. 客户端 CA secret

客户端 CA secret 由 Kafka 集群中的 Cluster Operator 管理。

<cluster_name>-clients-ca-cert 中的正式是 Kafka 代理信任的证书。

& *lt;cluster_name>* -clients-ca secret 用于为客户端应用程序的证书签名。如果打算在没有使用 User Operator 的情况下发布应用程序证书，则此 secret 必须可以被 Apache Kafka 组件的流访问。如果需要，您可以使用 OpenShift 基于角色的访问控制来执行此操作。

表 16.10. < *cluster_name*>-clients-ca secret 中的字段

字段	Description
ca.key	客户端 CA 的当前私钥。

表 16.11. < *cluster_name*>-clients-ca-cert secret 中的字段

字段	Description
ca.p12	用于存储证书和密钥的 PKCS #12 存储。
ca.password	用于保护 PKCS #12 存储的密码。
ca.crt	客户端 CA 的当前证书。

16.2.5. User Operator 生成的用户 secret

用户 secret 由 User Operator 管理。

使用 User Operator 创建用户时，会使用用户名称生成 secret。

表 16.12. *user_name* secret 中的字段

Secret 名称	secret 中的字段	Description
<user_name>	**user.p12**	用于存储证书和密钥的 PKCS #12 存储。
	user.password	用于保护 PKCS #12 存储的密码。
	user.crt	用户的证书，由客户端 CA 签名
	user.key	用户的私钥

16.2.6. 在集群 CA secret 中添加标签和注解

通过在 Kafka 自定义资源中配置 clusterCaCert 模板属性，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 Streams 中为 Apache Kafka 自定义资源配置模板属性。

模板自定义示例，将标签和注解添加到 secret

```
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
    # ...
```

16.2.7. 在 CA secret 中禁用 ownerReference

默认情况下，集群和客户端 CA secret 使用设置为 Kafka 自定义资源的 ownerReference 属性创建。这意味着，当 Kafka 自定义资源被删除时，OpenShift 也会删除 CA secret （收集的 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 generateSecretOwnerReference 属性设置为 false 来禁用 ownerReference。当禁用 ownerReference 时，当删除对应的 Kafka 自定义资源时，OpenShift 不会删除 CA secret。

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
# ...
```

其他资源

- certificateAuthority 模式参考

16.3. 证书续订和有效期

集群 CA 和客户端 CA 证书只在有限的时间段内有效，称为有效期。这通常定义为生成证书的天数。

对于 Cluster Operator 自动创建的 CA 证书，您可以配置以下有效周期：

- 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不接受证书由 CA 私钥签名的对等点的连接。组件和客户端需要信任*新的* CA 证书。

要允许在不丢失服务的情况下续订 CA 证书，Cluster Operator 会在旧的 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周期为 30 天。

续订周期从当前证书的到期日期开始向后测量。

针对续订周期的有效期

```
Not Before                                Not After
   |                                          |
   |<---------------- validityDays ---------------->|
                         <--- renewalDays --->|
```

要在创建 Kafka 集群后更改有效期和续订周期，请配置并应用 Kafka 自定义资源，并 手动续订 CA 证书。如果您没有手动续订证书，下次自动更新证书时将使用新周期。

证书有效期和续订周期的 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 的行为取决于集群 CA 和客户端 CA 的 generateCertificateAuthority 证书生成属性的设置。

true

如果属性设为 true，Cluster Operator 会自动生成 CA 证书，并在续订周期内自动更新。

false

如果属性设为 false，Cluster Operator 不会生成 CA 证书。如果您要安装自己的证书，则使用此选项。

16.3.1. 使用自动生成的 CA 证书的续订过程

Cluster Operator 在续订 CA 证书时按照以下顺序执行以下进程：

1. 生成新的 CA 证书，但保留现有的密钥。

   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）。

   这不是必要的，因为签名密钥没有改变，而是保留客户端证书的有效性周期与 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。

   用户证书由客户端 CA 签名。当客户端 CA 续订时，由 User Operator 生成的用户证书会被续订。

16.3.2. 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。

当连接到集群时，并确保它们正确运行，客户端应用程序必须：

- 信任在 <cluster> - *cluster*-ca-cert Secret 中发布的集群 CA 证书。
- 使用其 < *user-name&gt*; Secret 中发布的凭证来连接到集群。

  User Secret 以 PEM 和 PKCS #12 格式提供凭证，或者在使用 SCRAM-SHA 身份验证时提供密码。User Operator 在创建用户时创建用户凭证。

您必须确保在证书续订后客户端继续工作。续订过程取决于客户端的配置方式。

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在续订期间客户端使用新证书。在续订周期结束时无法执行此操作可能会导致客户端应用程序无法连接到集群。

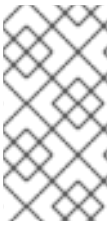
注意

对于在同一 OpenShift 集群和命名空间中运行的工作负载，Secret 可以挂载为卷，以便客户端 Pod 从机密的当前状态构造其密钥存储和信任存储。有关此过程的详情，请参阅配置内部客户端以信任集群 CA。

16.3.3. 手动续订 Cluster Operator 管理的 CA 证书

Cluster Operator 在相应的证书续订期间开始时自动续订的集群和客户端 CA 证书。但是，您可以使用 strimzi.io/force-renew 注解在证书续订期开始前手动续订这些证书的一个或多个证书。出于安全原因，您可能这样做，或者更改了证书的续订或有效期周期。

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

- 必须部署 Cluster Operator。
- 安装 CA 证书和私钥的 Kafka 集群。
- OpenSSL TLS 管理工具，用于检查 CA 证书的有效性周期。

在此过程中，我们在 my-project 命名空间中使用名为 my-cluster 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。

续订集群 CA secret

```
oc annotate secret my-cluster-cluster-ca-cert -n my-project strimzi.io/force-renew="true"
```

续订客户端 CA secret

```
oc annotate secret my-cluster-clients-ca-cert -n my-project strimzi.io/force-renew="true"
```

2. 在下一个协调时，Cluster Operator 会生成新证书。

如果配置了维护时间窗，Cluster Operator 会在下一次维护时间窗内第一次协调时生成新的 CA 证书。

3. 检查新 CA 证书的有效性周期。

检查新集群 CA 证书的有效性周期

```
oc get secret my-cluster-cluster-ca-cert -n my-project -o=jsonpath='{.data.ca\.crt}' |
base64 -d | openssl x509 -noout -dates
```

检查新客户端 CA 证书的有效性周期

```
oc get secret my-cluster-clients-ca-cert -n my-project -o=jsonpath='{.data.ca\.crt}' |
base64 -d | openssl x509 -noout -dates
```

该命令返回一个 notBefore 和 notAfter date，这是 CA 证书的有效开始和结束日期。

4. 更新客户端配置以信任新的集群 CA 证书。

请参阅：

- 第 16.4 节 “配置内部客户端以信任集群 CA”
- 第 16.5 节 “配置外部客户端以信任集群 CA”

16.3.4. 手动从已过期的 Cluster Operator 管理的 CA 证书中恢复

Cluster Operator 会在续订周期开始时自动更新集群和客户端 CA 证书。然而，意外的操作问题或中断可能会阻止续订过程，如 Cluster Operator 的停机时间或 Kafka 集群不可用。如果 CA 证书过期，Kafka 集群组件无法相互通信，Cluster Operator 在无需人工干预的情况下无法续订 CA 证书。

要立即执行恢复，请按照给出的顺序操作此流程中所述的步骤。您可以从过期的集群和客户端 CA 证书中恢复。此过程涉及删除包含过期证书的 secret，以便 Cluster Operator 生成新证书的 secret。有关 Apache Kafka 中流管理的 secret 的更多信息，请参阅 第 16.2.2 节 “Cluster Operator 生成的 secret”。

注意

如果您使用自己的 CA 证书并使其过期，则该过程类似，但您需要 续订 CA 证书，而不是使用 Cluster Operator 生成的证书。

先决条件

- 必须部署 Cluster Operator。
- 安装 CA 证书和私钥的 Kafka 集群。
- OpenSSL TLS 管理工具，用于检查 CA 证书的有效性周期。

在此过程中，我们在 my-project 命名空间中使用名为 my-cluster 的 Kafka 集群。

流程

1. 删除包含过期 CA 证书的 secret。

删除 Cluster CA secret

```
oc delete secret my-cluster-cluster-ca-cert -n my-project
```

删除 Clients CA secret

```
oc delete secret my-cluster-clients-ca-cert -n my-project
```

2. 等待 Cluster Operator 生成新证书。

- 新的 CA 集群证书以验证 Kafka 代理的身份在相同名称的 secret 中创建了(my-cluster-cluster-ca-cert)。
- 在相同名称的 secret 中创建了用于验证 Kafka 用户身份的新 CA 客户端证书(my-cluster-clients-ca-cert)。

3. 检查新 CA 证书的有效性周期。

检查新集群 CA 证书的有效性周期

```
oc get secret my-cluster-cluster-ca-cert -n my-project -o=jsonpath='{.data.ca\.crt}' |
base64 -d | openssl x509 -noout -dates
```

检查新客户端 CA 证书的有效性周期

```
oc get secret my-cluster-clients-ca-cert -n my-project -o=jsonpath='{.data.ca\.crt}' |
base64 -d | openssl x509 -noout -dates
```

该命令返回一个 notBefore 和 notAfter date，这是 CA 证书的有效开始和结束日期。

4. 删除使用 CA 证书的组件 pod 和 secret。

a. 删除 ZooKeeper secret。

b. 等待 Cluster Operator 检测缺少的 ZooKeeper secret 并重新创建它。

c. 删除所有 ZooKeeper pod。

d. 删除 Kafka secret。

e. 等待 Cluster Operator 检测缺少的 Kafka secret 并重新创建它。

f. 删除所有 Kafka pod。

如果您只恢复客户端 CA 证书，则只需要删除 Kafka secret 和 pod。

您可以使用以下 oc 命令查找资源，并验证它是否已被删除。

```
oc get <resource_type> --all-namespaces | grep <kafka_cluster_name>
```

将 <resource_type > 替换为资源的类型，如 Pod 或 Secret。

5. 等待 Cluster Operator 检测缺少的 Kafka 和 ZooKeeper pod，并使用更新的 CA 证书重新创建它们。

在协调时，Cluster Operator 会自动更新其他组件以信任新的 CA 证书。

6. 验证 Cluster Operator 日志中没有与证书验证相关的问题。

7. 更新客户端配置以信任新的集群 CA 证书。

请参阅：

- 第 16.4 节 “配置内部客户端以信任集群 CA”

- 第 16.5 节 “配置外部客户端以信任集群 CA”

16.3.5. 替换 Cluster Operator 管理的 CA 证书使用的私钥

您可以替换由 Cluster Operator 生成的集群 CA 和客户端 CA 证书使用的私钥。当替换私钥时，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更新的私钥的 Secret。

表 16.13. 替换私钥的命令

私钥，用于	Secret	注解命令
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"**

在下一次协调时，Cluster Operator 将：

- 为您注解的 Secret 生成新私钥
- 生成一个新的 CA 证书

如果配置了维护时间窗，Cluster Operator 会在下一次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 Cluster Operator 更新的集群和客户端 CA 证书。

其他资源

- 第 16.2 节 “由 Operator 生成的 secret”
- 第 29.1 节 “滚动更新的维护时间窗”

16.4. 配置内部客户端以信任集群 CA

此流程描述了如何配置位于 OpenShift 集群内的 Kafka 客户端 - 连接到 TLS 侦听器 - 信任集群 CA 证书。

为内部客户端实现此目的的最简单方法是，使用卷挂载来访问 包含所需 证书和密钥的 Secret。

按照以下步骤为基于 Java 的 Kafka Producer、Consumer 和 Streams API 配置由集群 CA 签名的信任证书。

根据集群 CA 的证书格式，选择遵循的步骤：PKCS #12 (.p12) 或 PEM (.crt)。

步骤描述了如何挂载 Cluster Secret，将 Kafka 集群的身份验证到客户端 pod。

先决条件

- Cluster Operator 必须正在运行。

- OpenShift 集群中需要有一个 Kafka 资源。
- 您需要在 OpenShift 集群中使用 TLS 进行连接的 Kafka 客户端应用程序，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，我们挂载以下内容：

   - PKCS #12 会进入一个准确的路径中，这可以进行配置
   - 密码进入环境变量中，可用于 Java 配置

2. 使用以下属性配置 Kafka 客户端：

- 安全协议选项：
  - security.protocol ：当使用 TLS 加密时（带有或没有 mTLS 验证）时 SSL。
  - security.protocol：通过 TLS 使用 SCRAM-SHA 身份验证时 SASL_SSL。
- ssl.truststore.location，使用导入证书的信任存储位置。
- ssl.truststore.password，使用用于访问 truststore 的密码。
- ssl.truststore.type=PKCS12 来识别信任存储类型。

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. 使用提取的证书，以 X.509 格式使用证书的客户端配置 TLS 连接。

16.5. 配置外部客户端以信任集群 CA

此流程描述了如何配置位于 OpenShift 集群外的 Kafka 客户端 - 连接到 外部监听程序 - 信任集群 CA 证书。在替换旧客户端 CA 证书时，请在设置客户端和续订期间按照以下步骤操作。

按照以下步骤为基于 Java 的 Kafka Producer、Consumer 和 Streams API 配置由集群 CA 签名的信任证书。

根据集群 CA 的证书格式，选择遵循的步骤：PKCS #12 (.p12) 或 PEM (.crt)。

步骤描述了如何从 Cluster Secret 获取证书，以验证 Kafka 集群的身份。

重要

& *lt;cluster_name*> -cluster-ca-cert secret 在 CA 证书续订期间包含多个 CA 证书。客户端必须将它们*全部*添加到其信任存储中。

先决条件

- Cluster Operator 必须正在运行。
- OpenShift 集群中需要有一个 Kafka 资源。
- 您需要一个 Kafka 客户端应用程序，该应用程序将使用 TLS 进行连接，并且需要信任集群 CA 证书。

使用 PKCS #12 格式 (.p12)

1. 从 Kafka 集群的 < *cluster_name*> -cluster-ca-cert Secret 中提取集群 CA 证书和密钥。

```
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
```

将 *<cluster_name>* 替换为 Kafka 集群的名称。

2. 使用以下属性配置 Kafka 客户端：

- 安全协议选项：
  - security.protocol ：使用 TLS 时 SSL。
  - security.protocol：通过 TLS 使用 SCRAM-SHA 身份验证时 SASL_SSL。
- ssl.truststore.location，使用导入证书的信任存储位置。
- ssl.truststore.password，使用用于访问 truststore 的密码。如果信任存储不需要，则可以省略此属性。
- ssl.truststore.type=PKCS12 来识别信任存储类型。

使用 PEM 格式(.crt)

1. 从 Kafka 集群的 &lt ;*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. 使用提取的证书，以 X.509 格式使用证书的客户端配置 TLS 连接。

16.6. 使用您自己的 CA 证书和私钥

安装和使用您自己的 CA 证书和私钥，而不是使用 Cluster Operator 生成的默认值。您可以替换集群和客户端 CA 证书和私钥。

您可以使用以下方法切换到使用您自己的 CA 证书和私钥：

- 在部署 Kafka 集群前安装您自己的 CA 证书和私钥
- 在部署 Kafka 集群后，将默认 CA 证书和私钥替换为您自己的默认 CA 证书和私钥

在部署 Kafka 集群后替换默认 CA 证书和私钥的步骤与用来更新您自己的 CA 证书和私钥的步骤相同。

如果您使用自己的证书，则不会自动更新证书。您需要在 CA 证书和私钥过期前续订它们。

续订选项：

- 仅续订 CA 证书
- 续订 CA 证书和私钥（或替换默认值）

16.6.1. 安装您自己的 CA 证书和私钥

安装您自己的 CA 证书和私钥，而不是使用 Cluster Operator 生成的集群和客户端 CA 证书和私钥。

默认情况下，Apache Kafka 的 Streams 使用以下 集群 CA 和客户端 CA secret，这些 secret 会自动续订。

- 集群 CA secret
  - <cluster_name>-cluster-ca
  - <cluster_name>-cluster-ca-cert
- 客户端 CA secret

- <cluster_name>-clients-ca
- <cluster_name>-clients-ca-cert

要安装您自己的证书，请使用相同的名称。

先决条件

- Cluster Operator 正在运行。
- Kafka 集群还没有部署。

  如果您已经部署了 Kafka 集群，您可以将 默认 CA 证书替换为您自己的。
- 集群 CA 或客户端 CA 的 PEM 格式的 X.509 证书和密钥。
  - 如果要使用不是 Root CA 的集群或客户端 CA，则必须在证书文件中包含整个链。链应按以下顺序：
    1. 集群或客户端 CA
    2. 一个或多个中间 CA
    3. root CA
  - 链中的所有 CA 都应该使用 X509v3 Basic Constraints 扩展进行配置。基本约束限制证书链的路径长度。
- 用于转换证书的 OpenSSL TLS 管理工具。

开始前

Cluster Operator 以 PEM (Privacy Enhanced Mail) 和 PKCS #12 (Public-Key Cryptography Standards) 格式生成密钥和证书。您可以以任一格式添加自己的证书。

有些应用程序无法使用 PEM 证书，且只支持 PKCS #12 证书。如果您没有 PKCS #12 格式的集群证书，请使用 OpenSSL TLS 管理工具从 ca.crt 文件中生成一个。

certificate 生成命令示例

```
openssl pkcs12 -export -in ca.crt -nokeys -out ca.p12 -password pass:<P12_password> -
caname ca.crt
```

将 <P12_password> 替换为您自己的密码。

流程

1. 创建包含 CA 证书的新 secret。

仅使用 PEM 格式的证书创建客户端 secret

```
oc create secret generic <cluster_name>-clients-ca-cert --from-file=ca.crt=ca.crt
```

使用 PEM 和 PKCS #12 格式的证书创建集群 secret

```
oc create secret generic <cluster_name>-cluster-ca-cert \
  --from-file=ca.crt=ca.crt \
  --from-file=ca.p12=ca.p12 \
  --from-literal=ca.password=P12-PASSWORD
```

将 <cluster_name> 替换为 Kafka 集群的名称。

2. 创建一个包含私钥的新 secret。

```
oc create secret generic <ca_key_secret> --from-file=ca.key=ca.key
```

3. 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>"
```

- 标签 strimzi.io/kind=Kafka 标识 Kafka 自定义资源。
- 标签 strimzi.io/cluster="<cluster_name>" 标识 Kafka 集群。

4. 注解 secret

```
oc annotate secret <ca_certificate_secret> strimzi.io/ca-cert-generation="
<ca_certificate_generation>"
```

```
oc annotate secret <ca_key_secret> strimzi.io/ca-key-generation="
<ca_key_generation>"
```

- 注解 strimzi.io/ca-cert-generation="<ca_certificate_generation>" 定义新 CA 证书的生成。
- 注解 strimzi.io/ca-key-generation="<ca_key_generation>" 定义新 CA 密钥的生成。

从 0 （零）作为您自己的 CA 证书的增量值(strimzi.io/ca-cert-generation=0)开始。在续订证书时设置更高的增量值。

5. 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。

配置集群 CA 的片段 Kafka 资源示例，以使用您自己提供的证书

```
kind: Kafka
version: kafka.strimzi.io/v1beta2
spec:
  # ...
  clusterCa:
    generateCertificateAuthority: false
```

其他资源

- 第 16.6.2 节 “续订您自己的 CA 证书”
- 第 16.6.3 节 “自行续订或替换 CA 证书和私钥”
- 第 15.3.4 节 “为 TLS 加密提供自己的 Kafka 侦听器证书”

16.6.2. 续订您自己的 CA 证书

如果您使用自己的 CA 证书，则需要手动续订。Cluster Operator 不会自动续订。在续订期间内续订 CA 证书，然后再过期。

当您更新 CA 证书并继续同一私钥时，执行此流程中的步骤。如果您要续订您自己的 CA *证书和私钥*，请参阅 第 16.6.3 节 “自行续订或替换 CA 证书和私钥”。

该流程描述了以 PEM 格式续订 CA 证书。

先决条件

- Cluster Operator 正在运行。

- 您有 PEM 格式的新集群或客户端 X.509 证书。

流程

1. 更新 CA 证书的 Secret。

   编辑现有 secret 以添加新的 CA 证书并更新证书生成注解值。

```
oc edit secret <ca_certificate_secret_name>
```

   <ca_certificate_secret_name > 是 Secret 的名称，即 &lt *;kafka_cluster_name&gt*; -cluster-ca-cert 用于集群 CA 证书，& *lt;kafka_cluster_name&*gt; -clients-ca-cert 用于客户端 CA 证书。

   以下示例显示了与名为 my-cluster 的 Kafka 集群关联的集群 CA 证书的 secret。

集群 CA 证书的 secret 配置示例

```
apiVersion: v1
kind: Secret
data:
  ca.crt: LS0tLS1CRUdJTiBDRVJUSUZJQ0F... 1
metadata:
  annotations:
    strimzi.io/ca-cert-generation: "0" 2
  labels:
    strimzi.io/cluster: my-cluster
    strimzi.io/kind: Kafka
  name: my-cluster-cluster-ca-cert
  #...
type: Opaque
```

1. 当前 base64 编码的 CA 证书

2.

当前 CA 证书生成注解值

2. 将新 CA 证书编码为 base64。

```
cat <path_to_new_certificate> | base64
```

3. 更新 CA 证书。

将上一步中 base64 编码的 CA 证书复制为 数据 下的 ca.crt 属性的值。

4. 增加 CA 证书生成注解的值。

使用更高的增量值更新 strimzi.io/ca-cert-generation 注解。例如，将 strimzi.io/ca-cert-generation=0 更改为 strimzi.io/ca-cert-generation=1。如果 Secret 缺少注解，则该值将被视为 0， 因此添加值为 1 的注解。

当 Apache Kafka 的 Streams 生成证书时，Cluster Operator 会自动递增证书生成注解。对于您自己的 CA 证书，使用更高的增量值设置注解。该注解需要高于当前 secret 的值，以便 Cluster Operator 可以推出 pod 并更新证书。每个 CA 证书续订都必须递增 strimzi.io/ca-cert-generation。

5. 使用新的 CA 证书和证书生成注解值保存 secret。

使用新 CA 证书更新的 secret 配置示例

```
apiVersion: v1
kind: Secret
data:
  ca.crt: GCa6LS3RTHeKFiFDGBOUDYFAZ0F... 1
metadata:
  annotations:
    strimzi.io/ca-cert-generation: "1" 2
  labels:
    strimzi.io/cluster: my-cluster
    strimzi.io/kind: Kafka
  name: my-cluster-cluster-ca-cert
  #...
type: Opaque
```

1 新的 base64 编码的 CA 证书

2 新的 CA 证书生成注解值

在下一个协调中，Cluster Operator 对 ZooKeeper、Kafka 和其他组件执行滚动更新来信任新的 CA 证书。

如果配置了维护时间窗，Cluster Operator 会在下一次维护时间窗内第一次协调时滚动 pod。

16.6.3. 自行续订或替换 CA 证书和私钥

如果您使用自己的 CA 证书和私钥，则需要手动续订它们。Cluster Operator 不会自动续订。在续订期间内续订 CA 证书，然后再过期。您还可以使用相同的流程，将由 Apache Kafka operator Streams 生成的 CA 证书和私钥替换为您自己的 Apache Kafka operator。

当您续订或替换 CA 证书和私钥时，执行此流程中的步骤。如果您只是续订自己的 CA 证书，请参阅 第 16.6.2 节 “续订您自己的 CA 证书”。

该流程描述了以 PEM 格式续订 CA 证书和私钥。

在执行以下步骤前，请确保新 CA 证书的 CN (Common Name)与当前 CA 证书不同。例如，当 Cluster Operator 自动更新证书时，它会添加 *v<version_number& gt;* 后缀来识别版本。通过在每个续订中添加不同的后缀，使用您自己的 CA 证书执行相同的操作。通过使用其他密钥生成新的 CA 证书，您可以保留存储在 Secret 中的当前 CA 证书。

先决条件

- Cluster Operator 正在运行。

- 您有 PEM 格式的新集群或客户端 X.509 证书和密钥。

流程

1. 暂停 Kafka 自定义资源的协调。

   a. 在 OpenShift 中注解自定义资源，将 pause-reconciliation 注解设置为 true ：

   ```
   oc annotate Kafka <name_of_custom_resource> strimzi.io/pause-reconciliation="true"
   ```

   例如，对于名为 my-cluster 的 Kafka 自定义资源：

   ```
   oc annotate Kafka my-cluster strimzi.io/pause-reconciliation="true"
   ```

   b. 检查自定义资源的状态条件是否显示 ReconciliationPaused 的更改：

   ```
   oc describe Kafka <name_of_custom_resource>
   ```

   在 lastTransitionTime 中 type 条件会变为 ReconciliationPaused。

2. 检查 Kafka 自定义资源中的 generateCertificateAuthority 属性的设置。

   如果属性设为 false，Cluster Operator 不会生成 CA 证书。如果您使用自己的证书，则需要此设置。

3. 如果需要，编辑现有 Kafka 自定义资源，并将 generateCertificateAuthority 属性设置为 false。

   ```
   oc edit Kafka <name_of_custom_resource>
   ```

   以下示例显示了一个 Kafka 自定义资源，其集群和客户端 CA 证书生成委派给用户。

使用您自己的 CA 证书的 Kafka 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
# ...
spec:
# ...
  clusterCa:
    generateCertificateAuthority: false 1
  clientsCa:
    generateCertificateAuthority: false 2
# ...
```

1 使用您自己的集群 CA

2 使用您自己的客户端 CA

4. 更新 CA 证书的 Secret。

   a. 编辑现有 secret 以添加新的 CA 证书并更新证书生成注解值。

   ```
   oc edit secret <ca_certificate_secret_name>
   ```

   <ca_certificate_secret_name> 是 Secret 的名称，即集群 CA 证书的 <kafka_cluster_name>-cluster-ca-cert，客户端 CA 证书的 <kafka_cluster_name>-clients-ca-cert。

   以下示例显示了与名为 my-cluster 的 Kafka 集群关联的集群 CA 证书的 secret。

   集群 CA 证书的 secret 配置示例

   ```
   apiVersion: v1
   kind: Secret
   ```

```
data:
  ca.crt: LS0tLS1CRUdJTiBDRVJUSUZJQ0F... 1
metadata:
  annotations:
    strimzi.io/ca-cert-generation: "0" 2
  labels:
    strimzi.io/cluster: my-cluster
    strimzi.io/kind: Kafka
  name: my-cluster-cluster-ca-cert
  #...
type: Opaque
```

1 当前 base64 编码的 CA 证书

2 当前 CA 证书生成注解值

b. 重命名当前的 CA 证书以保留它。

将数据 下的当前 ca.crt 属性重命名为 ca-<date>.crt，其中 <date> 是 *YEAR-MONTH-DAYTHOUR-MINUTE-SECONDZ* 格式的证书到期日期。例如 ca-2023-01-26T17-32-00Z.crt：保留 属性的值，因为它是保留当前的 CA 证书。

c. 将新 CA 证书编码为 base64。

```
cat <path_to_new_certificate> | base64
```

d. 更新 CA 证书。

在 data 下创建一个新的 ca.crt 属性，并将上一步中的 base64 编码的 CA 证书复制为 ca.crt 属性的值。

e. 增加 CA 证书生成注解的值。

使用更高的增量值更新 strimzi.io/ca-cert-generation 注解。例如，将 strimzi.io/ca-cert-generation=0 更改为 strimzi.io/ca-cert-generation=1。如果 Secret 缺少注解，则该

值将被视为 0， 因此添加值为 1 的注解。

当 Apache Kafka 的 Streams 生成证书时，Cluster Operator 会自动递增证书生成注解。对于您自己的 CA 证书，使用更高的增量值设置注解。该注解需要高于当前 secret 的值，以便 Cluster Operator 可以推出 pod 并更新证书。每个 CA 证书续订都必须递增 strimzi.io/ca-cert-generation。

f. 使用新的 CA 证书和证书生成注解值保存 secret。

使用新 CA 证书更新的 secret 配置示例

```
apiVersion: v1
kind: Secret
data:
  ca.crt: GCa6LS3RTHeKFiFDGBOUDYFAZ0F... 1
  ca-2023-01-26T17-32-00Z.crt: LS0tLS1CRUdJTiBDRVJUSUZJQ0F... 2
metadata:
  annotations:
    strimzi.io/ca-cert-generation: "1" 3
  labels:
    strimzi.io/cluster: my-cluster
    strimzi.io/kind: Kafka
  name: my-cluster-cluster-ca-cert
  #...
type: Opaque
```

1 新的 base64 编码的 CA 证书

2 旧的 base64 编码的 CA 证书

3 新的 CA 证书生成注解值

5.

更新用于为新 CA 证书签名的 CA 密钥的 Secret。

a.

编辑现有的 secret，以添加新的 CA 密钥并更新密钥生成注解值。

```
oc edit secret <ca_key_name>
```

<ca_key_name> 是 CA 密钥的名称，对于集群 CA 密钥是 < kafka_cluster_name>-cluster-ca，客户端 CA 密钥是 < kafka_cluster_name>-clients-ca。

以下示例显示了与名为 my-cluster 的 Kafka 集群关联的集群 CA 密钥的 secret。

集群 CA 密钥的 secret 配置示例

```
apiVersion: v1
kind: Secret
data:
  ca.key: SA1cKF1GFDzOIiPOIUQBHDNFGDFS... (1)
metadata:
  annotations:
    strimzi.io/ca-key-generation: "0" (2)
  labels:
    strimzi.io/cluster: my-cluster
    strimzi.io/kind: Kafka
  name: my-cluster-cluster-ca
  #...
type: Opaque
```

(1) 当前 base64 编码的 CA 密钥

(2) 当前 CA 密钥生成注解值

b.

将 CA 密钥编码为 base64。

```
cat <path_to_new_key> | base64
```

c. 更新 CA 密钥。

将上一步中 base64 编码的 CA 密钥复制为 data 下的 ca.key 属性的值。

d. 增加 CA 密钥生成注解的值。

使用更高的增量值更新 strimzi.io/ca-key-generation 注解。例如，将 strimzi.io/ca-key-generation=0 更改为 strimzi.io/ca-key-generation=1。如果 Secret 缺少注解，它将被视为 0， 因此添加值为 1 的注解。

当 Apache Kafka 的 Streams 生成证书时，Cluster Operator 会自动递增密钥生成注解。对于您自己的 CA 证书以及新的 CA 密钥，将注解设置为更高的增量值。该注解需要高于当前 secret 的值，以便 Cluster Operator 可以滚动 pod 并更新证书和密钥。每个 CA 证书续订都必须递增 strimzi.io/ca-key-generation。

e. 使用新的 CA 密钥和密钥生成注解值保存 secret。

使用新 CA 密钥更新的 secret 配置示例

```
apiVersion: v1
kind: Secret
data:
  ca.key: AB0cKF1GFDzOIiPOIUQWERZJQ0F... 1
metadata:
  annotations:
    strimzi.io/ca-key-generation: "1" 2
  labels:
    strimzi.io/cluster: my-cluster
    strimzi.io/kind: Kafka
  name: my-cluster-cluster-ca
  #...
type: Opaque
```

1

2

6. 从暂停恢复。

要恢复 Kafka 自定义资源协调，请将 pause-reconciliation 注解设置为 false。

```
oc annotate --overwrite Kafka <name_of_custom_resource> strimzi.io/pause-reconciliation="false"
```

您还可以通过删除 pause-reconciliation 注解来执行相同的操作。

```
oc annotate Kafka <name_of_custom_resource> strimzi.io/pause-reconciliation-
```

在下一个协调中，Cluster Operator 对 ZooKeeper、Kafka 和其他组件执行滚动更新来信任新的 CA 证书。滚动更新完成后，Cluster Operator 将启动一个新服务器证书，以生成由新 CA 密钥签名的新服务器证书。

如果配置了维护时间窗，Cluster Operator 会在下一次维护时间窗内第一次协调时滚动 pod。

7. 等待滚动更新移至新的 CA 证书。

8. 从 secret 配置中删除任何过时的证书，以确保集群不再信任它们。

```
oc edit secret <ca_certificate_secret_name>
```

删除旧证书的 secret 配置示例

```
apiVersion: v1
kind: Secret
data:
  ca.crt: GCa6LS3RTHeKFiFDGBOUDYFAZ0F...
metadata:
  annotations:
    strimzi.io/ca-cert-generation: "1"
```

```
  labels:
    strimzi.io/cluster: my-cluster
    strimzi.io/kind: Kafka
  name: my-cluster-cluster-ca-cert
  #...
type: Opaque
```

9. 启动集群的手动滚动更新，以获取对 secret 配置所做的更改。

   请参阅 第 29.2 节 “使用注解启动 Kafka 和其他操作对象的滚动更新”。

第 17 章 将安全上下文应用到 APACHE KAFKA POD 和容器的流

安全性上下文约束定义 pod 和容器的限制。通过指定安全上下文，Pod 和容器仅具有所需的权限。例如，权限可以控制运行时操作或对资源的访问。

17.1. OPENSHIFT 平台处理安全上下文

处理安全上下文取决于您使用的 OpenShift 平台的工具。

例如，OpenShift 使用内置的安全性上下文约束(SCC)来控制权限。SCC 是控制 Pod 能够访问的安全功能的设置和策略。

默认情况下，OpenShift 会自动注入安全上下文配置。在大多数情况下，这意味着您不需要为 Cluster Operator 创建的 pod 和容器配置安全上下文。虽然您仍然可以创建和管理自己的 SCC。

如需更多信息，请参阅 OpenShift 文档。

第 18 章 通过添加或删除代理来扩展集群

通过添加代理来扩展 Kafka 集群，可以提高集群的性能和可靠性。添加更多代理会增加可用资源，允许集群处理更大的工作负载并处理更多信息。它还可以通过提供更多副本和备份来提高容错。相反，删除使用率不足的代理可减少资源消耗并提高效率。必须仔细执行扩展，以避免中断或数据丢失。通过在集群中的所有代理间重新分发分区，每个代理的资源使用率会减少，这可以提高集群的整体吞吐量。

注意

要增加 Kafka 主题的吞吐量，您可以增加该主题的分区数量。这允许在集群中的不同代理之间共享主题的负载。但是，如果每个代理受特定资源（如 I/O）的限制，则添加更多分区不会增加吞吐量。在这种情况下，您需要在集群中添加更多代理。

调整 Kafka.spec.kafka.replicas 配置会影响作为副本的集群中代理数量。主题的实际复制因素由 default.replication.factor 和 min.insync.replicas 的设置决定，以及可用代理的数量。例如，一个复制因素为 3 表示主题的每个分区在三个代理之间复制，确保在代理失败时容错。

副本配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3
    # ...
  config:
      # ...
      default.replication.factor: 3
      min.insync.replicas: 2
 # ...
```

当通过 Kafka 资源配置添加代理时，节点 ID 以 0 （零）开始，Cluster Operator 会为新节点分配下一个最低 ID。代理删除过程从集群中 ID 的最高代理 pod 开始。

如果您要使用节点池管理集群中的节点，请调整 KafkaNodePool.spec.replicas 配置以更改节点池中的节点数。另外，在使用节点池扩展现有集群时，您可以为 扩展操作分配节点 ID。

当您添加或删除代理时，Kafka 不会自动重新分配分区。执行此操作的最佳方法是使用 Cruise Control。在扩展集群或缩减时，您可以使用 Cruise Control 的 add-brokers 和 remove-brokers 模式。

- 在扩展 Kafka 集群后，使用 add-brokers 模式，将现有代理中的分区副本移到新添加的代理中。
- 在缩减 Kafka 集群前，使用 remove-brokers 模式，将分区副本移出要删除的代理。

18.1. 在缩减操作中跳过检查

默认情况下，Apache Kafka 的 Streams 会执行检查，以确保在 Kafka 集群上启动缩减操作前，代理中没有分区副本。检查适用于仅执行代理角色或代理和控制器双角色的节点。

如果找到副本，则不会进行缩减，以防止潜在的数据丢失。要缩减集群，在尝试再次缩减前，代理不能保留任何副本。

然而，在有些情况下，您可能想要绕过此机制。例如，在忙碌的集群中禁用检查可能是必需的，例如，因为新主题为代理生成副本。这种情况可能会无限期地阻止缩减，即使代理几乎为空。以这种方式覆盖阻塞机制有影响：有关缩减代理的主题存在可能会导致 Kafka 集群协调失败。

您可以通过注解 Kafka 集群的 Kafka 资源来绕过阻塞机制。通过将 strimzi.io/skip-broker-scaledown-check 注解设置为 true 来注解资源：

添加注解以跳过缩减操作上的检查

```
oc annotate Kafka my-kafka-cluster strimzi.io/skip-broker-scaledown-check="true"
```

此注解指示 Apache Kafka 的 Streams 跳过 scale-down 检查。将 my-kafka-cluster 替换为特定 Kafka 资源的名称。

要恢复缩减操作的检查，请删除注解：

删除注解以跳过缩减操作的检查

```
oc annotate Kafka my-kafka-cluster strimzi.io/skip-broker-scaledown-check-
```

第 19 章 使用 CRUISE CONTROL 重新平衡集群

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

- 监控集群工作负载
- 根据预定义的限制重新平衡集群

运行更均衡的 Kafka 集群来帮助更有效地使用代理 pod。

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

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

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

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

注意

Apache Kafka 的流提供了 Cruise Control 的示例配置文件。

19.1. CRUISE CONTROL 组件和功能

Cruise Control 包括四个主要组件 - Load Monitor、Anomaly Detector 和 Executor- 以及用于客户端交互的 REST API。Apache Kafka 的流使用 REST API 来支持以下 Cruise Control 功能：

- 从优化目标生成优化方案。

- 根据优化提议，重新平衡 Kafka 集群。

优化目标

优化目标描述了从重新平衡实现的特定目标。例如，目标可能是在代理之间更均匀地分布主题副本。您可通过配置更改要包含的目标。目标定义为一个硬目标或软目标。您可以通过 Cruise Control 部署配置添加硬目标。您还具有适合这些类别的主要、默认和用户提供的目标。

- 硬 目标是预先设置的，必须满足优化建议才能成功进行。

- 为了成功优化提议，无需满足软目标。如果表示满足所有硬目标，则可以设置它们。

- 主要目标从 Cruise Control 中继承。其中一些预先设置为硬目标。默认在优化提议中使用主要目标。

- 默认情况下，默认目标与主目标相同。您可以指定您自己的一组默认目标。

- 用户提供的目标是 配置用于生成特定优化提议的默认目标子集。

优化提议

优化建议由您要从重新平衡实现的目标组成。您可以生成优化建议，以创建提议的更改概述，以及重新平衡的结果。目标以特定优先级顺序进行评估。然后您可以选择批准或拒绝提议。您可以使用调整的目标集合拒绝再次运行它。

您可以在三种模式之一中生成优化方案。

- full 是默认模式，运行完整重新平衡。

- add-brokers 是扩展 Kafka 集群时添加代理后使用的模式。

- remove-brokers 是缩减 Kafka 集群时删除代理之前使用的模式。

目前还不支持其他 Cruise Control 功能，包括自我修复、通知、编写目标以及更改主题复制因素。

其他资源

- Cruise Control 文档

19.2. 优化目标概述

优化目标是在 Kafka 集群中重新发布工作负载和资源利用率的限制。要重新平衡 Kafka 集群，Cruise Control 使用优化目标来 生成优化建议，您可以批准或拒绝。

19.2.1. 优先级的目标顺序

Apache Kafka 的流支持在 Cruise Control 项目中开发的大多数优化目标。支持的目标（以优先级默认降序排列）如下：

1. 机架感知性
2. 每个代理对一组主题的最小领导副本数
3. 副本容量
4. 容量目标
   - 磁盘容量
   - 网络入站容量
   - 网络出站容量

- **CPU 容量**

5. **副本分发**
6. **潜在的网络输出**
7. **资源分布目标**
   - **磁盘使用分布**
   - **网络入站使用分布**
   - **网络出站使用分布**
   - **CPU 使用率分布**
8. **领导字节速率分布**
9. **主题副本分发**
10. **领导副本分发**
11. **首选领导选举机制**
12. **intra-broker 磁盘容量**
13. **intra-broker 磁盘用量分布**

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

"编写您自己的"目标，还不支持 Kafka 分配器目标。

19.2.2. Apache Kafka 自定义资源的 Streams 中的目标配置

您可以在 Kafka 和 KafkaRebalance 自定义资源中配置优化目标。Cruise Control 具有硬优化目标的配置，必须满足以及主要、默认和用户提供的优化目标。

您可以在以下配置中指定优化目标：

- 主目标 categories- Kafka.spec.cruiseControl.config.goals
- Hard goals — Kafka.spec.cruiseControl.config.hard.goals
- Default goals — Kafka.spec.cruiseControl.config.default.goals
- 用户提供的目标 iwl - KafkaRebalance.spec.goals

注意

资源分布目标取决于代理资源的 容量限制。

19.2.3. 硬和软优化目标

硬目标是在优化提议时 *必须满足* 的目标。在 Cruise 控制代码中没有定义为 *硬目标* 的目标被称为 *软目标*。您可以将软目标视为 *最佳工作* 目标： *它们不需要* 在优化建议中满足，但包含在优化计算中。违反了一个或多个软目标的优化建议，但满足所有硬目标是有效的。

Cruise Control 将计算满足所有硬目标以及尽可能多的软目标（按优先级顺序）的优化建议。无法满足所有硬目标的优化建议将由 Cruise 控制而拒绝，而 *不会* 发送给用户进行批准。

注意

例如，您可能有一个软目标来在集群间平均分配主题的副本（主题分布目标）。如果这样做可让所有配置的硬目标满足，则 Cruise Control 将忽略这个目标。

在 Cruise Control 中，以下 主要优化目标 是硬目标：

```
RackAwareGoal; ReplicaCapacityGoal; DiskCapacityGoal; NetworkInboundCapacityGoal;
NetworkOutboundCapacityGoal; CpuCapacityGoal
```

在 Cruise Control 部署配置中，您可以使用 Kafka.spec.cruiseControl.config 中的 hard.goals 属性指定强制哪个硬目标。

- 要强制执行所有硬目标，只需省略 hard.goals 属性。
- 要更改用于 Cruise Control 强制执行的硬目标，请使用其完全限定的域名在 hard.goals 属性中指定所需的目标。
- 要防止执行特定硬目标，请确保目标没有包含在 default.goals 和 hard.goals 列表中配置中。

注意

无法配置哪些目标被视为软或硬目标。这种区别由 Cruise Control 代码决定。

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:
      # Note that `default.goals` (superset) must also include all `hard.goals` (subset)
      default.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundCapacityGoal
      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 Control 生成有效优化方案的可能性。

如果在 KafkaRebalance 自定义资源中指定 skipHardGoalCheck: true，Cruise Control *不会检查*用户提供的优化目标（在 KafkaRebalance.spec.goals）列表是否包含 *所有配置的* 硬目标 (hard.goals)。因此，如果有些，但不是所有的，则用户提供的优化目标都位于 hard.goals 列表中，Cruise Control 仍会将其视为硬目标，即使指定了 skipHardGoalCheck: true。

19.2.4. 主要优化目标

所有用户都提供了*主要的优化目标*。没有在主优化目标中列出的目标在 Cruise Control 操作中不可用。

除非更改了 Cruise Control 部署配置，否则 Apache Kafka 的 Streams 将从 Cruise Control 中继承以下主要优化目标，以降序排列：

```
RackAwareGoal; MinTopicLeadersPerBrokerGoal; ReplicaCapacityGoal; DiskCapacityGoal;
NetworkInboundCapacityGoal; NetworkOutboundCapacityGoal; CpuCapacityGoal;
ReplicaDistributionGoal; PotentialNwOutGoal; DiskUsageDistributionGoal;
NetworkInboundUsageDistributionGoal; NetworkOutboundUsageDistributionGoal;
CpuUsageDistributionGoal; TopicReplicaDistributionGoal; LeaderReplicaDistributionGoal;
LeaderBytesInDistributionGoal; PreferredLeaderElectionGoal
```

其中一些目标被预先设置为硬目标。

要降低复杂性，我们建议您使用继承的主要优化目标，除非您需要 *完全* 排除 KafkaRebalance 资源中

使用的一个或多个目标。如果需要，可以在配置中为默认优化目标修改主要优化目标的优先级顺序（如果需要）。

如果需要，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 goals 属性。
- 如果您需要修改继承的主要优化目标，请在 goals 配置选项中指定目标列表（按优先级降序排列）。

注意

为了避免在生成优化建议时出现错误，请确保对 Kafka.spec.cruiseControl.config 中的 goals 或 default.goals 的更改包含为 hard.goals 属性指定的所有硬目标。为了说明这一点，还必须为主要优化目标和默认目标指定硬目标（作为子集）。

19.2.5. 默认优化目标

Cruise Control *使用默认优化目标*来生成*缓存的优化建议*。有关缓存的优化建议的详情，请参考第 19.3 节 “优化提议概述”。

您可以通过在 KafkaRebalance 自定义资源中设置 用户提供的优化目标 来覆盖默认的优化目标。

除非在 Cruise Control 部署配置中 指定 default.goals，否则主要优化目标将用作默认优化目标。在这种情况下，缓存的优化方案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:
      # Note that `default.goals` (superset) must also include all `hard.goals` (subset)
      default.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal
      hard.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal
      # ...
```

如果没有指定默认优化目标，则使用主优化目标生成缓存的提议。

19.2.6. 用户提供的优化目标

用户提供的优化目标 会缩小为特定优化提议配置的默认目标。您可以根据需要在 KafkaRebalance 自定义资源的 spec.goals 中设置它们：

```
KafkaRebalance.spec.goals
```

用户提供的优化目标可以为不同的场景生成优化建议。例如，您可能想要在不考虑磁盘容量或磁盘利用率的情况下在 Kafka 集群中优化领导副本分布。因此，您可以创建一个 KafkaRebalance 自定义资源，其中包含对领导副本分发的单个用户提供的目标。

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
- 是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 自定义资源中。请参阅 第 19.6 节 “生成优化建议”。

其他资源

- 使用 Kafka 配置和部署 Cruise 控制
- Cruise Control Wiki 中的配置。

19.3. 优化提议概述

配置 KafkaRebalance 资源以生成优化建议并应用推荐的更改。*optimization proposal* 是要生成一个更加均衡的 Kafka 集群、在代理中平均分配分区工作负载的建议概述。

每个优化建议都基于一组用于生成它的 优化目标，受代理资源 配置的任何容量限制。

所有优化的提议都是对提议重新平衡的影响的*估算*。您可以批准或拒绝提议。在不生成优化建议的情况下，您无法批准集群重新平衡。

您可以在以下重新平衡模式下运行优化建议：

- full
- add-brokers
- remove-brokers

19.3.1. 重新平衡模式

您可以使用 KafkaRebalance 自定义资源的 spec.mode 属性指定重新平衡模式。

full

full 模式通过在集群中的所有代理间移动副本来运行完全重新平衡。如果 KafkaRebalance 自定义资源中没有定义 spec.mode 属性，则这是默认模式。

add-brokers

在扩展 Kafka 集群后，通过添加一个或多个代理来使用 add-brokers 模式。通常，在扩展 Kafka 集群后，新的代理仅用于托管新创建的主题的分区。如果没有创建新主题，则不会使用新添加的代理，现有代理仍保留在同一负载中。通过在向集群添加代理后立即使用 add-brokers 模式，重新平衡操作会将副本从现有代理移到新添加的代理中。您可以使用 KafkaRebalance 自定义资源的 spec.brokers 属性将新代理指定为列表。

remove-brokers

在缩减 Kafka 集群前，通过删除一个或多个代理，使用 remove-brokers 模式。如果您缩减 Kafka 集群，代理也会关闭，即使它们托管副本。这可能会导致复制的分区，并可能导致某些分区位于其最小 ISR 下（同步副本）。为了避免这种潜在问题，remove-brokers 模式会将副本从要删除的代理中移出。当这些代理不再托管副本时，可以安全地运行缩减操作。您可以将您要删除的代理指定为 KafkaRebalance 自定义资源的 spec.brokers 属性中的列表。

通常，通过跨代理分布负载，使用 full rebalance 模式来重新平衡 Kafka 集群。只有在您要扩展集群或缩减并相应地重新平衡副本时，才使用 add-brokers 和 remove-brokers 模式。

运行重新平衡的过程实际上在三种不同的模式中相同。唯一的区别是通过 spec.mode 属性指定模式，如果需要，列出添加或将通过 spec.brokers 属性删除的代理。

19.3.2. 优化提议的结果

当生成优化建议时，会返回概述和代理负载。

概述

该摘要包含在 KafkaRebalance 资源中。概述建议的集群重新平衡，并指示涉及的更改的规模。成功生成的优化方案概述包含在 KafkaRebalance 资源的 Status.OptimizationResult 属性中。提供的信息是完整优化提议的摘要。

代理负载

代理负载存储在包含作为 JSON 字符串的数据的 ConfigMap 中。代理负载显示在提议重新平衡

的值前和之后，以便您可以看到集群中每个代理的影响。

19.3.3. 手动批准或拒绝优化建议

优化提议摘要显示了所提议的更改范围。

您可以使用 KafkaRebalance 资源的名称从命令行返回摘要。

返回优化提议概述

```
oc describe kafkarebalance <kafka_rebalance_resource_name> -n <namespace>
```

您还可以使用 jq 命令行 JSON 解析器工具。

使用 jq 返回优化提议概述

```
oc get kafkarebalance -o json | jq <jq_query>.
```

使用摘要决定是否批准或拒绝优化提议。

批准优化提议

您可以通过将 KafkaRebalance 资源的 strimzi.io/rebalance 注解设置为 批准 来删除优化建议。Cruise Control 将提议应用到 Kafka 集群，并启动集群重新平衡操作。

拒绝优化方案

如果您选择不批准优化方案，您可以更改优化目标或更新任何重新平衡性能调优选项，然后生成另一个提议。您可以通过将 strimzi.io/rebalance 注解设置为 refresh 来为 KafkaRebalance 资源生成一个新的优化建议。

使用优化建议评估重新平衡所需的移动。例如，概述描述了 inter-broker 和 intra-broker 移动。inter-broker 重新平衡在独立代理间移动数据。在使用 JBOD 存储配置时，intra-broker 重新平衡可在同一代理上的磁盘之间移动数据。即使您没有提前并批准提议，此类信息也很有用。

因为在重新平衡时在 Kafka 集群中出现额外的负载，您可能会拒绝优化过程或延迟其批准。

在以下示例中，提议建议在不同的代理间重新平衡数据。重新平衡涉及在代理间移动 55 分区副本，包括 12MB 数据。虽然分区副本间的移动对性能有高影响，但数据总数不大。如果总数据量较大，您可以拒绝提议，或者在批准重新平衡以限制 Kafka 集群性能的影响时的时间。

重新平衡性能调优选项有助于降低数据移动的影响。如果可扩展重新平衡周期，您可以将重新平衡分成较小的批处理。一次减少数据移动会减少集群的负载。

优化提议概述示例

```
Name:         my-rebalance
Namespace:    myproject
Labels:       strimzi.io/cluster=my-cluster
Annotations:  API Version:  kafka.strimzi.io/v1alpha1
Kind:         KafkaRebalance
Metadata:
# ...
Status:
  Conditions:
    Last Transition Time:  2022-04-05T14:36:11.900Z
    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:         12
    Monitored Partitions Percentage:      100
    Num Intra Broker Replica Movements:   0
    Num Leader Movements:                 24
    Num Replica Movements:                55
    On Demand Balancedness Score After:   82.91290759174306
    On Demand Balancedness Score Before:  78.01176356230222
    Recent Windows:                       5
  Session Id:                             a4f833bd-2055-4213-bfdd-ad21f95bf184
```

这个提议还会将 24 个分区领导移到不同的代理中。这需要对 ZooKeeper 配置进行更改，这对性能有低的影响。

balancedness 分数是优化提议前后 Kafka 集群的整体平衡的测量。平衡分数基于优化目标。如果满足所有目标，则分数为 100。当一个目标不满足时，分数会降低。比较均衡分数，以查看 Kafka 集群是否低于重新平衡。

19.3.4. 自动批准优化方案

要节省时间，您可以自动批准优化提议。使用自动化时，当您生成优化建议时，它会直接进入集群重新平衡中。

要启用优化提议 auto-approval 机制，请创建 KafkaRebalance 资源，并将 strimzi.io/rebalance-auto-approval 注解设置为 true。如果注解没有设置或设置为 false，则优化建议需要手动批准。

启用 auto-approval 机制的重新平衡请求示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
  name: my-rebalance
  labels:
    strimzi.io/cluster: my-cluster
  annotations:
    strimzi.io/rebalance-auto-approval: "true"
spec:
  mode: # any mode
  # ...
```

在自动批准优化提议时，您仍然可以检查状态。当重新平衡完成后，KafkaRebalance 资源的状态将变为 Ready。

19.3.5. 优化提议概述属性

下表解释优化提议概述部分中包含的属性。

表 19.1. 优化提议概述中包含的属性

JSON 属性	Description
numIntraBrokerReplicaMovements	在集群代理磁盘间传输的分区副本数。 **重新平衡操作期间的性能影响** ：高，但低于 **numReplicaMovements**。
excludedBrokersForLeadership	尚未支持。返回空列表。
numReplicaMovements	在独立代理之间移动的分区副本数量。 **在重新平衡操作期间影响性能** ：高。
onDemandBalancednessScoreBefore, onDemandBalancednessScoreAfter	Kafka 集群在生成优化建议之前和之后的整体 *均衡度* 的测量。 分数是通过从 100 中减去每个违反软目标的 **BalancednessScore** 的总和。Cruise Control 根据几个因素（包括优先级 - 目标在 **default.goal 或用户提供的目标列表中**）为**每个优化目标** 分配一个 **BalancednessScore**。 **Before** score 基于 Kafka 集群的当前配置。**After** 分数基于生成的优化提议。
intraBrokerDataToMoveMB	将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**）。 **在重新平衡操作期间影响性能** ：变量.集群重新平衡所需的时间越大，完成集群重新平衡所需的时间。在同一代理的磁盘间移动大量数据比不同的代理之间的影响较低（请参阅 **dataToMoveMB**）。
recentWindows	优化提议的指标窗口数量。
dataToMoveMB	将移动到独立代理的每个分区副本的大小总和（请参阅 **numReplicaMovements**）。 **在重新平衡操作期间影响性能** ：变量.集群重新平衡所需的时间越大，完成集群重新平衡所需的时间。
monitoredPartitionsPercentage	Kafka 集群中由优化提议涵盖的分区百分比。受 **excludedTopics** 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，则此处列出了与该表达式匹配的所有主题名称。这些主题不包括在优化建议中计算分区副本/领导移动。

JSON 属性	Description
numLeaderMovements	领导程序切换到不同副本的分区数量。这涉及更改 ZooKeeper 配置。 **在重新平衡操作期间影响性能** ：低.
excludedBrokersForReplicaMove	尚未支持。返回空列表。

19.3.6. 代理加载属性

代理负载存储在 ConfigMap 中（名称与 KafkaRebalance 自定义资源的名称相同），与 JSON 格式的字符串相同。此 JSON 字符串由一个 JSON 对象组成，其中包含链接到每个代理多个指标的代理 ID 的键。每个指标由三个值组成。第一个是应用优化提议前的指标值，第二个是应用提议后的指标的预期值，第三个是应用前两个值（在减前）之间的差别。

注意

当 KafkaRebalance 资源处于 ProposalReady 状态并在重新平衡完成后保留时，ConfigMap 会出现。

您可以使用 ConfigMap 的名称从命令行查看其数据。

返回 ConfigMap 数据

```
oc describe configmaps <my_rebalance_configmap_name> -n <namespace>
```

您还可以使用 jq 命令行 JSON 解析器工具 从 ConfigMap 中提取 JSON 字符串。

使用 jq 从 ConfigMap 中提取 JSON 字符串

```
oc get configmaps <my_rebalance_configmap_name> -o json | jq '.["data"]
["brokerLoad.json"]|fromjson|.'
```

下表解释优化提议的代理加载 ConfigMap 中包含的属性：

JSON 属性	Description
领导	此代理上分区领导的副本数。
replicas	此代理上的副本数量。
cpuPercentage	CPU 使用率作为定义容量的百分比。
diskUsedPercentage	磁盘利用率作为定义容量的百分比。
diskUsedMB	以 MB 为单位的绝对磁盘使用情况。
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	如果此代理成为当前主机的所有副本的领导，则假设的最大网络输出率。

19.3.7. 缓存的优化方案

Cruise Control 根据配置的默认优化目标维护*缓存的优化建议*。从工作负载模型生成，缓存的优化方案每 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优化建议，Cruise Control 会返回最新缓存的提议。

要更改缓存的优化提议刷新间隔，请编辑 Cruise Control 部署配置中的 proposal.expiration.ms 设置。考虑快速更改集群的间隔较短，尽管这会在 Cruise Control 服务器上增加负载。

其他资源

- 第 19.2 节 “优化目标概述”
- 第 19.6 节 “生成优化建议”

- 第 19.7 节 “批准优化提议”

19.4. 重新平衡性能调优概述

您可以调整集群重新平衡的几个性能调优选项。这些选项控制如何执行重新平衡中的分区副本和领导移动，以及分配给重新平衡操作的带宽。

19.4.1. 分区重新分配命令

优化建议 由单独的分区重新分配命令组成。当您 批准 提议时，Cruise Control 服务器会将这些命令应用到 Kafka 集群。

分区重新分配命令由以下一种操作组成：

- 分区移动：强制将分区副本及其数据传输到新位置。分区移动可以采用两种形式之一：
  - inter-broker movement ：分区副本被移到不同代理上的日志目录中。
  - intra-broker movement ：分区副本被移到同一代理的不同日志目录中。
- 领导移动：这涉及切换分区副本的领导。

Cruise Control 在批处理中向 Kafka 集群发出分区重新分配命令。重新平衡期间集群的性能会受到每个批处理中包含的每种移动数量的影响。

19.4.2. 副本移动策略

集群重新平衡性能也会受到 *副本移动策略的影响*，该策略应用于分区重新分配命令的批处理。默认情况下，Cruise Control 使用 BaseReplicaMovementStrategy，它只需按照生成顺序应用命令。但是，如果提议的早期有一些非常大的分区重新分配，则此策略可能会减慢其他重新分配的应用程序。

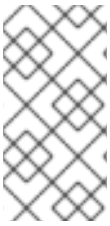
Cruise Control 提供了四个替代副本移动策略，可用于优化提议：

- PrioritizeSmallReplicaMovementStrategy: Order reassignments 以升序表示。
- PrioritizeLargeReplicaMovementStrategy: Order reassignments，以降序排列。
- PostponeUrpReplicaMovementStrategy: 优先考虑为没有处于不同步状态的分区的服务的重新分配。
- PrioritizeMinIsrWithOfflineReplicasStrategy: prioritiesitize reassignments with (At/Under) MinISR 分区，具有离线副本。只有在 Kafka 自定义资源的 spec 中将 cruiseControl.config.concurrency.adjuster.min.isr.check.enabled 设置为 true 时，此策略才会正常工作。

这些策略可以配置为序列。第一个策略尝试使用其内部逻辑比较两个分区重新分配。如果重新分配等同于，那么它将按顺序传递给下一个策略，以确定顺序等。

19.4.3. intra-broker 磁盘平衡

在同一代理的磁盘间移动大量数据比独立代理间的影响较低。如果您正在运行使用带有同一代理中的多个磁盘的 JBOD 存储的 Kafka 部署，Cruise Control 可以平衡磁盘之间的分区。



注意

如果您将 JBOD 存储与单一磁盘配合使用，在代理磁盘平衡中，则会导致一个有 0 分区移动的成成，因为没有磁盘之间的平衡。

要执行 intra-broker 磁盘平衡，请在 KafkaRebalance.spec 下将 rebalanceDisk 设置为 true。当将 rebalanceDisk 设置为 true 时，不要在 KafkaRebalance.spec 中设置 goals 字段，因为 Cruise Control 会自动设置 intra-broker 目标并忽略 inter-broker 目标。Cruise Control 不会同时执行 inter-broker 和 intra-broker 平衡。

19.4.4. 重新平衡调整选项

Cruise Control 提供多个配置选项用于调优上述重新平衡参数。在使用 Kafka 配置和部署 Cruise Control 时，您可以设置这些调整选项或优化提议级别：

- Cruise Control server 设置可以在 Kafka. spec.cruiseControl.config 下的 Kafka 自定义

资源中设置。

- 单个重新平衡性能配置可以在 KafkaRebalance.spec 下设置。

下表总结了相关的配置。

表 19.2. 重新平衡性能调优配置

Cruise Control 属性	KafkaRebalance 属性	默认	描述
num.concurrent.partition.movements.per.broker	concurrentPartitionMovementsPerBroker	5	每个分区重新分配批处理中的最大 inter-broker 分区移动数
num.concurrent.intra.broker.partition.movements	concurrentIntraBrokerPartitionMovements	2	每个分区重新分配批处理中的最大 intra-broker 分区移动数
num.concurrent.leader.movements	concurrentLeaderMovements	1000	每个分区重新分配批处理中的最大分区领导更改数
default.replication.throttle	replicationThrottle	null（无限制）	分配给分区重新分配的带宽（每秒字节数）

Cruise Control 属性	KafkaRebalance 属性	默认	描述
default.replica.movement.strategies	**replicaMovementStrategies**	**BaseReplicaMovementStrategy**	用于决定为生成的提议执行分区重新分配命令的顺序（优先级顺序）的列表。对于 server 设置，使用以逗号分隔的字符串，以及策略类的完全限定名称（将 **com.linkedin.kafka.cruisecontrol.executor.strategy.** 添加到每个类名称的开头）。对于 **KafkaRebalance** 资源设置，请使用策略类名称的 YAML 数组。
-	**rebalanceDisk**	false	启用 intra-broker 磁盘平衡，平衡同一代理上磁盘之间的磁盘空间利用率。只适用于使用多个磁盘的 JBOD 存储的 Kafka 部署。

更改默认设置会影响重新平衡完成所需的时间，以及重新平衡期间在 Kafka 集群上放置的负载。使用较低值可减少负载，但会增加所花费的时间，反之亦然。

其他资源

- CruiseControlSpec 模式参考
- KafkaRebalanceSpec 模式参考

19.5. 使用 KAFKA 配置和部署 CRUISE 控制

配置 Kafka 资源，以使用 Kafka 集群部署 Cruise Control。您可以使用 Kafka 资源的 cruiseControl

属性来配置部署。为每个 Kafka 集群部署一个 Cruise Control 实例。

在 Cruise Control config 中使用 goals 配置来指定优化目标，以生成优化建议。您可以使用 brokerCapacity 更改与资源分布相关的目标的默认容量限制。如果代理在带有异构网络资源的节点上运行，您可以使用 覆盖 来为每个代理设置网络容量限制。

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

有关 Cruise Control 配置选项的更多信息，请参阅 Apache Kafka 自定义资源 API 参考流。

先决条件

- 一个 OpenShift 集群
- 正在运行的 Cluster Operator

流程

1. 编辑 Kafka 资源的 cruiseControl 属性。

   以下示例配置中显示了您可以配置的属性：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    brokerCapacity: 1
      inboundNetwork: 10000KB/s
      outboundNetwork: 10000KB/s
      overrides: 2
      - brokers: [0]
        inboundNetwork: 20000KiB/s
        outboundNetwork: 20000KiB/s
      - brokers: [1, 2]
        inboundNetwork: 30000KiB/s
        outboundNetwork: 30000KiB/s
      # ...
    config: 3
```

```
    # Note that `default.goals` (superset) must also include all `hard.goals` (subset)
    default.goals: > 4
      com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal
      # ...
    hard.goals: >
      com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal
      # ...
    cpu.balance.threshold: 1.1
    metadata.max.age.ms: 300000
    send.buffer.bytes: 131072
    webserver.http.cors.enabled: true 5
    webserver.http.cors.origin: "*"
    webserver.http.cors.exposeheaders: "User-Task-ID,Content-Type"
    # ...
  resources: 6
    requests:
      cpu: 1
      memory: 512Mi
    limits:
      cpu: 2
      memory: 2Gi
  logging: 7
      type: inline
      loggers:
        rootLogger.level: INFO
  template: 8
    pod:
      metadata:
        labels:
          label1: value1
      securityContext:
        runAsUser: 1000001
        fsGroup: 0
      terminationGracePeriodSeconds: 120
  readinessProbe: 9
    initialDelaySeconds: 15
    timeoutSeconds: 5
  livenessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 5
  metricsConfig: 10
    type: jmxPrometheusExporter
    valueFrom:
      configMapKeyRef:
        name: cruise-control-metrics
        key: metrics-config.yml
# ...
```

1

代理资源的容量限制。

2

在带有异构网络资源的节点上运行时，覆盖为特定代理设置网络容量限制。

3

Cruise Control 配置。标准 Cruise Control 配置可能会提供给不是由 Apache Kafka 的 Streams 直接管理的属性。

4

优化目标配置，其中包括用于默认优化目标(default.goals)、主要优化目标（目标）和硬目标(hard.goals)的配置。

5

为 Cruise Control API 的只读访问权限启用并配置了 CORS。

6

为保留支持的资源（当前 cpu 和 memory ）的请求，以及指定可消耗的最大资源的限制。

7

Cruise Control loggers 和日志级别直接(内联)或通过 ConfigMap 间接添加(外部)。自定义 Log4j 配置必须放在 ConfigMap 中的 log4j.properties 键下。Cruise Control 具有一个名为 rootLogger.level 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。

8

模板自定义。这里有一个额外的安全属性调度 pod。

9

检查检查以了解何时重启容器（存活度）以及何时容器可以接受流量（就绪度）。

10

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

2. 创建或更新资源：

```
oc apply -f <kafka_configuration_file>
```

3. 检查部署的状态：

```
oc get deployments -n <my_cluster_operator_namespace>
```

输出显示部署名称和就绪

```
NAME                        READY  UP-TO-DATE  AVAILABLE
my-cluster-cruise-control 1/1    1           1
```

my-cluster 是 Kafka 集群的名称。

READY 显示就绪/预期的副本数。当 AVAILABLE 输出显示为 1 时，部署成功。

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。Cruise Control 需要这些主题才能正常工作，且不得删除或更改。您可以使用指定的配置选项更改主题的名称。

表 19.3. 自动创建的主题

自动创建的主题配置	默认主题名称	创建人	功能
metric.reporter.topic	**strimzi.cruisecontrol.metrics**	Apache Kafka Metrics Reporter 的流	将 Metrics Reporter 中的原始指标存储在每个 Kafka 代理中。
partition.metric.sample.store.topic	**strimzi.cruisecontrol.partitionmetricsamples**	Sything Control	存储每个分区的派生指标。它们由指标示例聚合器创建。
broker.metric.sample.store.topic	**strimzi.cruisecontrol.modeltrainingsamples**	Sything Control	存储用于创建 Cluster Workload Model 的指标样本。

要防止删除 Cruise Control 所需的记录，日志压缩会在自动创建的主题中禁用。

注意

如果在启用了 Cruise Control 的 Kafka 集群中更改自动创建主题的名称，旧的主题不会被删除，应手动删除旧的主题。

接下来要做什么

配置和部署 Cruise Control 后，您可以生成优化提议。

其他资源

- 优化目标概述

19.6. 生成优化建议

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。分析优化方案中的信息，并决定是否批准它。您可以使用优化提议的结果来重新平衡 Kafka 集群。

您可以在以下模式之一中运行优化建议：

- full （默认）
- add-brokers
- remove-brokers

您使用的模式取决于您是否在 Kafka 集群中已在运行的所有代理进行重新平衡；或者要在扩展后或缩减 Kafka 集群前重新平衡。如需更多信息，请参阅使用代理扩展重新平衡模式。

先决条件

- 您已将 Cruise Control 部署到 Apache Kafka 集群的 Streams 中。
- 您已配置了优化目标，以及（可选）代理资源的容量限制。

有关配置 Cruise Control 的详情，请参考 第 19.5 节 “使用 Kafka 配置和部署 Cruise 控制”。

流程

1. 创建 KafkaRebalance 资源并指定适当的模式。

完整 模式（默认）

要使用 Kafka 资源中定义的 *默认优化目标*，请将 spec 属性留空。默认情况下，Cruise Control 以 full 模式重新平衡 Kafka 集群。

默认具有完整重新平衡的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
  name: my-rebalance
  labels:
    strimzi.io/cluster: my-cluster
spec: {}
```

您还可以通过 spec.mode 属性指定 full 模式来运行完整的重新平衡。

指定 full 模式的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
  name: my-rebalance
  labels:
```

```
    strimzi.io/cluster: my-cluster
spec:
  mode: full
```

add-brokers 模式

如果要在扩展后重新平衡 Kafka 集群，请指定 add-brokers 模式。

在这个模式中，现有副本被移到新添加的代理中。您需要将代理指定为列表。

指定 add-brokers 模式的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
  name: my-rebalance
  labels:
    strimzi.io/cluster: my-cluster
spec:
  mode: add-brokers
  brokers: [3, 4] 1
```

1

扩展操作添加的新添加的代理列表。此属性是必需的。

remove-brokers 模式

如果要在缩减前重新平衡 Kafka 集群，请指定 remove-brokers 模式。

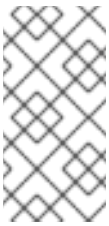
在这个模式中，副本将从要删除的代理中移出。您需要指定作为列表删除的代理。

指定 remove-brokers 模式的配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
```

```
kind: KafkaRebalance
metadata:
  name: my-rebalance
  labels:
    strimzi.io/cluster: my-cluster
spec:
  mode: remove-brokers
  brokers: [3, 4] (1)
```

(1) 缩减操作要删除的代理列表。此属性是必需的。



注意

无论您使用的重新平衡模式如何，以下步骤和批准或停止重新平衡的步骤都相同。

2. 要配置 *用户提供的优化目标*，而不是使用默认目标，请添加 goals 属性并输入一个或多个目标。

在以下示例中，机架感知和副本容量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
```

3. 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
```

4. （可选）要自动批准优化提议，请将 strimzi.io/rebalance-auto-approval 注解设置为 true：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaRebalance
metadata:
  name: my-rebalance
  labels:
    strimzi.io/cluster: my-cluster
  annotations:
    strimzi.io/rebalance-auto-approval: "true"
spec:
  goals:
    - RackAwareGoal
    - ReplicaCapacityGoal
  skipHardGoalCheck: true
```

5. 创建或更新资源：

```
oc apply -f <kafka_rebalance_configuration_file>
```

Cluster Operator 从 Cruise Control 请求优化提议。这可能需要几分钟时间，具体取决于 Kafka 集群的大小。

6. 如果您使用自动批准机制，请等待优化提议的状态更改为 Ready。如果您没有启用自动批准机制，请等待优化提议的状态更改为 ProposalReady ：

```
oc get kafkarebalance -o wide -w -n <namespace>
```

PendingProposal

PendingProposal 状态意味着重新平衡 Operator 正在轮询 Cruise Control API 来检查优化提议是否就绪。

ProposalReady

ProposalReady 状态表示优化提议已准备好审核和批准。

当状态变为 ProposalReady 时，优化提议可以批准。

7. 查看优化方案。

优化建议包含在 KafkaRebalance 资源的 Status.Optimization Result 属性中。

```
oc describe kafkarebalance <kafka_rebalance_resource_name>
```

优化提议示例

```
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 部分中的属性描述了待处理的集群重新平衡操作。有关每个属性的描述，请参阅优化提议 的内容。

CPU 容量不足

如果 Kafka 集群在 CPU 使用率上过载，您可能会在 KafkaRebalance 状态中看到 CPU 容量错误。值得注意的是，这个利用率值不受 excludedTopics 配置的影响。虽然优化建议不会重新分配排除主题的副本，但它们的负载仍然在利用率计算中考虑。

CPU 使用率错误示例

```
com.linkedin.kafka.cruisecontrol.exception.OptimizationFailureException:
[CpuCapacityGoal] Insufficient capacity for cpu (Utilization 615.21, Allowed Capacity 420.00,
Threshold: 0.70). Add at least 3 brokers with the same cpu capacity (100.00) as broker-0. Add
at least 3 brokers with the same cpu capacity (100.00) as broker-0.
```

注意

此错误以百分比而不是 CPU 内核数显示 CPU 容量。因此，它不会直接映射到 Kafka 自定义资源中配置的 CPU 数量。它希望每个代理有一个 *虚拟* CPU，它在 Kafka.spec.kafka.resources.limits.cpu 中配置 CPU 的周期。这对重新平衡行为没有影响，因为 CPU 使用率和容量的比率保持不变。

接下来要做什么

第 19.7 节 “批准优化提议”

其他资源

- 第 19.3 节 “优化提议概述”

19.7. 批准优化提议

如果其状态为 ProposalReady，您可以批准 Cruise Control 生成的 优化方案。然后，Cruise Control 会将优化提议应用到 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。在批准优化提议前，您必须：

- 刷新提议已过期。
- 仔细检查 提议的内容。

先决条件

- 您已从 Cruise 控制中生成了优化的提议。
- KafkaRebalance 自定义资源状态为 ProposalReady。

流程

对您要批准的优化提议执行这些步骤。

1. 除非新生成了优化建议，请检查它是否基于 Kafka 集群状态的当前信息。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 strimzi.io/rebalance=refresh 注解 OpenShift 中的 KafkaRebalance 资源：

   ```
   oc annotate kafkarebalance <kafka_rebalance_resource_name>
   strimzi.io/rebalance="refresh"
   ```

2. 等待优化提议的状态更改为 ProposalReady ：

   ```
   oc get kafkarebalance -o wide -w -n <namespace>
   ```

   PendingProposal

   PendingProposal 状态意味着重新平衡 Operator 正在轮询 Cruise Control API 来检查优化提议是否就绪。

   ProposalReady

ProposalReady 状态表示优化提议已准备好审核和批准。

当状态变为 ProposalReady 时，优化提议可以批准。

3. 批准您希望 Cruise Control 适用的优化方案。

使用 strimzi.io/rebalance=approve 注解 OpenShift 中的 KafkaRebalance 资源：

```
oc annotate kafkarebalance <kafka_rebalance_resource_name>
strimzi.io/rebalance="approve"
```

4. Cluster Operator 会检测注解的资源，并指示 Cruise Control 重新平衡 Kafka 集群。

5. 等待优化提议的状态更改为 Ready ：

```
oc get kafkarebalance -o wide -w -n <namespace>
```

重新平衡

Rebalancing 状态代表重新平衡正在进行。

Ready

Ready 状态表示重新平衡完成。

NotReady

NotReady 状态意味着出现错误 - 请参阅 KafkaRebalance 资源 中的修复问题。

当状态更改为 Ready 时，重新平衡已完成。

要使用同一 KafkaRebalance 自定义资源来生成另一个优化提议，请将 refresh 注解应用到自定义资源。这会将自定义资源移到 PendingProposal 或 ProposalReady 状态。然后，您可以查看优化提议并批准它（如果需要）。

其他资源

- 第 19.3 节 “优化提议概述”
- 第 19.8 节 “停止集群重新平衡”

19.8. 停止集群重新平衡

启动后，集群重新平衡操作可能需要一些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 完成当前分区重新分配的批处理，然后停止重新平衡。当重新平衡停止后，已完成的分区重新分配已被应用；因此，与重新平衡操作开始相比，Kafka 集群的状态会有所不同。如果需要进一步重新平衡，您应该生成新的优化方案。

注意

在 intermediate（停止）状态中的 Kafka 集群的性能可能比初始状态更糟。

先决条件

- 您已通过使用 approve 注解了 KafkaRebalance 自定义资源来批准优化的提议。
- KafkaRebalance 自定义资源的状态是 Rebalancing。

流程

1. 在 OpenShift 中注解 KafkaRebalance 资源：

```
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. 等待状态变为 Stopped。

其他资源

- 第 19.3 节 “优化提议概述”

19.9. 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。资源也会进入 NotReady 状态。

要继续集群重新平衡操作，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：

- KafkaRebalance 资源中的错误参数。
- 在 KafkaRebalance 资源中指定 Kafka 集群的 strimzi.io/cluster 标签缺失。
- Cruise Control 服务器没有部署，因为缺少 Kafka 资源中的 cruiseControl 属性。
- 无法访问 Cruise Control 服务器。

修复这个问题后，您需要将 refresh 注解添加到 KafkaRebalance 资源。在 "refresh" 期间，通过 Cruise Control 服务器请求一个新的优化提议。

先决条件

- 您已 批准了优化方案。
- 重新平衡操作的 KafkaRebalance 自定义资源的状态是 NotReady。

流程

1. 从 KafkaRebalance 状态获取有关错误的信息：

```
oc describe kafkarebalance rebalance-cr-name
```

2. 尝试解决 KafkaRebalance 资源中的问题。

3. 在 OpenShift 中注解 KafkaRebalance 资源：

```
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. 等待状态变为 PendingProposal，或者直接更改为 ProposalReady。

其他资源

- 第 19.3 节 “优化提议概述”

第 20 章 使用分区重新分配工具

在扩展 Kafka 集群时，您可能需要添加或删除代理并更新分区的发布或主题的复制因素。要更新分区和主题，您可以使用 kafka-reassign-partitions.sh 工具。

Apache Kafka Cruise Control 集成和主题 Operator 的流不支持更改主题的复制因素。但是，您可以使用 kafka-reassign-partitions.sh 工具更改主题的复制因素。

该工具还可用于重新分配分区并在代理间平衡分区分布以提高性能。但是，建议您使用 Cruise Control 进行自动分区重新分配和集群重新平衡。Cruise Control 可以在不停机的情况下将主题从一个代理移到另一个代理，这是重新分配分区的最高效方法。

建议以单独的交互式 pod 而不是代理容器中运行 kafka-reassign-partitions.sh 工具。在代理容器中运行 Kafka bin/ 脚本可能会导致 JVM 从与 Kafka 代理相同的设置开始，这可能会导致中断。通过在单独的 pod 中运行 kafka-reassign-partitions.sh 工具，您可以避免出现这个问题。使用 -ti 选项运行 pod 会创建一个带有终端的交互式 pod，用于在 pod 中运行 shell 命令。

使用终端运行交互式 pod

```
oc run helper-pod -ti --image=registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0 --rm=true --restart=Never -- bash
```

20.1. 分区重新分配工具概述

分区重新分配工具提供以下功能来管理 Kafka 分区和代理：

重新分发分区副本

通过添加或删除代理来扩展集群，并将 Kafka 分区从大量载入的代理移到使用不足的代理中。要做到这一点，您必须创建一个分区重新分配计划，标识要移动哪些主题和分区，以及移动它们的位置。建议对这类操作进行 Cruise Control，因为它 可自动执行集群重新平衡过程。

扩展和缩减主题复制因素

增加或减少 Kafka 主题的复制因素。要做到这一点，您必须创建一个分区重新分配计划，标识分区之间的现有复制分配，并使用复制因素更改更新分配。

更改首选领导

更改 Kafka 分区的首选领导。如果当前首选领导不可用，或者要在集群中的代理间重新分发负载，这很有用。要做到这一点，您必须创建一个分区重新分配计划，通过更改副本顺序为每个分区指定新的首选领导。

更改日志目录以使用特定的 JBOD 卷

更改 Kafka 代理的日志目录以使用特定的 JBOD 卷。如果要将 Kafka 数据移到不同的磁盘或存储设备中，这非常有用。要做到这一点，您必须创建一个分区重新分配计划，该计划为每个主题指定新日志目录。

20.1.1. 生成分区重新分配计划

分区重新分配工具(kafka-reassign-partitions.sh)通过生成分区分配计划来工作，指定应将哪些分区从当前代理移到新代理中。

如果您对计划满意，您可以执行它。然后，该工具执行以下操作：

- 将分区数据迁移到新代理
- 更新 Kafka 代理上的元数据以反映新分区分配
- 触发 Kafka 代理的滚动重启，以确保新分配生效

分区重新分配工具有三个不同的模式：

--generate

取一组主题和代理，并生成 *重新分配 JSON 文件*，这会导致将这些主题的分区分配给这些代理。由于这对整个主题进行操作，因此当您只需要重新分配某些主题的一些分区时，无法使用它。

--execute

取一个 *重新分配 JSON 文件*，并将其应用到集群中的分区和代理。因此，获得分区的代理遵循分区领导机。对于给定分区，当新代理发现并加入 ISR （同步副本）后，旧代理将停止为后续，并将删除其副本。

--verify

使用与 --execute 步骤相同的 *重新分配 JSON 文件*，-- verify 会检查文件中所有分区是否已移至其预期的代理中。如果重新分配已完成，-- verify 也会删除任何生效的流量节流(--throttle)。除非被删除，throttles 将继续影响集群，即使重新分配完成后也是如此。

在任何给定时间，集群中只能有一个重新分配运行，且无法取消正在运行的重新分配。如果需要取消重新分配，请等待它完成，然后执行另一个重新分配来恢复第一个重新分配的影响。kafka-reassign-partitions.sh 将打印这个 reversion 的重新分配 JSON 作为其输出的一部分。非常大的重新分配应该被分解为多个较小的重新分配，以防需要停止 in-progress 重新分配。

20.1.2. 在分区重新分配 JSON 文件中指定主题

kafka-reassign-partitions.sh 工具使用一个重新分配 JSON 文件，该文件指定要重新分配的主题。如果要移动特定分区，您可以生成重新分配 JSON 文件或手动创建文件。

一个基本的重新分配 JSON 文件在以下示例中显示结构，它描述了属于两个 Kafka 主题的三个分区。每个分区都会被重新分配给一组新的副本，这些副本由代理 ID 识别。版本、topic、分区、和 replicas 属性都是必需的。

分区重新分配 JSON 文件结构示例

```
{
  "version": 1, 1
  "partitions": [ 2
    {
      "topic": "example-topic-1", 3
      "partition": 0, 4
      "replicas": [1, 2, 3] 5
    },
    {
      "topic": "example-topic-1",
      "partition": 1,
      "replicas": [2, 3, 4]
    },
    {
      "topic": "example-topic-2",
      "partition": 0,
      "replicas": [3, 4, 5]
    }
  ]
}
```

1

重新分配 JSON 文件格式的版本。目前只支持版本 1，因此应始终为 1。

2

指定重新分配的分区的数组。

3

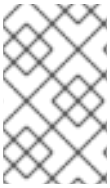
分区所属的 Kafka 主题的名称。

4

被重新分配的分区的 ID。

5

应该为此分区分配为副本的代理 ID 的排序数组。列表中的第一个代理是领导副本。



注意

不包含在 JSON 中的分区不会改变。

如果您只使用主题数组指定 主题，分区重新分配工具会重新分配属于指定主题的所有分区。

为主题重新分配所有分区的重新分配 JSON 文件结构示例

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

20.1.3. 在 JBOD 卷间重新分配分区

当在 Kafka 集群中使用 JBOD 存储时，您可以在特定卷及其日志目录间重新分配分区（每个卷只有一个日志目录）。

要将分区重新分配给特定卷，请在重新分配 JSON 文件中为每个分区添加 log_dirs 值。每个 log_dirs 数组包含与 replicas 数组相同的条目数，因为每个副本都应分配给特定的日志目录。log_dirs 数组包含一个到日志目录的绝对路径或特殊值 any。any 值表示 Kafka 可以为该副本选择任何可用的日志目录，这在 JBOD 卷间重新分配分区时很有用。

使用日志目录重新分配 JSON 文件结构示例

```
{
  "version": 1,
  "partitions": [
    {
      "topic": "example-topic-1",
      "partition": 0,
      "replicas": [1, 2, 3]
      "log_dirs": ["/var/lib/kafka/data-0/kafka-log1", "any", "/var/lib/kafka/data-1/kafka-log2"]
    },
    {
      "topic": "example-topic-1",
      "partition": 1,
      "replicas": [2, 3, 4]
      "log_dirs": ["any",  "/var/lib/kafka/data-2/kafka-log3", "/var/lib/kafka/data-3/kafka-log4"]
    },
    {
      "topic": "example-topic-2",
      "partition": 0,
      "replicas": [3, 4, 5]
      "log_dirs": ["/var/lib/kafka/data-4/kafka-log5", "any",  "/var/lib/kafka/data-5/kafka-log6"]
    }
  ]
}
```

20.1.4. 分区重新分配节流

分区重新分配可能会是一个较慢的过程，因为它涉及在代理间传输大量数据。为了避免对客户端产生不利影响，您可以限制重新分配过程。使用带有 kafka-reassign-partitions.sh 工具的 --throttle 参数，以节流重新分配。您可以为代理间移动分区指定每秒的最大阈值（以字节为单位）。例如，-- throttle 5000000 为移动分区设置 50 MBps 的最大阈值。

节流可能会导致重新分配完成所需的时间。

- 如果节流过低，则新分配的代理将无法与要发布的记录保持同步，且重新分配永远不会完成。

- 如果节流过高，客户端将会受到影响。

例如，对于生成者，这可能比等待确认的正常延迟高。对于消费者，这可能因为轮询之间延迟更高的吞吐量导致的吞吐量下降。

20.2. 生成重新分配 JSON 文件来重新分配分区

使用 kafka-reassign-partitions.sh 工具生成一个重新分配 JSON 文件，以便在扩展 Kafka 集群后重新分配分区。添加或删除代理不会自动重新分发现有分区。要平衡分区分布并充分利用新代理，您可以使用 kafka-reassign-partitions.sh 工具重新分配分区。

您可以从连接到 Kafka 集群的交互式 pod 容器运行该工具。

以下流程描述了使用 mTLS 的安全重新分配过程。您需要一个使用 TLS 加密和 mTLS 身份验证的 Kafka 集群。

您需要以下内容来建立连接：

- 创建 Kafka 集群时由 Cluster Operator 生成的集群 CA 证书和私钥

- 当用户为对 Kafka 集群进行客户端访问时，User Operator 生成的用户 CA 证书和私钥

在此过程中，CA 证书和对应的密码会从集群和包含它们的用户 secret 中提取，这些 secret 以 PKCS #12 (.p12 和 .password) 的格式提取。密码允许访问包含证书的 .p12 存储。您可以使用 .p12 存储指定一个信任存储和密钥存储来验证与 Kafka 集群的连接。

先决条件

- 您有一个正在运行的 Cluster Operator。

-

您有一个正在运行的 Kafka 集群，它基于配置了内部 TLS 加密和 mTLS 身份验证的 Kafka 资源。

使用 TLS 加密和 mTLS 验证的 Kafka 配置

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    listeners:
      # ...
      - name: tls
        port: 9093
        type: internal
        tls: true 1
        authentication:
          type: tls 2
    # ...
```

1 为内部监听程序启用 TLS 加密。

2 侦听器身份验证机制指定为 mutual tls。

- 正在运行的 Kafka 集群包含一组要重新分配的主题和分区。

my-topic的主题配置示例

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

```
spec:
  partitions: 10
  replicas: 3
  config:
    retention.ms: 7200000
    segment.bytes: 1073741824
    # ...
```

- 有一个配置了 ACL 规则的 KafkaUser，它用于指定从 Kafka 代理生成和使用主题的权限。

使用 ACL 规则的 Kafka 用户配置示例，允许对 my-topic 和 my-cluster执行操作

```
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:
      # access to the topic
      - resource:
          type: topic
          name: my-topic
        operations:
          - Create
          - Describe
          - Read
          - AlterConfigs
        host: "*"
      # access to the cluster
      - resource:
          type: cluster
        operations:
          - Alter
          - AlterConfigs
        host: "*"
      # ...
  # ...
```

1. 定义为 mutual tls 的用户身份验证机制。

2. ACL 规则的简单授权和附带列表。

流程

1. 从 Kafka 集群的 < *cluster_name*> -cluster-ca-cert secret 中提取集群 CA 证书和密钥。

```
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
```

将 *<cluster_name >* 替换为 Kafka 集群的名称。当使用 Kafka 资源部署 Kafka 时，使用 Kafka 集群名称(<cluster_*name*> -cluster-ca-cert)创建带有集群CA 证书的 secret。例如，my-cluster-cluster-ca-cert。

2. 使用 Apache Kafka 镜像的 Streams 运行一个新的交互式 pod 容器，以连接到正在运行的 Kafka 代理。

```
oc run --restart=Never --image=registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0 <interactive_pod_name> -- /bin/sh -c "sleep 3600"
```

将 *<interactive_pod_name >* 替换为 pod 的名称。

3. 将集群 CA 证书复制到交互式 pod 容器。

```
oc cp ca.p12 <interactive_pod_name>:/tmp
```

4. 从有权访问 Kafka 代理的 Kafka 用户的 secret 中提取用户 CA 证书和密钥。

```
oc get secret <kafka_user> -o jsonpath='{.data.user\.p12}' | base64 -d > user.p12
```

```
oc get secret <kafka_user> -o jsonpath='{.data.user\.password}' | base64 -d > user.password
```

将 *<kafka_user>* 替换为 Kafka 用户的名称。当使用 KafkaUser 资源创建 Kafka 用户时，会使用 Kafka 用户名创建带有用户 CA 证书的 secret。例如，my-user。

5. 将用户 CA 证书复制到交互式 pod 容器。

```
oc cp user.p12 <interactive_pod_name>:/tmp
```

CA 证书允许交互式 pod 容器使用 TLS 连接到 Kafka 代理。

6. 创建 config.properties 文件，以指定用于验证与 Kafka 集群连接的信任存储和密钥存储。

使用您在前面的步骤中提取的证书和密码。

```
bootstrap.servers=<kafka_cluster_name>-kafka-bootstrap:9093 1
security.protocol=SSL 2
ssl.truststore.location=/tmp/ca.p12 3
ssl.truststore.password=<truststore_password> 4
ssl.keystore.location=/tmp/user.p12 5
ssl.keystore.password=<keystore_password> 6
```

1. 连接到 Kafka 集群的 bootstrap 服务器地址。使用您自己的 Kafka 集群名称替换 *<kafka_cluster_name>*。

2. 使用 TLS 加密时的安全协议选项。

3. truststore 位置包含了 Kafka 集群的公钥证书 (ca.p12)。

4. 用于访问信任存储的密码(ca.password)。

5.

6

用于访问密钥存储的密码(user.password)。

7. 将 config.properties 文件复制到交互式 pod 容器中。

```
oc cp config.properties <interactive_pod_name>:/tmp/config.properties
```

8. 准备名为 topics.json 的 JSON 文件，以指定要移动的主题。

将主题名称指定为用逗号分开的列表。

重新分配 my-topic的所有分区的 JSON 文件示例

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

您还可以使用此文件 更改主题 的复制因素。

9. 将 *topics.json* 文件复制到交互式 pod 容器中。

```
oc cp topics.json <interactive_pod_name>:/tmp/topics.json
```

10. 在交互式 pod 容器中启动 shell 进程。

```
oc exec -n <namespace> -ti <interactive_pod_name> /bin/bash
```

将 *<namespace>* 替换为运行 pod 的 OpenShift 命名空间。

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

将 my-topic 分区移到指定代理的示例

```
bin/kafka-reassign-partitions.sh --bootstrap-server my-cluster-kafka-bootstrap:9093 \
  --command-config /tmp/config.properties \
  --topics-to-move-json-file /tmp/topics.json \
  --broker-list 0,1,2,3,4 \
  --generate
```

其他资源

- 配置 Kafka
- 第 10.4 节 “配置 Kafka 主题”
- 第 11.1 节 “配置 Kafka 用户”

20.3. 添加代理后重新分配分区

在增加 Kafka 集群中的代理数量后，使用 kafka-reassign-partitions.sh 工具生成的重新分配文件来重新分配分区。重新分配文件应该描述了如何将分区重新分配给放大 Kafka 集群中的代理。您可以将文件中指定的重新分配应用到代理，然后验证新分区分配。

此流程描述了使用 TLS 的安全扩展过程。您需要一个使用 TLS 加密和 mTLS 身份验证的 Kafka 集群。

kafka-reassign-partitions.sh 工具可用于重新分配 Kafka 集群中的分区，无论您是否通过集群管理所有节点，还是使用节点池来管理集群中的节点组。

注意

虽然您可以使用 kafka-reassign-partitions.sh 工具，但建议使用 Cruise Control 进行自动分区重新分配和集群重新平衡。Cruise Control 可以在不停机的情况下将主题从一个代理移到另一个代理，这是重新分配分区的最高效方法。

先决条件

- 您有一个正在运行的 Kafka 集群，它基于配置了内部 TLS 加密和 mTLS 身份验证的 Kafka 资源。
- 您已生成了一个重新分配 JSON 文件，名为 reassignment.json。
- 您正在运行连接到正在运行的 Kafka 代理的交互式 pod 容器。
- 您作为配置了 ACL 规则的 KafkaUser 连接，该规则指定管理 Kafka 集群及其主题的权限。

流程

1. 通过增加 Kafka.spec.kafka.replicas 配置选项，添加您需要的许多新代理。
2. 验证新代理 pod 是否已启动。
3. 如果没有这样做，请运行交互式 pod 容器来生成一个重新分配 JSON 文件，名为 reassignment.json。
4. 将 reassignment.json 文件复制到交互式 pod 容器中。

```
oc cp reassignment.json <interactive_pod_name>:/tmp/reassignment.json
```

将 *<interactive_pod_name>* 替换为 pod 的名称。

5. 在交互式 pod 容器中启动 shell 进程。

```
oc exec -n <namespace> -ti <interactive_pod_name> /bin/bash
```

将 *<namespace>* 替换为运行 pod 的 OpenShift 命名空间。

6. 使用交互式 pod 容器中的 kafka-reassign-partitions.sh 脚本运行分区重新分配。

```
bin/kafka-reassign-partitions.sh --bootstrap-server
 <cluster_name>-kafka-bootstrap:9093 \
 --command-config /tmp/config.properties \
 --reassignment-json-file /tmp/reassignment.json \
 --execute
```

将 *<cluster_name>* 替换为 Kafka 集群的名称。例如， my-cluster-kafka-bootstrap:9093

如果要节流复制，您还可以传递 --throttle 选项，并传递 --throttle 选项，以字节为单位。例如：

```
bin/kafka-reassign-partitions.sh --bootstrap-server
  <cluster_name>-kafka-bootstrap:9093 \
  --command-config /tmp/config.properties \
  --reassignment-json-file /tmp/reassignment.json \
  --throttle 5000000 \
  --execute
```

此命令将输出两个重新分配 JSON 对象。第一个记录正在移动的分区的当前分配。如果您需要稍后恢复重新分配，您应该将其保存到本地文件（而非 pod 中的文件）。第二个 JSON 对象是在重新分配 JSON 文件中传递的目标重新分配。

如果您需要在重新分配过程中更改节流，您可以使用带有不同节流率相同的命令。例如：

```
bin/kafka-reassign-partitions.sh --bootstrap-server
  <cluster_name>-kafka-bootstrap:9093 \
  --command-config /tmp/config.properties \
  --reassignment-json-file /tmp/reassignment.json \
  --throttle 10000000 \
  --execute
```

7. 使用来自任何代理 pod 的 kafka-reassign-partitions.sh 命令行工具验证重新分配是否已完成。这与上一步中的命令相同，但使用 --verify 选项而不是 --execute 选项。

```
bin/kafka-reassign-partitions.sh --bootstrap-server
  <cluster_name>-kafka-bootstrap:9093 \
  --command-config /tmp/config.properties \
  --reassignment-json-file /tmp/reassignment.json \
  --verify
```

当 --verify 命令报告正在移动的每个分区都成功完成时，重新分配已完成。最后 --verify 也具有删除任何重新分配节流的影响。

8. 现在，如果您保存了 JSON 以将分配恢复到其原始代理，您可以删除恢复的文件。

20.4. 在删除代理前重新分配分区

在减少 Kafka 集群中的代理数量前，使用 kafka-reassign-partitions.sh 工具生成的重新分配文件来重新分配分区。重新分配文件必须描述如何将分区重新分配给 Kafka 集群中的剩余代理。您可以将文件中指定的重新分配应用到代理，然后验证新分区分配。首先删除编号最高的 pod 中的代理。

此流程描述了使用 TLS 的安全扩展过程。您需要一个使用 TLS 加密和 mTLS 身份验证的 Kafka 集群。

kafka-reassign-partitions.sh 工具可用于重新分配 Kafka 集群中的分区，无论您是否通过集群管理所有节点，还是使用节点池来管理集群中的节点组。

注意

虽然您可以使用 kafka-reassign-partitions.sh 工具，但建议使用 Cruise Control 进行自动分区重新分配和集群重新平衡。Cruise Control 可以在不停机的情况下将主题从一个代理移到另一个代理，这是重新分配分区的最高效方法。

先决条件

- 您有一个正在运行的 Kafka 集群，它基于配置了内部 TLS 加密和 mTLS 身份验证的 Kafka 资源。
- 您已生成了一个重新分配 JSON 文件，名为 reassignment.json。
- 您正在运行连接到正在运行的 Kafka 代理的交互式 pod 容器。

- 您作为配置了 ACL 规则的 KafkaUser 连接，该规则指定管理 Kafka 集群及其主题的权限。

流程

1. 如果没有这样做，请运行交互式 pod 容器来生成一个重新分配 JSON 文件，名为 reassignment.json。

2. 将 reassignment.json 文件复制到交互式 pod 容器中。

```
oc cp reassignment.json <interactive_pod_name>:/tmp/reassignment.json
```

将 *<interactive_pod_name>* 替换为 pod 的名称。

3. 在交互式 pod 容器中启动 shell 进程。

```
oc exec -n <namespace> -ti <interactive_pod_name> /bin/bash
```

将 *<namespace>* 替换为运行 pod 的 OpenShift 命名空间。

4. 使用交互式 pod 容器中的 kafka-reassign-partitions.sh 脚本运行分区重新分配。

```
bin/kafka-reassign-partitions.sh --bootstrap-server
 <cluster_name>-kafka-bootstrap:9093 \
 --command-config /tmp/config.properties \
 --reassignment-json-file /tmp/reassignment.json \
 --execute
```

将 *<cluster_name>* 替换为 Kafka 集群的名称。例如，my-cluster-kafka-bootstrap:9093

如果要节流复制，您还可以传递 --throttle 选项，并传递 --throttle 选项，以字节为单位。例如：

```
bin/kafka-reassign-partitions.sh --bootstrap-server
  <cluster_name>-kafka-bootstrap:9093 \
  --command-config /tmp/config.properties \
```

```
  --reassignment-json-file /tmp/reassignment.json \
  --throttle 5000000 \
  --execute
```

此命令将输出两个重新分配 JSON 对象。第一个记录正在移动的分区的当前分配。如果您需要稍后恢复重新分配，您应该将其保存到本地文件（而非 pod 中的文件）。第二个 JSON 对象是在重新分配 JSON 文件中传递的目标重新分配。

如果您需要在重新分配过程中更改节流，您可以使用带有不同节流率相同的命令。例如：

```
bin/kafka-reassign-partitions.sh --bootstrap-server
  <cluster_name>-kafka-bootstrap:9093 \
  --command-config /tmp/config.properties \
  --reassignment-json-file /tmp/reassignment.json \
  --throttle 10000000 \
  --execute
```

5. 使用来自任何代理 pod 的 kafka-reassign-partitions.sh 命令行工具验证重新分配是否已完成。这与上一步中的命令相同，但使用 --verify 选项而不是 --execute 选项。

```
bin/kafka-reassign-partitions.sh --bootstrap-server
  <cluster_name>-kafka-bootstrap:9093 \
  --command-config /tmp/config.properties \
  --reassignment-json-file /tmp/reassignment.json \
  --verify
```

当 --verify 命令报告正在移动的每个分区都成功完成时，重新分配已完成。最后 --verify 也具有删除任何重新分配节流的影响。

6. 现在，如果您保存了 JSON 以将分配恢复到其原始代理，您可以删除恢复的文件。

7. 当所有分区重新分配完成后，会删除代理不应负责集群中的任何分区。您可以通过检查代理的数据日志目录不包含任何实时分区日志来验证这一点。如果代理中的日志目录包含一个与扩展正则表达式 \.[a-z0-9]-delete$ 不匹配的目录，则代理仍然具有实时分区，不应停止。

您可以通过执行以下命令来检查它：

```
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 的数量。

如果上述命令打印任何输出，则代理仍有实时分区。在这种情况下，重新分配还没有完成，或者重新分配 JSON 文件不正确。

8. 当您确认代理没有实时分区时，您可以编辑 Kafka 资源的 Kafka.spec.kafka.replicas 属性，以减少代理数量。

20.5. 更改主题的复制因素

要更改 Kafka 集群中主题的复制因素，请使用 kafka-reassign-partitions.sh 工具。这可以通过从连接到 Kafka 集群的交互式 pod 容器运行工具，并使用重新分配文件描述应如何更改主题副本。

此流程描述了使用 TLS 的安全进程。您需要一个使用 TLS 加密和 mTLS 身份验证的 Kafka 集群。

先决条件

- 您有一个正在运行的 Kafka 集群，它基于配置了内部 TLS 加密和 mTLS 身份验证的 Kafka 资源。
- 您正在运行连接到正在运行的 Kafka 代理的交互式 pod 容器。
- 您已生成了一个重新分配 JSON 文件，名为 reassignment.json。
- 您作为配置了 ACL 规则的 KafkaUser 连接，该规则指定管理 Kafka 集群及其主题的权限。

请参阅 生成重新分配 JSON 文件。

在此过程中，名为 my-topic 的主题有 4 个副本，而我们希望将其减小到 3。名为 topics.json 的 JSON 文件指定主题，用于生成 reassignment.json 文件。

示例 JSON 文件指定 my-topic

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

流程

1. 如果没有这样做，请运行交互式 pod 容器来生成一个重新分配 JSON 文件，名为 reassignment.json。

显示当前和提议的副本分配的 JSON 文件示例

```
Current partition replica assignment
{"version":1,"partitions":[{"topic":"my-topic","partition":0,"replicas":
[3,4,2,0],"log_dirs":["any","any","any","any"]},{"topic":"my-
topic","partition":1,"replicas":[0,2,3,1],"log_dirs":["any","any","any","any"]},
{"topic":"my-topic","partition":2,"replicas":[1,3,0,4],"log_dirs":
["any","any","any","any"]}]}

Proposed partition reassignment configuration
{"version":1,"partitions":[{"topic":"my-topic","partition":0,"replicas":
[0,1,2,3],"log_dirs":["any","any","any","any"]},{"topic":"my-
topic","partition":1,"replicas":[1,2,3,4],"log_dirs":["any","any","any","any"]},
{"topic":"my-topic","partition":2,"replicas":[2,3,4,0],"log_dirs":
["any","any","any","any"]}]}
```

如果您需要稍后恢复更改，请将此文件的副本保存到本地。

2. 编辑 reassignment.json，以从每个分区中删除副本。

例如，使用 jq 命令行 JSON 解析器工具 删除主题的每个分区的列表中最后一个副本：

删除每个分区的最后一个主题副本

```
jq '.partitions[].replicas |= del(.[-1])' reassignment.json > reassignment.json
```

显示更新的副本的重新分配文件示例

```
{"version":1,"partitions":[{"topic":"my-topic","partition":0,"replicas":[0,1,2],"log_dirs":
["any","any","any","any"]},{"topic":"my-topic","partition":1,"replicas":
[1,2,3],"log_dirs":["any","any","any","any"]},{"topic":"my-
topic","partition":2,"replicas":[2,3,4],"log_dirs":["any","any","any","any"]}]}
```

3. 将 reassignment.json 文件复制到交互式 pod 容器中。

```
oc cp reassignment.json <interactive_pod_name>:/tmp/reassignment.json
```

将 *<interactive_pod_name>* 替换为 pod 的名称。

4. 在交互式 pod 容器中启动 shell 进程。

```
oc exec -n <namespace> -ti <interactive_pod_name> /bin/bash
```

将 *<namespace>* 替换为运行 pod 的 OpenShift 命名空间。

5. 使用交互式 pod 容器中的 kafka-reassign-partitions.sh 脚本进行主题副本更改。

```
bin/kafka-reassign-partitions.sh --bootstrap-server
 <cluster_name>-kafka-bootstrap:9093 \
 --command-config /tmp/config.properties \
 --reassignment-json-file /tmp/reassignment.json \
 --execute
```

注意

从代理中删除副本不需要任何代理数据移动，因此不需要节流复制。如果要添加副本，则可能需要更改节流率。

6. 验证对主题副本的更改已使用任何代理 pod 中的 kafka-reassign-partitions.sh 命令行工具完成。这与上一步中的命令相同，但使用 --verify 选项而不是 --execute 选项。

```
bin/kafka-reassign-partitions.sh --bootstrap-server
  <cluster_name>-kafka-bootstrap:9093 \
  --command-config /tmp/config.properties \
  --reassignment-json-file /tmp/reassignment.json \
  --verify
```

当 --verify 命令报告正在移动的每个分区都成功完成时，重新分配已完成。最后 --verify 也具有删除任何重新分配节流的影响。

7. 使用 --describe 选项运行 bin/kafka-topics.sh 命令，以查看对主题的更改结果。

```
bin/kafka-topics.sh --bootstrap-server
  <cluster_name>-kafka-bootstrap:9093 \
  --command-config /tmp/config.properties \
  --describe
```

减少主题副本数的结果

```
my-topic  Partition: 0  Leader: 0  Replicas: 0,1,2 Isr: 0,1,2
my-topic  Partition: 1  Leader: 2  Replicas: 1,2,3 Isr: 1,2,3
my-topic  Partition: 2  Leader: 3  Replicas: 2,3,4 Isr: 2,3,4
```

8. 最后，编辑 KafkaTopic 自定义资源，将 .spec.replicas 更改为 3，然后等待协调。

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

```
spec:
  partitions: 3
  replicas: 3
```

第 21 章 为 APACHE KAFKA 的 STREAMS 设置指标和仪表板

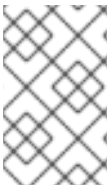
收集指标对于了解 Kafka 部署的健康状态和性能至关重要。通过监控指标，您可以在问题变得至关重要前主动识别问题，并根据资源分配和容量规划做出明智的决策。如果没有指标，您可能会对 Kafka 部署的行为有有限的可见性，这有助于进行故障排除。设置指标可节省长时间运行的时间和资源，并帮助确保 Kafka 部署的可靠性。

对于 Apache Kafka 的每个组件都提供了指标数据，它可让您了解其独立性能。虽然其他组件需要配置来公开指标暴露，但 Apache Kafka operator 的 Streams 默认自动公开 Prometheus 指标。这些指标包括：

- 协调计数
- 正在处理自定义资源数
- 协调持续时间
- JVM 指标

您还可以通过在 Kafka 资源的监听程序或授权配置中启用 enableMetrics 属性来收集特定于 oauth 身份验证和 opa 或 keycloak 授权的指标。同样，您可以在自定义资源中启用 oauth 身份验证的指标，如 KafkaBridge,KafkaConnect,KafkaMirrorMaker, 和 KafkaMirrorMaker2。

Apache Kafka 控制台的 Streams 提供了一个用户界面，用于监控 Kafka 集群中的指标。通过将由 Streams for Apache Kafka 管理的 Kafka 集群连接到 Apache Kafka 控制台的 Streams，您可以访问组件（如代理、主题、分区和消费者组）的详细信息。



注意

Apache Kafka 控制台的流当前作为技术预览提供。

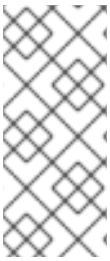
您还可以使用 Prometheus 和 Grafana 监控 Apache Kafka 的 Streams。当使用 Prometheus 规则配置时，Prometheus 会消耗集群中正在运行的 pod 的指标。Grafana 在仪表板中视觉化这些指标，为监控提供了一个直观的接口。

为方便指标集成，Apache Kafka 的 Streams 提供了 Apache Kafka 组件的 Streams 的 Prometheus 规则和 Grafana 仪表板示例。您可以自定义 Grafana 仪表板示例，以满足您的特定部署要求。您可以使用规则来定义根据特定指标触发警报的条件。

根据您的监控要求，您可以执行以下操作：

- 设置并部署 Prometheus 以公开指标
- 部署 Kafka Exporter 以提供额外指标
- 使用 Grafana 提供 Prometheus 指标

另外，您可以通过设置 分布式追踪，或使用 诊断工具 (report.sh)来检索故障排除数据，配置部署以跟踪消息端到端。



注意

Apache Kafka 的 Streams 为 Prometheus 和 Grafana 提供了示例安装文件，它们可作为监控 Apache Kafka 部署的 Streams 的起点。如需进一步支持，请尝试与 Prometheus 和 Grafana 开发人员社区互动。

支持指标和监控工具的文档

如需有关指标和监控工具的更多信息，请参阅支持文档：

- Prometheus
- Prometheus 配置
- Kafka Exporter
- Grafana Labs

- Apache Kafka Monitoring 描述了 Apache Kafka 公开的 JMX 指标
- ZooKeeper JMX 描述了 Apache ZooKeeper 公开的 JMX 指标

21.1. 使用 KAFKA EXPORTER 监控消费者滞后

Kafka Exporter 是一个开源项目，用于增强对 Apache Kafka 代理和客户端的监控。您可以配置 Kafka 资源，以使用 Kafka 集群部署 Kafka 导出器。Kafka Exporter 从与偏移、消费者组、消费者滞后和主题相关的 Kafka 代理中提取额外的指标数据。例如，使用指标数据来帮助识别速度较慢的用户。lag 数据作为 Prometheus 指标公开，然后可在 Grafana 中显示这些指标数据进行分析。

Kafka Exporter 从 __consumer_offsets 主题读取，该主题存储消费者组的提交偏移信息。要使 Kafka Exporter 能够正常工作，需要使用消费者组。

Kafka Exporter 的 Grafana 仪表板是 Apache Kafka 的 Streams 提供的多个 Grafana 仪表板示例 之一。

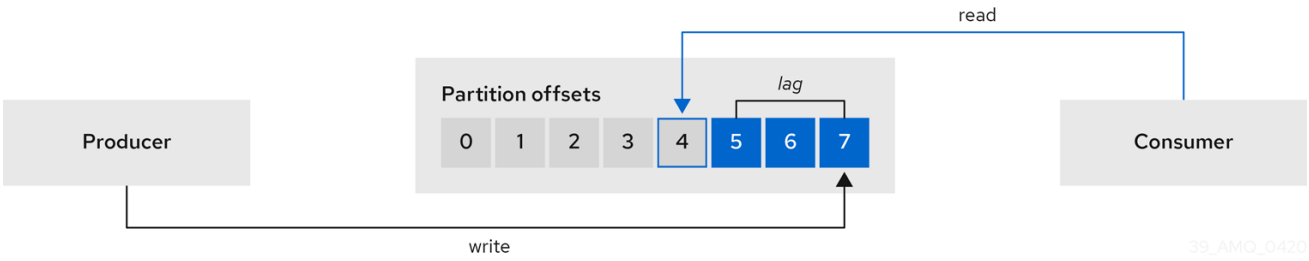
重要

Kafka Exporter 只提供与消费者滞后和消费者偏移相关的额外指标。对于常规 Kafka 指标，您必须在 Kafka 代理 中配置 Prometheus 指标。

消费者滞后表示消息的速度与消息的消耗的差别。具体来说，给定消费者组的消费者滞后指示分区中最后一个消息与当前由该消费者获取的消息之间的延迟。

lag 反映了与分区日志末尾相关的消费者偏移位置。

生成者和消费者偏移之间的消费者滞后



这种差异有时被称为生成者偏移和消费者偏移之间的 ***delta*** ： Kafka 代理主题分区中的读取和写入位置。

假设主题将 100 个消息流过一秒。生成者偏移（主题分区头）和消费者读取的最后偏移之间的 1000 个消息表示有 10 秒的延迟。

监控消费者滞后的重要性

对于依赖于实时数据的处理的应用程序，监控消费者来判断其是否不会变得太大。整个过程越好，流程从实时处理目标中得到的增长。

例如，消费者滞后可能是消耗太多的旧数据（这些数据尚未被清除）或出现计划外的关闭。

减少消费者滞后

使用 Grafana chart 分析滞后，检查操作是否对受影响的消费者组产生影响。例如，如果对 Kafka 代理进行了调整以减少滞后，仪表板将显示 ***Lag by consumer group*** 图表下降， ***Messages consumed per minute*** 图表增加。

减少滞后的典型操作包括：

- 通过添加新消费者来扩展消费者组
- 增加消息的保留时间，以保留在主题中
- 添加更多磁盘容量以增加消息缓冲

减少消费者滞后的操作取决于底层基础架构，支持 Apache Kafka 的用例流。例如，分发的消费者不太

可能从其磁盘缓存中获取请求的服务从代理服务。在某些情况下，可能可以接受自动丢弃信息，直到消费者发现为止。

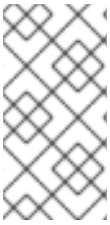
21.2. 监控 CRUISE CONTROL 操作

Cruise Control 监控 Kafka 代理，以跟踪代理、主题和分区的利用率。Cruise Control 还提供一组指标来监控其自身的性能。

Cruise Control 指标报告器从 Kafka 代理收集原始指标数据。数据由 Cruise Control 自动创建的主题生成。指标 用于生成 Kafka 集群的优化建议。

Cruise Control 指标可用于实时监控 Cruise Control 操作。例如，您可以使用 Cruise Control 指标监控运行重新平衡操作的状态，或在操作性能中检测到的任何异常情况提供警报。

您可以通过在 Cruise Control 配置中启用 Prometheus JMX Exporter 来公开 Cruise Control 指标。



注意

有关可用 Cruise Control 指标的完整列表，称为 ***sensors***，请参阅 Cruise Control 文档

21.2.1. 监控平衡分数

Cruise Control 指标包括平衡分数。Balancedness 是 Kafka 集群中平均分布工作负载的测量。

平衡分数的 Cruise Control 指标(balancedness-score)可能与 KafkaRebalance 资源中的均衡分数不同。Cruise Control 使用 anomaly.detection.goals 计算每个分数，这些分数可能与 KafkaRebalance 资源中使用的 default.goals 不同。anomaly.detection.goals 在 Kafka 自定义资源的 spec.cruiseControl.config 中指定。

注意

刷新 KafkaRebalance 资源获取优化提议。如果满足以下条件之一，则会获取最新的缓存的优化方案：

- KafkaRebalance 目标 与 Kafka 资源的 default.goals 部分中配置的目标匹配
- 未指定 KafkaRebalance 目标

否则，Cruise Control 会根据 KafkaRebalance 目标 生成新的优化建议。如果每次刷新时生成新提议，这可能会影响性能监控。

21.2.2. 为异常检测设置警报

Cruise Control 的 *omaly detector* 提供了阻止生成优化目标（如代理故障）的条件的指标数据。如果要提高可见性，您可以使用 anomaly detector 提供的指标来设置警报并发送通知。您可以设置 Cruise Control 的 *anomaly notifier*，以通过指定的通知频道根据这些指标路由警报。另外，您可以设置 Prometheus 来提取 anomaly detector 提供的指标数据并生成警报。然后 Prometheus Alertmanager 可以路由 Prometheus 生成的警报。

Cruise Control 文档 提供有关 AnomalyDetector 指标和 anomaly notifier 的信息。

21.3. 指标文件示例

您可以在 Apache Kafka 的 Streams 提供 的示例配置文件 中找到 Grafana 仪表板和其他指标配置文件示例。

由 Apache Kafka Streams 提供的指标文件示例

```
metrics
├── grafana-dashboards 1
│   ├── strimzi-cruise-control.json
│   ├── strimzi-kafka-bridge.json
│   ├── strimzi-kafka-connect.json
│   ├── strimzi-kafka-exporter.json
│   ├── strimzi-kafka-mirror-maker-2.json
│   ├── strimzi-kafka.json
```

```
│   ├── strimzi-operators.json
│   └── strimzi-zookeeper.json
├── grafana-install
│   └── grafana.yaml 2
├── prometheus-additional-properties
│   └── prometheus-additional.yaml 3
├── prometheus-alertmanager-config
│   └── alert-manager-config.yaml 4
├── prometheus-install
│   ├── alert-manager.yaml 5
│   ├── prometheus-rules.yaml 6
│   ├── prometheus.yaml 7
│   └── strimzi-pod-monitor.yaml 8
├── kafka-bridge-metrics.yaml 9
├── kafka-connect-metrics.yaml 10
├── kafka-cruise-control-metrics.yaml 11
├── kafka-metrics.yaml 12
└── kafka-mirror-maker-2-metrics.yaml 13
```

1

Apache Kafka 组件的不同流的 Grafana 仪表板示例。

2

Grafana 镜像的安装文件。

3

用于提取 CPU、内存和磁盘卷使用情况的指标的额外配置，它们直接来自节点上的 OpenShift cAdvisor 代理和 kubelet。

4

用于通过 Alertmanager 发送通知的 hook 定义。

5

用于部署和配置 Alertmanager 的资源。

6

与 Prometheus Alertmanager 搭配使用的警报规则示例（使用 Prometheus 部署）。

7

8

Prometheus Operator 将 PodMonitor 定义转换为 Prometheus 服务器的作业，以便直接从 Pod 中提取指标数据。

9

启用指标的 Kafka Bridge 资源。

10

为 Kafka Connect 定义 Prometheus JMX Exporter 重新标记规则的指标配置。

11

为 Cruise 控制定义 Prometheus JMX Exporter 重新标记规则的指标配置。

12

为 Kafka 和 ZooKeeper 定义 Prometheus JMX Exporter 重新标记规则的指标配置。

13

为 Kafka MirrorMaker 2 定义 Prometheus JMX Exporter 重新标记规则的指标配置。

21.3.1. Prometheus 指标配置示例

Apache Kafka 的 Streams 使用 Prometheus JMX Exporter 通过 HTTP 端点公开指标，该端点可以被 Prometheus 服务器提取。

Grafana 仪表板依赖于 Prometheus JMX Exporter 重新标记规则，这些规则在自定义资源配置中为 Apache Kafka 组件定义。

标签是一个 name-value 对。重新标记是动态写入标签的过程。例如，标签的值可能源自 Kafka 服务器和客户端 ID 的名称。

Apache Kafka 的流提供了带有重新标记规则的自定义资源配置 YAML 文件示例。在部署 Prometheus 指标配置时，您可以部署示例自定义资源，或者将指标配置复制到您自己的自定义资源定义中。

表 21.1. 带有指标配置的自定义资源示例

组件	自定义资源	YAML 文件示例
Kafka 和 ZooKeeper	**Kafka**	**kafka-metrics.yaml**
Kafka Connect	**KafkaConnect**	**kafka-connect-metrics.yaml**
Kafka MirrorMaker 2	**KafkaMirrorMaker2**	**kafka-mirror-maker-2-metrics.yaml**
Kafka Bridge	**KafkaBridge**	**kafka-bridge-metrics.yaml**
Sything Control	**Kafka**	**kafka-cruise-control-metrics.yaml**

21.3.2. 警报通知的 Prometheus 规则示例

警报通知的 Prometheus 规则示例提供了 Apache Kafka 的 Streams 提供的 指标配置文件示例。规则在 prometheus-rules.yaml 示例中指定，用于在 Prometheus 部署中使用。

prometheus-rules.yaml 文件包含以下组件的示例规则：

- **Kafka**
- **ZooKeeper**
- **Entity Operator**
- **Kafka Connect**
- **Kafka Bridge**
- **MirrorMaker**

- Kafka Exporter

文件中提供了每个示例规则的描述。

警报规则提供有关指标中观察到的特定条件的通知。在 Prometheus 服务器上声明规则，但 Prometheus Alertmanager 负责警报通知。

Prometheus 警报规则使用 PromQL 表达式来持续评估的条件。

当警报表达式变为 true 时，条件会被满足，Prometheus 服务器会将警报数据发送到 Alertmanager。然后，Alertmanager 使用为其部署配置的通信方法发送通知。

有关警报规则定义的一般点：

- for 属性与规则一起使用，以确定条件在触发警报前必须保留的时间周期。
- tick 是一个基本的 ZooKeeper 时间单元，它以毫秒为单位测量，并使用 Kafka.spec.zookeeper.config 的 tickTime 参数进行配置。例如，如果 ZooKeeper tickTime=3000，3 ticks (3 x 3000)等于 9000 毫秒。
- ZookeeperRunningOutOfSpace 指标和警报的可用性取决于所使用的 OpenShift 配置和存储实现。某些平台的存储实现可能无法提供指标提供警报所需的可用空间的信息。

Alertmanager 可以配置为使用电子邮件、聊天消息或其他通知方法。根据您的特定需求，调整示例规则的默认配置。

21.3.3. Grafana 仪表板示例

如果部署 Prometheus 以提供指标，您可以使用 Apache Kafka 提供的流 Grafana 仪表板示例来监控 Apache Kafka 组件的 Streams。

示例仪表板在 examples/metrics/grafana-dashboards 目录中作为 JSON 文件提供。

所有仪表板都提供 JVM 指标，以及特定于组件的指标。例如，Apache Kafka operator 的 Streams 的 Grafana 仪表板提供有关它们正在处理的协调或自定义资源数量的信息。

示例仪表板不显示 Kafka 支持的所有指标。仪表板填充了用于监控的指标集合。

表 21.2. Grafana 仪表板文件示例

组件	JSON 文件示例
Apache Kafka operator 的流	**strimzi-operators.json**
Kafka	**strimzi-kafka.json**
ZooKeeper	**strimzi-zookeeper.json**
Kafka Connect	**strimzi-kafka-connect.json**
Kafka MirrorMaker 2	**strimzi-kafka-mirror-maker-2.json**
Kafka Bridge	**strimzi-kafka-bridge.json**
Sything Control	**strimzi-cruise-control.json**
Kafka Exporter	**strimzi-kafka-exporter.json**

注意

当 Kafka Exporter 不可用指标时，因为集群中没有流量，Kafka Exporter Grafana 仪表板将显示 N/A 用于数字字段，且没有 显示图形的数据。

21.4. 通过配置启用 PROMETHEUS 指标

要为 Prometheus 在 Apache Kafka 中启用和公开指标，请使用指标配置属性。

以下组件需要 metricsConfig 配置来公开指标：

- Kafka

- KafkaConnect
- MirrorMaker
- Sything Control
- ZooKeeper

此配置可让 Prometheus JMX Exporter 通过 HTTP 端点公开指标。JMX 导出器 HTTP 端点的端口是 9404。Prometheus 提取此端点来收集 Kafka 指标。

您可以将 enableMetrics 属性设置为 true，以便为这些组件公开指标：

- Kafka Bridge
- OAuth 2.0 身份验证和授权框架
- 用于授权的开放策略代理(OPA)

要在 Apache Kafka 的 Streams 中部署 Prometheus 指标配置，您可以使用自己的配置或 Apache Kafka 提供的 Streams 提供的示例自定义资源 配置文件 ：

- kafka-metrics.yaml
- kafka-connect-metrics.yaml
- kafka-mirror-maker-2-metrics.yaml
- kafka-bridge-metrics.yaml

- kafka-cruise-control-metrics.yaml
- oauth-metrics.yaml

这些文件包含启用 Prometheus 指标所需的重新标记规则和配置。对于使用 Apache Kafka 的 Streams 尝试 Prometheus，它们是很好的起点。

此流程演示了如何在 Kafka 资源中部署 Prometheus 指标配置示例。为其他资源部署示例文件时，此过程相同。

如果要包含 Kafka Exporter 指标，请在 Kafka 资源中添加 kafkaExporter 配置。

重要

Kafka Exporter 仅提供额外的与消费者滞后相关的指标。对于常规 Kafka 指标，您必须在 Kafka 代理 中配置 Prometheus 指标。

流程

1. 使用 Prometheus 配置部署示例自定义资源。

   例如，对于每个 Kafka 资源，您可以应用 kafka-metrics.yaml 文件。

   部署示例配置

   ```
   oc apply -f kafka-metrics.yaml
   ```

   另外，您可以将 kafka-metrics.yaml 中的示例配置复制到您自己的 Kafka 资源。

   复制示例配置

```
oc edit kafka <kafka_configuration_file>
```

复制 metricsConfig 属性及其引用的 ConfigMap 到 Kafka 资源。

Kafka 的指标配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    metricsConfig: 1
      type: jmxPrometheusExporter
      valueFrom:
        configMapKeyRef:
          name: kafka-metrics
          key: kafka-metrics-config.yml
---
kind: ConfigMap 2
apiVersion: v1
metadata:
  name: kafka-metrics
  labels:
    app: strimzi
data:
  kafka-metrics-config.yml: |
  # metrics configuration...
```

1

复制引用包含指标配置的 ConfigMap 的 metricsConfig 属性。

2

复制指定指标配置的整个 ConfigMap。

2. 要部署 Kafka Exporter，请添加 kafkaExporter 配置。

kafkaExporter 配置仅在 Kafka 资源中指定。

部署 Kafka Exporter 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  kafkaExporter:
    image: my-registry.io/my-org/my-exporter-cluster:latest 1
    groupRegex: ".*" 2
    topicRegex: ".*" 3
    groupExcludeRegex: "^excluded-.*" 4
    topicExcludeRegex: "^excluded-.*" 5
    resources: 6
      requests:
        cpu: 200m
        memory: 64Mi
      limits:
        cpu: 500m
        memory: 128Mi
    logging: debug 7
    enableSaramaLogging: true 8
    template: 9
      pod:
        metadata:
          labels:
            label1: value1
        imagePullSecrets:
          - name: my-docker-credentials
        securityContext:
          runAsUser: 1000001
          fsGroup: 0
        terminationGracePeriodSeconds: 120
    readinessProbe: 10
      initialDelaySeconds: 15
      timeoutSeconds: 5
    livenessProbe: 11
      initialDelaySeconds: 15
      timeoutSeconds: 5
# ...
```

1

ProductShortName OPTION: 容器镜像配置，仅在特殊情况下推荐使用。

2

指定指标中包含的消费者组的正则表达式。

3

指定指标中包含的主题的正则表达式。

4

指定指标中要排除的消费者组的正则表达式。

5

在指标中指定要排除的主题的正则表达式。

6

要保留的 CPU 和内存资源。

7

日志记录配置，以记录给定严重性(debug、info、warn、error、fatal)或更高严重性的消息。

8

启用 Sarama 日志的布尔值，这是 Kafka Exporter 使用的 Go 客户端库。

9

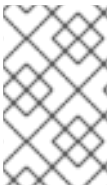
自定义部署模板和 pod。

10

健康检查就绪度探测。

11

健康检查存活度探测。



注意

要使 Kafka Exporter 能够正常工作，需要使用消费者组。

为 Kafka Bridge 启用指标

要为 Kafka Bridge 公开指标，请在 KafkaBridge 资源中将 enableMetrics 属性设置为 true。

Kafka Bridge 的指标配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  bootstrapServers: my-cluster-kafka:9092
  http:
    # ...
  enableMetrics: true
  # ...
```

为 OAuth 2.0 和 OPA 启用指标

要公开 OAuth 2.0 或 OPA 的指标，请在适当的自定义资源中将 enableMetrics 属性设置为 true。

OAuth 2.0 指标

在 Kafka 资源中为 Kafka 集群授权和 Kafka 侦听器身份验证启用指标。

您还可以在其他 受支持组件 的自定义资源中启用 OAuth 2.0 身份验证的指标。

opa 指标

为 Kafka 集群授权启用指标与 OAuth 2.0 相同。

在以下示例中，为 OAuth 2.0 侦听器身份验证和 OAuth 2.0 (keycloak)集群授权启用指标。

为 OAuth 2.0 启用指标的集群配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    listeners:
    - name: external3
      port: 9094
      type: loadbalancer
      tls: true
      authentication:
        type: oauth
        enableMetrics: true
      configuration:
        #...
    authorization:
      type: keycloak
      enableMetrics: true
  # ...
```

要将 OAuth 2.0 指标与 Prometheus 搭配使用，您可以使用 oauth-metrics.yaml 文件来部署示例 Prometheus 指标配置。将 ConfigMap 配置复制 oauth-metrics.yaml 文件包含到为 OAuth 2.0 启用指标的同一 Kafka 资源配置文件。

21.5. 在 OPENSHIFT 中查看 KAFKA 指标和仪表板

当 Apache Kafka 的流部署到 OpenShift Container Platform 中时，通过 *监控用户定义的项目* 来提供指标数据。此 OpenShift 功能使开发人员能够访问单独的 Prometheus 实例，用于监控自己的项目（如 Kafka 项目）。

如果启用了对用户定义的项目的监控，openshift-user-workload-monitoring 项目将包含以下组件：

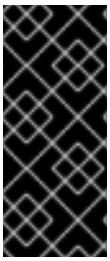
- Prometheus operator
- Prometheus 实例（由 Prometheus Operator 自动部署）
- Thanos Ruler 实例

Apache Kafka 的流使用这些组件消耗指标。

集群管理员必须为用户定义的项目启用监控，然后授予开发人员和其他用户权限来监控其自己的项目中的应用程序。

Grafana 部署

您可以将 Grafana 实例部署到包含 Kafka 集群的项目中。然后，Grafana 仪表板示例可用于视觉化 Grafana 用户界面中 Apache Kafka 的 Streams 的 Prometheus 指标。



重要

openshift-monitoring 项目为核心平台组件提供监控。*不要使用* 此项目中的 Prometheus 和 Grafana 组件，在 OpenShift Container Platform 4.x 上为 Apache Kafka 配置监控。

流程概述

要在 OpenShift Container Platform 中设置 Apache Kafka 监控的 Streams，请按照以下步骤执行：

1. 先决条件：部署 Prometheus 指标配置
2. 部署 Prometheus 资源

3. 为 Grafana 创建服务帐户

4. 使用 Prometheus 数据源部署 Grafana

5. 创建到 Grafana 服务的路由

6. 导入 Grafana 仪表板示例

21.5.1. 先决条件

- 您已使用示例 YAML 文件 部署了 Prometheus 指标配置。
- *启用对用户定义的项目的监控*。集群管理员已在 OpenShift 集群中创建了一个 cluster-monitoring-config 配置映射。
- 集群管理员已为您分配一个 monitoring-rules-edit 或 monitoring-edit 角色。

有关创建 cluster-monitoring-config 配置映射并授予用户权限来监控用户定义的项目的的更多信息，请参阅 OpenShift 文档。

21.5.2. 部署 Prometheus 资源

使用 Prometheus 获取 Kafka 集群中的监控数据。

您可以使用自己的 Prometheus 部署，或使用 Apache Kafka 的 Streams 提供的 指标配置文件 部署 Prometheus。要使用示例文件，请配置和部署 PodMonitor 资源。PodMonitor 直接从 Apache Kafka, ZooKeeper, Operators, Kafka Bridge, 和 Cruise Control 的 pod 中提取数据。

然后，您要为 Alertmanager 部署示例警报规则。

先决条件

- 正在运行的 Kafka 集群。
- 检查 流为 Apache Kafka 提供的示例警报规则。

流程

1. 检查是否启用了监控用户定义的项目：

```
oc get pods -n openshift-user-workload-monitoring
```

如果启用，则返回监控组件的 pod。例如：

```
NAME                                  READY   STATUS    RESTARTS   AGE
prometheus-operator-5cc59f9bc6-kgcq8   1/1     Running   0          25s
prometheus-user-workload-0             5/5     Running   1          14s
prometheus-user-workload-1             5/5     Running   1          14s
thanos-ruler-user-workload-0           3/3     Running   0          14s
thanos-ruler-user-workload-1           3/3     Running   0          14s
```

如果没有返回 pod，则禁用对用户定义的项目的监控。请参阅 第 21.5 节 “在 OpenShift 中查看 Kafka 指标和仪表板” 中的先决条件。

2. 在 examples/metrics/prometheus-install/strimzi-pod-monitor.yaml 中定义多个 PodMonitor 资源。

对于每个 PodMonitor 资源，编辑 spec.namespaceSelector.matchNames 属性：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: cluster-operator-metrics
  labels:
    app: strimzi
spec:
  selector:
    matchLabels:
      strimzi.io/kind: cluster-operator
  namespaceSelector:
    matchNames:
      - <project-name> 1
  podMetricsEndpoints:
```

```
    - path: /metrics
      port: http
  # ...
```

1

从其中提取指标的 pod 正在运行的项目，如 Kafka。

3. 将 strimzi-pod-monitor.yaml 文件部署到运行 Kafka 集群的项目中：

```
oc apply -f strimzi-pod-monitor.yaml -n MY-PROJECT
```

4. 将示例 Prometheus 规则部署到同一项目中：

```
oc apply -f prometheus-rules.yaml -n MY-PROJECT
```

21.5.3. 为 Grafana 创建服务帐户

Apache Kafka 的 Streams 的 Grafana 实例需要使用分配了 cluster-monitoring-view 角色的服务帐户运行。

如果您使用 Grafana 显示监控指标，请创建一个服务帐户。

先决条件

- 部署 Prometheus 资源

流程

1. 在包含 Kafka 集群的项目中为 Grafana 创建 ServiceAccount ：

```
oc create sa grafana-service-account -n my-project
```

在本例中，在 my-project 命名空间中创建一个名为 grafana-service-account 的服务帐户。

2. 创建一个 ClusterRoleBinding 资源，将 cluster-monitoring-view 角色分配给 Grafana

ServiceAccount。此处的资源名为 grafana-cluster-monitoring-binding。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: grafana-cluster-monitoring-binding
  labels:
    app: strimzi
subjects:
  - kind: ServiceAccount
    name: grafana-service-account
    namespace: my-project
roleRef:
  kind: ClusterRole
  name: cluster-monitoring-view
  apiGroup: rbac.authorization.k8s.io
```

3. 将 ClusterRoleBinding 部署到同一项目中：

```
oc apply -f grafana-cluster-monitoring-binding.yaml -n my-project
```

4. 为服务帐户创建令牌 secret：

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-sa
  annotations:
    kubernetes.io/service-account.name: "grafana-service-account" (1)
type: kubernetes.io/service-account-token (2)
```

(1) 指定服务帐户。

(2) 指定服务帐户令牌 secret。

5. 创建 Secret 对象和访问令牌：

```
oc create -f <secret_configuration>.yaml
```

部署 Grafana 时需要访问令牌。

21.5.4. 使用 Prometheus 数据源部署 Grafana

部署 Grafana 以显示 Prometheus 指标。Grafana 应用程序需要配置 OpenShift Container Platform 监控堆栈。

OpenShift Container Platform 在 openshift-monitoring 项目中包含一个 *Thanos Querier* 实例。Thanos Querier 用于聚合平台指标。

要使用所需的平台指标，Grafana 实例需要一个可以连接到 Thanos Querier 的 Prometheus 数据源。要配置此连接，您可以创建一个配置映射来使用令牌进行身份验证，到与 Thanos Querier 一起运行的 oauth-proxy sidecar。datasource.yaml 文件被用作配置映射的来源。

最后，您可以使用作为卷挂载到包含 Kafka 集群的项目的配置映射部署 Grafana 应用程序。

先决条件

- 您已部署了 Prometheus 资源。
- 您已为 Grafana 创建了服务帐户。

流程

1. 获取 Grafana ServiceAccount 的访问令牌：

```
oc describe sa/grafana-service-account | grep Tokens:
oc describe secret grafana-service-account-token-mmlp9 | grep token:
```

在本例中，服务帐户名为 grafana-service-account。复制要在下一步中使用的访问令牌。

2. 创建包含 Grafana 的 Thanos Querier 配置的 datasource.yaml 文件。

将访问令牌粘贴到 httpHeaderValue1 属性中，如下所示。

```
apiVersion: 1

datasources:
- name: Prometheus
  type: prometheus
  url: https://thanos-querier.openshift-monitoring.svc.cluster.local:9091
  access: proxy
  basicAuth: false
  withCredentials: false
  isDefault: true
  jsonData:
    timeInterval: 5s
    tlsSkipVerify: true
    httpHeaderName1: "Authorization"
  secureJsonData:
    httpHeaderValue1: "Bearer ${GRAFANA-ACCESS-TOKEN}" (1)
  editable: true
```

(1) *GRAFANA-ACCESS-TOKEN* ：Grafana ServiceAccount 的访问令牌值。

3. 从 datasource.yaml 文件创建一个名为 grafana-config 的配置映射：

```
oc create configmap grafana-config --from-file=datasource.yaml -n MY-PROJECT
```

4. 创建由 Deployment 和 Service 组成的 Grafana 应用程序。

grafana-config 配置映射作为数据源配置的卷挂载。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
  labels:
    app: strimzi
spec:
  replicas: 1
  selector:
    matchLabels:
      name: grafana
  template:
    metadata:
      labels:
        name: grafana
    spec:
      serviceAccountName: grafana-service-account
      containers:
```

```
      - name: grafana
        image: grafana/grafana:10.4.2
        ports:
        - name: grafana
          containerPort: 3000
          protocol: TCP
        volumeMounts:
        - name: grafana-data
          mountPath: /var/lib/grafana
        - name: grafana-logs
          mountPath: /var/log/grafana
        - name: grafana-config
          mountPath: /etc/grafana/provisioning/datasources/datasource.yaml
          readOnly: true
          subPath: datasource.yaml
        readinessProbe:
          httpGet:
            path: /api/health
            port: 3000
          initialDelaySeconds: 5
          periodSeconds: 10
        livenessProbe:
          httpGet:
            path: /api/health
            port: 3000
          initialDelaySeconds: 15
          periodSeconds: 20
      volumes:
      - name: grafana-data
        emptyDir: {}
      - name: grafana-logs
        emptyDir: {}
      - name: grafana-config
        configMap:
          name: grafana-config
---
apiVersion: v1
kind: Service
metadata:
  name: grafana
  labels:
    app: strimzi
spec:
  ports:
  - name: grafana
    port: 3000
    targetPort: 3000
    protocol: TCP
  selector:
    name: grafana
  type: ClusterIP
```

5. 将 Grafana 应用程序部署到包含 Kafka 集群的项目中：

```
oc apply -f <grafana-application> -n <my-project>
```

21.5.5. 创建到 Grafana 服务的路由

您可以通过公开 Grafana 服务的路由访问 Grafana 用户界面。

先决条件

- 部署 Prometheus 资源
- 为 Grafana 创建服务帐户
- 使用 Prometheus 数据源部署 Grafana

流程

- 创建到 grafana 服务的边缘路由：

```
oc create route edge <my-grafana-route> --service=grafana --namespace=KAFKA-NAMESPACE
```

21.5.6. 导入 Grafana 仪表板示例

使用 Grafana 在可自定义仪表板上提供 Prometheus 指标的视觉化。

Apache Kafka 的 Streams 以 JSON 格式提供 Grafana 的示例仪表板配置文件。

- examples/metrics/grafana-dashboards

此流程使用 Grafana 仪表板示例。

示例仪表板是监控关键指标的良好起点，但它们不会显示 Kafka 支持的所有指标。您可以根据基础架构修改示例仪表板或添加其他指标。

先决条件

- 部署 Prometheus 资源
- 为 Grafana 创建服务帐户
- 使用 Prometheus 数据源部署 Grafana
- 创建到 Grafana 服务的路由

流程

1. 获取到 Grafana 服务的路由详情。例如：

```
oc get routes

NAME              HOST/PORT                        PATH  SERVICES
MY-GRAFANA-ROUTE  MY-GRAFANA-ROUTE-amq-streams.net       grafana
```

2. 在 Web 浏览器中，使用 Route 主机和端口的 URL 访问 Grafana 登录屏幕。
3. 输入您的用户名和密码，然后单击 Log In。

   默认的 Grafana 用户名和密码都是 admin。第一次登录后，您可以更改密码。
4. 在 Configuration > Data Sources 中，检查是否创建了 Prometheus 数据源。数据源是在 第 21.5.4 节 “使用 Prometheus 数据源部署 Grafana” 中创建的。
5. 单击 + 图标，然后单击 Import。
6. 在 examples/metrics/grafana-dashboards 中，复制要导入的仪表板的 JSON。
7. 将 JSON 粘贴到文本框中，然后点 Load。

8. 为其他 Grafana 仪表板重复步骤 5-7。

从 Dashboards 主页中查看导入的 Grafana 仪表板。

第 22 章 分布式追踪简介

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

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

以下 Kafka 组件内置了对追踪的支持：

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

Kafka 代理不支持追踪。

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

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

OpenTelemetry

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

小心

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

22.1. 追踪选项

在 Jaeger tracing 系统中使用 OpenTelemetry。

OpenTelemetry 提供独立于追踪或监控系统的 API 规格。

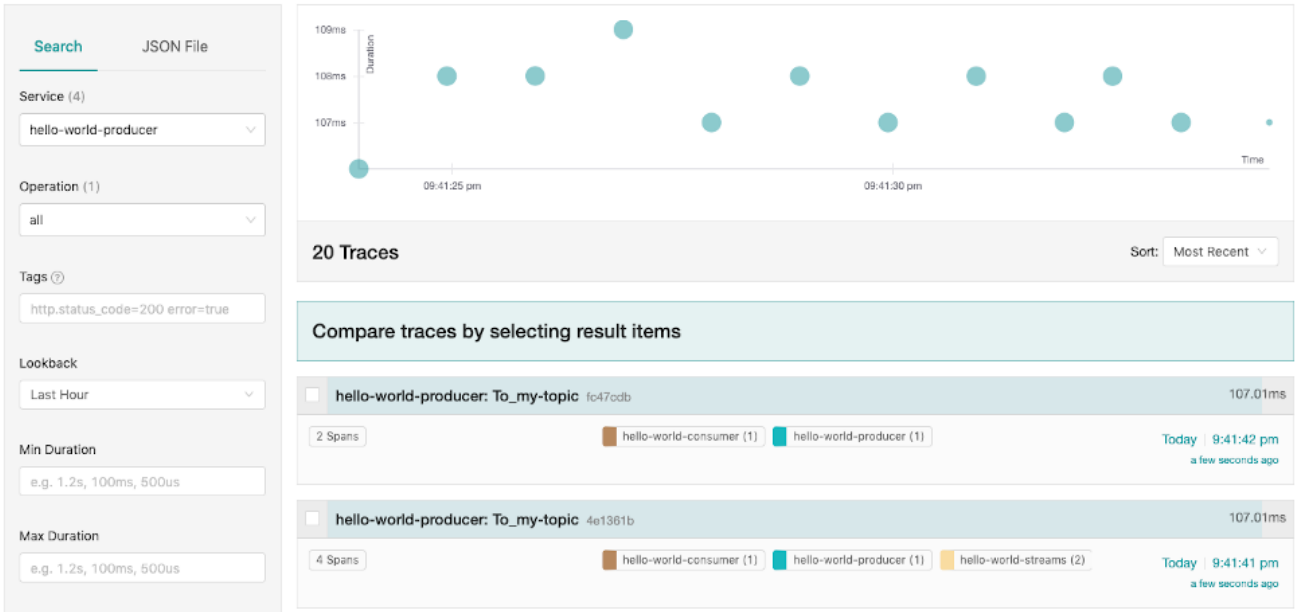
您可以使用 API 检测应用程序代码以进行追踪。

- 检测的应用程序会为分布式系统中的单个请求生成 *trace*。
- 跟踪由 *范围* 组成，用于定义一段时间内的特定工作单元。

Jaeger 是基于微服务的分布式系统的追踪系统。

- Jaeger 用户界面允许您查询、过滤和分析 trace 数据。

Jaeger 用户界面显示一个简单的查询



其他资源

- Jaeger 文档
- OpenTelemetry 文档

22.2. 用于追踪的环境变量

在为 Kafka 组件启用追踪或为 Kafka 客户端初始化 tracer 时，请使用环境变量。

追踪环境变量可能会改变。有关最新信息，请参阅 OpenTelemetry 文档。

下表描述了用于设置 tracer 的关键环境变量。

表 22.1. OpenTelemetry 环境变量

属性	必需	Description
OTEL_SERVICE_NAME	是	OpenTelemetry 的 Jaeger tracing 服务的名称。
OTEL_EXPORTER_JAEGER_ENDPOINT	是	用于追踪的导出器。
OTEL_TRACES_EXPORTER	是	用于追踪的导出器。默认设置为 **otlp**。如果使用 Jaeger tracing，则需要将此环境变量设置为 **jaeger**。如果您使用另一个追踪实现，请指定使用的导出器。

22.3. 设置分布式追踪

通过在自定义资源中指定追踪类型，在 Kafka 组件中启用分布式追踪。Kafka 客户端中的检测追踪程序用于对消息的端到端跟踪。

要设置分布式追踪，请遵循以下步骤：

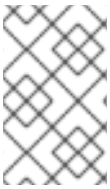
- 为 MirrorMaker、Kafka Connect 和 Kafka Bridge 启用追踪

- 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：
  - 用于追踪的工具生成者和消费者
  - 用于追踪的工具 Kafka Streams 应用程序

22.3.1. 先决条件

在设置分布式追踪前，请确保 Jaeger 后端组件部署到 OpenShift 集群中。我们建议使用 Jaeger operator 在 OpenShift 集群上部署 Jaeger。

有关部署说明，请参阅 Jaeger 文档。



注意

对于除 Apache Kafka 的流外的应用程序和系统设置追踪不在此内容范围内。

22.3.2. 在 MirrorMaker、Kafka Connect 和 Kafka Bridge 资源中启用追踪

MirrorMaker、MirrorMaker 2、Kafka Connect 和 Apache Kafka Bridge 的 Streams 支持分布式追踪。配置组件的自定义资源，以指定和启用 tracer 服务。

在资源中启用追踪会触发以下事件：

- 拦截器类在组件的集成使用者和制作者中更新。
- 对于 MirrorMaker、MirrorMaker 2 和 Kafka Connect，追踪代理会根据资源中定义的追踪配置初始化 tracer。

- 对于 Kafka Bridge，基于资源中定义的追踪配置由 Kafka Bridge 本身初始化的 tracer。

您可以启用使用 OpenTelemetry 的追踪。

MirrorMaker 和 MirrorMaker 2 中的追踪

对于 MirrorMaker 和 MirrorMaker 2，信息从源集群追踪到目标集群。跟踪数据记录进入和离开 MirrorMaker 或 MirrorMaker 2 组件的消息。

Kafka Connect 中的追踪

对于 Kafka Connect，只有 Kafka Connect 生成和消耗的消息才会被 traced。要跟踪 Kafka Connect 和外部系统之间发送的消息，您必须在连接器中为这些系统配置追踪。

Kafka Bridge 中的追踪

对于 Kafka Bridge，Kafka Bridge 生成和消耗的消息会被跟踪。也跟踪来自客户端应用程序的 HTTP 请求，以通过 Kafka Bridge 发送和接收信息。要进行端到端追踪，您必须在 HTTP 客户端中配置追踪。

流程

为每个 KafkaMirrorMaker, KafkaMirrorMaker2, KafkaConnect, 和 KafkaBridge 资源执行这些步骤。

1. 在 spec.template 属性中，配置 tracer 服务。

   - 使用追踪环境变量作为模板配置属性。
   - 对于 OpenTelemetry，将 spec.tracing.type 属性设置为 opentelemetry。

使用 OpenTelemetry 的 Kafka Connect 的追踪配置示例

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

```
spec:
  #...
  template:
    connectContainer:
      env:
        - name: OTEL_SERVICE_NAME
          value: my-otel-service
        - name: OTEL_EXPORTER_OTLP_ENDPOINT
          value: "http://otlp-host:4317"
  tracing:
    type: opentelemetry
  #...
```

使用 OpenTelemetry 的 MirrorMaker 的追踪配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker
metadata:
  name: my-mirror-maker
spec:
  #...
  template:
    mirrorMakerContainer:
      env:
        - name: OTEL_SERVICE_NAME
          value: my-otel-service
        - name: OTEL_EXPORTER_OTLP_ENDPOINT
          value: "http://otlp-host:4317"
  tracing:
    type: opentelemetry
#...
```

使用 OpenTelemetry 的 MirrorMaker 2 的追踪配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mm2-cluster
spec:
  #...
  template:
    connectContainer:
```

```
      env:
        - name: OTEL_SERVICE_NAME
          value: my-otel-service
        - name: OTEL_EXPORTER_OTLP_ENDPOINT
          value: "http://otlp-host:4317"
    tracing:
      type: opentelemetry
  #...
```

使用 OpenTelemetry 的 Kafka Bridge 的追踪配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  #...
  template:
    bridgeContainer:
      env:
        - name: OTEL_SERVICE_NAME
          value: my-otel-service
        - name: OTEL_EXPORTER_OTLP_ENDPOINT
          value: "http://otlp-host:4317"
  tracing:
    type: opentelemetry
#...
```

2. 创建或更新资源：

```
oc apply -f <resource_configuration_file>
```

22.3.3. 初始化 Kafka 客户端的追踪

为 OpenTelemetry 初始化 tracer，然后检测您的客户端应用程序以进行分布式追踪。您可以检测 Kafka producer 和消费者客户端，以及 Kafka Streams API 应用程序。

使用一组 追踪环境变量 配置和初始化 tracer。

流程

在每个客户端应用程序中添加 tracer 的依赖项：

1. **将 Maven 依赖项添加到客户端应用程序的 pom.xml 文件中：**

OpenTelemetry 的依赖项

```
<dependency>
  <groupId>io.opentelemetry.semconv</groupId>
  <artifactId>opentelemetry-semconv</artifactId>
  <version>1.21.0-alpha</version>
</dependency>
<dependency>
  <groupId>io.opentelemetry</groupId>
  <artifactId>opentelemetry-exporter-otlp</artifactId>
  <version>1.34.1</version>
  <exclusions>
    <exclusion>
      <groupId>io.opentelemetry</groupId>
      <artifactId>opentelemetry-exporter-sender-okhttp</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>io.opentelemetry</groupId>
  <artifactId>opentelemetry-exporter-sender-grpc-managed-channel</artifactId>
  <version>1.34.1</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>io.opentelemetry</groupId>
  <artifactId>opentelemetry-sdk-extension-autoconfigure</artifactId>
  <version>1.34.1</version>
</dependency>
<dependency>
  <groupId>io.opentelemetry.instrumentation</groupId>
  <artifactId>opentelemetry-kafka-clients-2.6</artifactId>
  <version>1.32.0-alpha</version>
</dependency>
<dependency>
  <groupId>io.opentelemetry</groupId>
  <artifactId>opentelemetry-sdk</artifactId>
  <version>1.34.1</version>
</dependency>
<dependency>
  <groupId>io.opentelemetry</groupId>
  <artifactId>opentelemetry-exporter-sender-jdk</artifactId>
  <version>1.34.1-alpha</version>
  <scope>runtime</scope>
```

```
</dependency>
<dependency>
  <groupId>io.grpc</groupId>
  <artifactId>grpc-netty-shaded</artifactId>
  <version>1.61.0</version>
</dependency>
```

2. 使用追踪环境变量定义 tracer 的配置。

3. 创建一个 tracer，它使用环境变量初始化：

为 OpenTelemetry 创建 tracer

```
OpenTelemetry ot = GlobalOpenTelemetry.get();
```

4. 将 tracer 注册为全局 tracer：

```
GlobalTracer.register(tracer);
```

5. 检测您的客户端：

- 第 22.3.4 节 “用于追踪的制作者和消费者的工具”
- 第 22.3.5 节 “为追踪检测 Kafka Streams 应用程序”

22.3.4. 用于追踪的制作者和消费者的工具

检测应用程序代码，以便在 Kafka 生成者和消费者中启用追踪。使用 decorator 模式或拦截器来检测您的 Java 生成者和消费者应用程序代码以进行追踪。然后，您可以在从主题生成或检索消息时记录 trace。

OpenTelemetry 检测项目提供类来支持生成者和消费者的工具。

decorator 检测

对于 decorator 检测，请为追踪创建修改后的制作者或消费者实例。

拦截器检测

对于拦截器检测，请将追踪功能添加到消费者或生成者配置中。

先决条件

- 您已为 客户端 初始化了追踪。

  您可以通过在项目中添加追踪 JAR 作为依赖项来启用生成者和消费者应用程序的检测。

流程

在每个制作者和消费者应用的应用程序代码中执行这些步骤。使用 decorator 模式或拦截器（拦截器）检测您的客户端应用程序代码。

- 要使用 decorator 模式，请创建一个修改后的制作者或消费者实例来发送或接收消息。

  您传递了原始 KafkaProducer 或 KafkaConsumer 类。

OpenTelemetry 的 decorator 检测示例

```
// Producer instance
Producer < String, String > op = new KafkaProducer < > (
   configs,
   new StringSerializer(),
   new StringSerializer()
   );
   Producer < String, String > producer = tracing.wrap(op);
KafkaTracing tracing = KafkaTracing.create(GlobalOpenTelemetry.get());
producer.send(...);

//consumer instance
Consumer<String, String> oc = new KafkaConsumer<>(
   configs,
   new StringDeserializer(),
```

```
    new StringDeserializer()
    );
    Consumer<String, String> consumer = tracing.wrap(oc);
consumer.subscribe(Collections.singleton("mytopic"));
ConsumerRecords<Integer, String> records = consumer.poll(1000);
ConsumerRecord<Integer, String> record = ...
SpanContext spanContext = TracingKafkaUtils.extractSpanContext(record.headers(),
tracer);
```

- 要使用拦截器，请在生成者或消费者配置中设置拦截器类。

您以通常的方式使用 KafkaProducer 和 KafkaConsumer 类。TracingProducerInterceptor 和 TracingConsumerInterceptor interceptor 类负责追踪功能。

使用拦截器的制作者配置示例

```
senderProps.put(ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,
    TracingProducerInterceptor.class.getName());

KafkaProducer<Integer, String> producer = new KafkaProducer<>(senderProps);
producer.send(...);
```

使用拦截器的消费者配置示例

```
consumerProps.put(ConsumerConfig.INTERCEPTOR_CLASSES_CONFIG,
    TracingConsumerInterceptor.class.getName());

KafkaConsumer<Integer, String> consumer = new KafkaConsumer<>
(consumerProps);
consumer.subscribe(Collections.singletonList("messages"));
ConsumerRecords<Integer, String> records = consumer.poll(1000);
ConsumerRecord<Integer, String> record = ...
SpanContext spanContext = TracingKafkaUtils.extractSpanContext(record.headers(),
tracer);
```

22.3.5. 为追踪检测 Kafka Streams 应用程序

检测应用程序代码，以便在 Kafka Streams API 应用程序中启用追踪。使用 decorator 模式或拦截器来检测您的 Kafka Streams API 应用程序以进行追踪。然后，您可以在从主题生成或检索消息时记录 trace。

decorator 检测

对于 decorator 检测，请为追踪创建一个修改后的 Kafka Streams 实例。对于 OpenTelemetry，您需要创建一个自定义 TracingKafkaClientSupplier 类，以提供 Kafka Streams 的追踪工具。

拦截器检测

对于拦截器检测，在 Kafka Streams producer 和消费者配置中添加追踪功能。

先决条件

- 您已为 客户端 初始化了追踪。

  您可以通过在项目中添加追踪 JAR 作为依赖项来启用 Kafka Streams 应用程序中的检测。

- 要使用 OpenTelemetry 检测 Kafka Streams，您需要编写自定义 TracingKafkaClientSupplier。

- 自定义 TracingKafkaClientSupplier 可以扩展 Kafka 的 DefaultKafkaClientSupplier，覆盖生成者和消费者创建方法，将实例嵌套与遥测相关的代码。

  自定义 TracingKafkaClientSupplier 示例

```
private class TracingKafkaClientSupplier extends DefaultKafkaClientSupplier {
    @Override
    public Producer<byte[], byte[]> getProducer(Map<String, Object> config) {
        KafkaTelemetry telemetry = KafkaTelemetry.create(GlobalOpenTelemetry.get());
        return telemetry.wrap(super.getProducer(config));
    }

    @Override
    public Consumer<byte[], byte[]> getConsumer(Map<String, Object> config) {
        KafkaTelemetry telemetry = KafkaTelemetry.create(GlobalOpenTelemetry.get());
        return telemetry.wrap(super.getConsumer(config));
    }
```

```
    @Override
    public Consumer<byte[], byte[]> getRestoreConsumer(Map<String, Object> config)
{
        return this.getConsumer(config);
    }

    @Override
    public Consumer<byte[], byte[]> getGlobalConsumer(Map<String, Object> config) {
        return this.getConsumer(config);
    }
}
```

流程

为每个 Kafka Streams API 应用程序执行这些步骤。

- 要使用 decorator 模式，创建一个 TracingKafkaClientSupplier 供应商接口的实例，然后为 KafkaStreams 提供供应商接口。

  decorator 检测示例

```
KafkaClientSupplier supplier = new TracingKafkaClientSupplier(tracer);
KafkaStreams streams = new KafkaStreams(builder.build(), new
StreamsConfig(config), supplier);
streams.start();
```

- 要使用拦截器，请在 Kafka Streams producer 和消费者配置中设置拦截器类。

  TracingProducerInterceptor 和 TracingConsumerInterceptor interceptor 类负责追踪功能。

  使用拦截器的制作者和消费者配置示例

```
props.put(StreamsConfig.PRODUCER_PREFIX +
```

```
ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,
TracingProducerInterceptor.class.getName());
props.put(StreamsConfig.CONSUMER_PREFIX +
ConsumerConfig.INTERCEPTOR_CLASSES_CONFIG,
TracingConsumerInterceptor.class.getName());
```

22.3.6. 引入不同的 OpenTelemetry 追踪系统

您可以指定 OpenTelemetry 支持的其他追踪系统，而不是默认的 OTLP 系统。您可以通过在 Apache Kafka 的 Streams 提供的 Kafka 镜像中添加所需的工件。还必须设置任何所需的特定环境变量。然后，您可以使用 OTEL_TRACES_EXPORTER 环境变量启用新的追踪实施。

此流程演示了如何实施 Zipkin tracing。

流程

1. 将追踪工件添加到 Apache Kafka 镜像的 Streams 的 /opt/kafka/libs/ 目录中。

   您可以使用 红帽生态系统目录 上的 Kafka 容器镜像作为基础镜像来创建新的自定义镜像。

   Zipkin 的 OpenTelemetry 工件

   ```
   io.opentelemetry:opentelemetry-exporter-zipkin
   ```

2. 为新的追踪实现设置追踪导出器和端点。

   Zikpin tracer 配置示例

   ```
   apiVersion: kafka.strimzi.io/v1beta2
   kind: KafkaMirrorMaker2
   metadata:
     name: my-mm2-cluster
   ```

```
spec:
  #...
  template:
    connectContainer:
      env:
        - name: OTEL_SERVICE_NAME
          value: my-zipkin-service
        - name: OTEL_EXPORTER_ZIPKIN_ENDPOINT
          value: http://zipkin-exporter-host-name:9411/api/v2/spans (1)
        - name: OTEL_TRACES_EXPORTER
          value: zipkin (2)
  tracing:
    type: opentelemetry
#...
```

(1) 指定要连接到的 Zipkin 端点。

(2) Zipkin exporter。

22.3.7. 为 OpenTelemetry 指定自定义 span 名称

追踪 *span* 是 Jaeger 中的逻辑工作单元，包括操作名称、开始时间和持续时间。span 具有内置名称，但您可以在使用的 Kafka 客户端检测中指定自定义范围名称。

指定自定义范围名称是可选的，只有在生成者和消费者客户端检测或 Kafka Streams 检测中使用 decorator 模式时才适用。

无法通过 OpenTelemetry 直接指定自定义 span 名称。相反，您可以通过向客户端应用程序添加代码来提取额外的标签和属性来检索范围名称。

提取属性的代码示例

```
//Defines attribute extraction for a producer
private static class ProducerAttribExtractor implements AttributesExtractor < ProducerRecord
< ? , ? > , Void > {
    @Override
```

```
    public void onStart(AttributesBuilder attributes, ProducerRecord < ? , ? > producerRecord) {
        set(attributes, AttributeKey.stringKey("prod_start"), "prod1");
    }
    @Override
    public void onEnd(AttributesBuilder attributes, ProducerRecord < ? , ? > producerRecord,
@Nullable Void unused, @Nullable Throwable error) {
        set(attributes, AttributeKey.stringKey("prod_end"), "prod2");
    }
}
//Defines attribute extraction for a consumer
private static class ConsumerAttribExtractor implements AttributesExtractor <
ConsumerRecord < ? , ? > , Void > {
    @Override
    public void onStart(AttributesBuilder attributes, ConsumerRecord < ? , ? > producerRecord)
{
        set(attributes, AttributeKey.stringKey("con_start"), "con1");
    }
    @Override
    public void onEnd(AttributesBuilder attributes, ConsumerRecord < ? , ? > producerRecord,
@Nullable Void unused, @Nullable Throwable error) {
        set(attributes, AttributeKey.stringKey("con_end"), "con2");
    }
}
//Extracts the attributes
public static void main(String[] args) throws Exception {
        Map < String, Object > configs = new HashMap < >
(Collections.singletonMap(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
"localhost:9092"));
        System.setProperty("otel.traces.exporter", "jaeger");
        System.setProperty("otel.service.name", "myapp1");
        KafkaTracing tracing = KafkaTracing.newBuilder(GlobalOpenTelemetry.get())
            .addProducerAttributesExtractors(new ProducerAttribExtractor())
            .addConsumerAttributesExtractors(new ConsumerAttribExtractor())
            .build();
```

第 23 章 使用 APACHE KAFKA DRAIN CLEANER 的 STREAMS 驱除 POD

Kafka 和 ZooKeeper pod 可能会在 OpenShift 升级、维护或 pod 重新调度过程中被驱除。如果您的 Kafka 和 ZooKeeper pod 由 Apache Kafka 的 Streams 部署，您可以使用 Apache Kafka Drain Cleaner 工具的 Streams 来处理 pod 驱除。Apache Kafka Drain Cleaner 的流处理驱除，而不是 OpenShift。

通过为 Apache Kafka Drain Cleaner 部署流，您可以使用 Cluster Operator 来移动 Kafka pod 而不是 OpenShift。Cluster Operator 确保主题永远不会被复制，并且 Kafka 可以在驱除过程中保持正常运行。Cluster Operator 会等待主题同步，因为 OpenShift worker 节点会持续排空。

准入 Webhook 通知 Apache Kafka Drain Cleaner 对 Kubernetes API 的 pod 驱除请求的流。然后，Apache Kafka Drain Cleaner 的流向 pod 添加滚动更新注解，以排空。这会通知 Cluster Operator 对被驱除的 pod 执行滚动更新。

注意

如果您不将流用于 Apache Kafka Drain Cleaner，您可以添加 pod 注解来手动执行滚动更新。

Webhook 配置

Apache Kafka Drain Cleaner 部署文件的流包括 ValidatingWebhookConfiguration 资源文件。资源提供了将 webhook 注册到 Kubernetes API 的配置。

配置定义了在 pod 驱除请求时要遵循的 Kubernetes API 规则。规则指定仅截获与 pod/eviction 子资源 相关的 CREATE 操作。如果满足这些规则，API 会转发通知。

clientConfig 指向 Apache Kafka Drain Cleaner 服务的流，以及公开 Webhook 的 /drainer 端点。Webhook 使用安全 TLS 连接，这需要身份验证。caBundle 属性指定验证 HTTPS 通信的证书链。证书以 Base64 编码。

pod 驱除通知的 Webhook 配置

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
# ...
webhooks:
```

```
- name: strimzi-drain-cleaner.strimzi.io
  rules:
    - apiGroups:   [""]
      apiVersions: ["v1"]
      operations:  ["CREATE"]
      resources:   ["pods/eviction"]
      scope:       "Namespaced"
  clientConfig:
    service:
      namespace: "strimzi-drain-cleaner"
      name: "strimzi-drain-cleaner"
      path: /drainer
      port: 443
      caBundle: Cg==
  # ...
```

23.1. 下载 APACHE KAFKA DRAIN CLEANER 部署文件的流

要部署和使用 Apache Kafka Drain Cleaner 的流，您需要下载部署文件。

Apache Kafka Drain Cleaner 部署文件的流可从 Apache Kafka 软件下载页面 的流获得。

23.2. 使用安装文件为 APACHE KAFKA DRAIN CLEANER 部署流

将 Apache Kafka Drain Cleaner 的 Streams 部署到运行 Cluster Operator 和 Kafka 集群的 OpenShift 集群。

Apache Kafka Drain Cleaner 的流可在两种不同的模式下运行。默认情况下，Drain Cleaner 拒绝（块）OpenShift 驱除请求，以防止 OpenShift 驱除 pod，而是使用 Cluster Operator 来移动 pod。这个模式可以更好地与各种集群自动扩展工具兼容，不需要任何特定的 PodDisuptionBudget 配置。另外，您可以启用旧模式，它允许驱除请求，同时指示 Cluster Operator 移动 pod。要使传统模式正常工作，您必须将 PodDisruptionBudget 配置为不允许将 maxUnavailable 选项设置为 0 的任何 pod 驱除。

先决条件

- 您已下载了 Apache Kafka Drain Cleaner 部署文件的流。
- 您有一个高度可用的 Kafka 集群部署，它运行了您要更新的 OpenShift worker 节点。

- 为高可用性复制主题。

  主题配置指定至少 3 个复制因素，最小同步副本的数量为复制因素的数量减 1。

为高可用性复制 Kafka 主题

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

排除 Kafka 或 ZooKeeper

如果您不想在 Drain Cleaner 操作中包含 Kafka 或 ZooKeeper pod，或者您希望在旧模式中使用 Drain Cleaner，请在 Drain Cleaner Deployment 配置文件中更改默认环境变量：

- 将 STRIMZI_DENY_EVICTION 设置为 false 以使用依赖 PodDisruptionBudget 配置的传统模式
- 将 STRIMZI_DRAIN_KAFKA 设置为 false 以排除 Kafka pod
- 将 STRIMZI_DRAIN_ZOOKEEPER 设置为 false 以排除 ZooKeeper pod

排除 ZooKeeper pod 的配置示例

```
apiVersion: apps/v1
kind: Deployment
```

```
spec:
  # ...
  template:
    spec:
      serviceAccountName: strimzi-drain-cleaner
      containers:
        - name: strimzi-drain-cleaner
          # ...
          env:
            - name: STRIMZI_DENY_EVICTION
              value: "true"
            - name: STRIMZI_DRAIN_KAFKA
              value: "true"
            - name: STRIMZI_DRAIN_ZOOKEEPER
              value: "false"
          # ...
```

流程

1. 如果您将 **STRIMZI_DENY_EVICTION** 环境变量设置为 **false** 来激活旧模式，还必须配置 **PodDisruptionBudget** 资源。使用 **template** 设置，在 **Kafka** 资源的 **Kafka** 和 **ZooKeeper** 部分中将**maxUnavailable** 设置为 **0** (零)。

   指定 pod 中断预算

   ```
   apiVersion: kafka.strimzi.io/v1beta2
   kind: Kafka
   metadata:
     name: my-cluster
     namespace: myproject
   spec:
     kafka:
       template:
         podDisruptionBudget:
           maxUnavailable: 0

     # ...
     zookeeper:
       template:
         podDisruptionBudget:
           maxUnavailable: 0
     # ...
   ```

此设置可防止在计划中断时自动驱除 pod，保留 Apache Kafka Drain Cleaner 和 Cluster Operator 的 Streams，以便在不同的 worker 节点上推出 pod。

如果要使用 Streams for Apache Kafka Drain Cleaner 排空 ZooKeeper 节点，为 ZooKeeper 添加相同的配置。

2. 更新 Kafka 资源：

```
oc apply -f <kafka_configuration_file>
```

3. 部署 Apache Kafka Drain Cleaner 的流。

- 要在 OpenShift 上运行 Drain Cleaner，请应用 /install/drain-cleaner/openshift 目录中的资源。

```
oc apply -f ./install/drain-cleaner/openshift
```

23.3. 使用 APACHE KAFKA DRAIN CLEANER 的 STREAMS

将 Apache Kafka Drain Cleaner 的 Streams 与 Cluster Operator 搭配使用，将 Kafka 代理或 ZooKeeper pod 从正在排空的节点中移出。当您运行 Apache Kafka Drain Cleaner 的 Streams 时，它会使用滚动更新 pod 注解来注解 pod。Cluster Operator 根据注解执行滚动更新。

先决条件

- 您已为 Apache Kafka Drain Cleaner 部署了流。

使用反关联性配置时的注意事项

在 Kafka 或 ZooKeeper pod 中使用 反关联性 时，请考虑在集群中添加备用 worker 节点。包含备用节点可确保集群在节点排空或其他节点临时不可用期间重新调度 pod。当 worker 节点排空并反关联性规则限制备用节点上的 pod 重新调度时，备用节点有助于防止重启 pod 变得不可调度。这可减少排空操作失败的问题。

流程

1. 排空托管 Kafka 代理或 ZooKeeper pod 的指定 OpenShift 节点。

```
oc get nodes
oc drain <name-of-node> --delete-emptydir-data --ignore-daemonsets --timeout=6000s
--force
```

2. 检查流中的 Apache Kafka Drain Cleaner 日志中的驱除事件，以验证 pod 是否已注解用于重启。

Apache Kafka Drain Cleaner 日志的流显示 pod 的注解

```
INFO ... Received eviction webhook for Pod my-cluster-zookeeper-2 in namespace my-
project
INFO ... Pod my-cluster-zookeeper-2 in namespace my-project will be annotated for
restart
INFO ... Pod my-cluster-zookeeper-2 in namespace my-project found and annotated
for restart

INFO ... Received eviction webhook for Pod my-cluster-kafka-0 in namespace my-
project
INFO ... Pod my-cluster-kafka-0 in namespace my-project will be annotated for restart
INFO ... Pod my-cluster-kafka-0 in namespace my-project found and annotated for
restart
```

3. 检查 Cluster Operator 日志中的协调事件，以验证滚动更新。

Cluster Operator 日志显示滚动更新

```
INFO  PodOperator:68 - Reconciliation #13(timer) Kafka(my-project/my-cluster):
Rolling Pod my-cluster-zookeeper-2
INFO  PodOperator:68 - Reconciliation #13(timer) Kafka(my-project/my-cluster):
Rolling Pod my-cluster-kafka-0
INFO  AbstractOperator:500 - Reconciliation #13(timer) Kafka(my-project/my-cluster):
reconciled
```

23.4. 监视流用于 APACHE KAFKA DRAIN CLEANER 的 TLS 证书

默认情况下，Drain Cleaner 部署会监视包含其用于身份验证的 TLS 证书的 secret。Drain Cleaner 监

视是否有变化，如证书续订。如果检测到更改，它会重新启动以重新加载 TLS 证书。Drain Cleaner 安装文件默认启用此行为。但是，您可以通过在 Drain Cleaner 安装文件的部署配置 (060-Deployment.yaml) 中将 STRIMZI_CERTIFICATE_WATCH_ENABLED 环境变量设置为 false 来禁用对证书的监控。

在启用了 STRIMZI_CERTIFICATE_WATCH_ENABLED 时，您还可以使用以下环境变量来监视 TLS 证书。

表 23.1. 排空用于监视 TLS 证书的 Cleaner 环境变量

环境变量	描述	默认
STRIMZI_CERTIFICATE_WATCH_ENABLED	启用或禁用证书监视	false
STRIMZI_CERTIFICATE_WATCH_NAMESPACE	部署 Drain Cleaner 的命名空间，以及证书 secret 所在的命名空间	strimzi-drain-cleaner
STRIMZI_CERTIFICATE_WATCH_POD_NAME	Drain Cleaner pod 名称	-
STRIMZI_CERTIFICATE_WATCH_SECRET_NAME	包含 TLS 证书的 secret 名称	strimzi-drain-cleaner
STRIMZI_CERTIFICATE_WATCH_SECRET_KEYS	包含 TLS 证书的 secret 中的字段列表	tls.crt, tls.key

控制监视操作的环境变量配置示例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-drain-cleaner
  labels:
    app: strimzi-drain-cleaner
  namespace: strimzi-drain-cleaner
spec:
  # ...
    spec:
      serviceAccountName: strimzi-drain-cleaner
      containers:
        - name: strimzi-drain-cleaner
          # ...
          env:
            - name: STRIMZI_DRAIN_KAFKA
              value: "true"
            - name: STRIMZI_DRAIN_ZOOKEEPER
              value: "true"
```

```
        - name: STRIMZI_CERTIFICATE_WATCH_ENABLED
          value: "true"
        - name: STRIMZI_CERTIFICATE_WATCH_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: STRIMZI_CERTIFICATE_WATCH_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
          # ...
```

提示

使用 Downward API 机制配置 STRIMZI_CERTIFICATE_WATCH_NAMESPACE 和 STRIMZI_CERTIFICATE_WATCH_POD_NAME。

第 24 章 检索诊断和故障排除数据

report.sh 诊断工具是由红帽提供的脚本，用于为 OpenShift 上的 Apache Kafka 部署收集故障排除流的基本数据。它收集相关的日志、配置文件和其他诊断数据，以帮助识别和解决问题。运行脚本时，您可以指定额外的参数来检索特定数据。

先决条件

- Bash 4 或更新版本以运行脚本。
- OpenShift oc 命令行工具已安装并配置为连接到正在运行的集群。

这会为 oc 命令行工具建立必要的身份验证，以与集群交互并检索所需的诊断数据。

流程

1. 下载并提取工具。

   诊断工具可从 Apache Kafka 软件下载页面 的 Streams 中提供。

2. 在提取工具的目录中打开一个终端并运行报告工具：

   ```
   ./report.sh --namespace=<cluster_namespace> --cluster=<cluster_name> --out-dir=
   <local_output_directory>
   ```

   将 <cluster_namespace > 替换为 Apache Kafka 部署的 Streams 的实际 OpenShift 命名空间， & lt;cluster_name > 替换为 Kafka 集群的名称，将 &lt ;local_output_directory > 替换为您要保存生成的报告的本地目录的路径。如果您没有指定目录，则会创建一个临时目录。

   根据需要包括其他可选报告选项：

   --bridge=<string>

   指定 Kafka Bridge 集群的名称，以获取其 pod 和日志上的数据。

   --connect=<string>

指定 Kafka Connect 集群的名称，以获取其 pod 和日志上的数据。

--mm2=<string>

指定 Mirror Maker 2 集群名称，以获取其 pod 和日志的数据。

--secrets=(off|hidden|all)

指定 secret 详细程度。默认为 hidden。可用的选项如下：

- all: 报告 Secret 密钥和数据值。
- hidden ：仅报告带有密钥的 Secret。数据值（如密码）会被删除。
- off ：不会报告 Secret。

使用数据收集选项的请求示例

```
./report.sh --namespace=my-amq-streams-namespace --cluster=my-kafka-cluster --
bridge=my-bridge-component --secrets=all --out-dir=~/reports
```

注意

如果需要，使用 chmod 命令将脚本的执行权限分配给您的用户。例如：chmod +x report.sh.

执行完脚本后，输出目录包含为 Apache Kafka 部署的每个组件收集的日志、配置和其他诊断数据的文件和目录。

报告诊断工具收集的数据

如果存在，则返回以下组件中的数据：

Cluster Operator

- **部署 YAML 和日志**
- **所有相关 pod 及其日志**
- **与集群操作器相关的资源的 YAML 文件(ClusterRoles、ClusterRoleBindings)**

drain Cleaner （如果存在）

- **部署 YAML 和日志**
- **Pod 日志**

自定义资源

- **自定义资源定义(CRD) YAML**
- **所有相关自定义资源(CR)的 YAML 文件**

事件

- **与指定命名空间相关的事件**

配置

- **Kafka pod 日志和配置文件(strimzi.properties)**

- ZooKeeper pod 日志和配置文件(zookeeper.properties)
- Entity Operator (Topic Operator, User Operator) pod 日志
- Cruise Control pod 日志
- Kafka Exporter pod 日志
- 如果在选项中指定，网桥 pod 日志
- 如果在选项中指定，请连接 pod 日志
- 如果在选项中指定，MirrorMaker 2 pod 日志

secret （如果在选项中请求）

- 与指定 Kafka 集群相关的所有 secret 的 YAML 文件

第 25 章 升级 APACHE KAFKA 的流

将 Apache Kafka 安装的 Streams 升级到 2.7 版本，并受益于新功能、性能改进和增强的安全选项。在升级过程中，Kafka 也更新至最新支持的版本，为 Apache Kafka 部署引入额外的功能和程序错误修复。

使用相同的方法升级 Cluster Operator，作为部署的初始方法。例如，如果您使用 Apache Kafka 安装文件的流，请修改这些文件来执行升级。将 Cluster Operator 升级到 2.7 后，下一步是将所有 Kafka 节点升级到最新支持的 Kafka 版本。Kafka 升级由 Cluster Operator 通过 Kafka 节点的滚动更新来执行。

如果您在新版本时遇到问题，则 Apache Kafka 的 Streams 可以 降级到 之前的版本。

Apache Kafka 版本的发布流可在 Apache Kafka 软件下载页面 的 Streams 中找到。

在不停机的情况下升级

对于配置了高可用性（至少 3 个和平均分布式分区）的主题，升级过程不应给消费者和生产者造成任何停机时间。

升级会触发滚动更新，其中代理会在进程的不同阶段重启一个。在这个时间内，整个集群可用性会临时减少，这可能会在代理失败时增加消息丢失的风险。

25.1. 所需的升级序列

要在没有停机的情况下升级代理和客户端，*您必须* 按照以下顺序完成 Apache Kafka 升级流程的流：

1. 确保您的 OpenShift 集群版本被支持。

   Apache Kafka 2.7 的流需要 OpenShift 4.12 到 4.15。

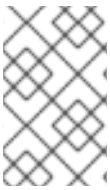
   您可以以最少的停机时间来升级 OpenShift。

2. 升级 Cluster Operator。

3. 根据集群配置升级 Kafka：

   a. 如果在 KRaft 模式中使用 Kafka，请更新 Kafka 版本和 spec.kafka.metadataVersion 来升级所有 Kafka 代理和客户端应用程序。

   b. 如果使用基于 ZooKeeper 的 Kafka，请更新 Kafka 版本和 inter.broker.protocol.version，以 升级所有 Kafka 代理和客户端应用程序。



注意

从 Apache Kafka 2.7 的 Streams 中，支持基于 KRaft 的集群之间的升级和降级。

25.2. APACHE KAFKA 升级路径流

对于 Apache Kafka，有两个升级路径可用于 Apache Kafka。

增量升级

增量升级涉及将 Apache Kafka 的流从以前的次版本升级到 2.7 版本。

多版本升级

多版本升级涉及将 Apache Kafka 的旧版本升级到 2.7 版本，并跳过一个或多个中间版本。例如，可以直接从 Apache Kafka 2.3.0 的 Streams 升级到 Apache Kafka 2.7。

25.2.1. 升级时支持 Kafka 版本

当升级 Apache Kafka 的流时，务必要确保与正在使用的 Kafka 版本兼容。

即使支持的 Kafka 版本在旧版本和新版本之间有所不同，也可以进行多版本升级。但是，如果您试图升级到不支持当前 Kafka 版本的 Apache Kafka 版本的新流，则会生成 Kafka 版本的错误。在这种情况下，您必须通过将 Kafka 自定义资源中的 spec.kafka.version 改为 Apache Kafka 版本支持的版本，将 Kafka 版本升级为 Apache Kafka 升级的 Streams 的一部分。

25.2.2. 从 1.7 之前的 Apache Kafka 版本升级

如果您要从 1.7 版本之前的版本升级到 Apache Kafka 的最新版本，请执行以下操作：

1. 按照标准序列，将 Apache Kafka 的流升级到 1.7 版本。
2. 使用 Apache Kafka 提供的 *API 转换工具* 将 Apache Kafka 自定义资源的 Streams 转换为 v1beta2。
3. 执行以下操作之一：
   - 将 Apache Kafka 的流升级到 1.8 到 0.26 之间的版本（默认禁用 ControlPlaneListener 功能门）。
   - 将 Apache Kafka 的流升级到 2.0 到 0.31 之间的版本（其中已默认启用 ControlPlaneListener 功能门），并禁用了 ControlPlaneListener 功能门。
4. 启用 ControlPlaneListener 功能门。
5. 按照标准序列，升级到 Apache Kafka 2.7 的流。

Apache Kafka 自定义资源的流使用版本 1.7 中的 v1beta2 API 版本启动。在将 CRD 和自定义资源升级到 Apache Kafka 1.8 或更新版本 前，必须转换 CRD 和自定义资源。有关使用 API 转换工具的详情，请参考 Apache Kafka 1.7 升级文档的流。

注意

作为首次升级到 1.7 版本的替代选择，您可以从 1.7 版本安装自定义资源，然后转换资源。

ControlPlaneListener 功能现在在 Apache Kafka 的 Streams 中永久启用。您必须升级到禁用 Apache Kafka 的 Streams 版本，然后使用 Cluster Operator 配置中的 STRIMZI_FEATURE_GATES 环境变量启用它。

禁用 ControlPlaneListener 功能门

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: -ControlPlaneListener
```

启用 ControlPlaneListener 功能门

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: +ControlPlaneListener
```

25.2.3. Kafka 版本和镜像映射

在升级 Kafka 时，请考虑 STRIMZI_KAFKA_IMAGES 环境变量和 Kafka.spec.kafka.version 属性的设置。

- 每个 Kafka 资源都可以配置 Kafka.spec.kafka.version，如果未指定，则默认为最新支持的 Kafka 版本(3.7.0)。
- Cluster Operator 的 STRIMZI_KAFKA_IMAGES 环境变量在 Kafka 版本和给定 Kafka 资源中请求特定的 Kafka 版本时提供映射(<kafka_version>=<image&gt ;)。例如：3.7.0=registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0。
  - 如果没有配置 Kafka.spec.kafka.image，则会使用给定版本的默认镜像。
  - 如果配置了 Kafka.spec.kafka.image，则默认镜像会被覆盖。

警告

Cluster Operator 无法验证镜像是否实际包含预期版本的 Kafka 代理。请小心操作，确保给定镜像与给定的 Kafka 版本对应。

25.3. 升级客户端的策略

升级 Kafka 客户端可确保它们从新版本的 Kafka 中引入的功能、修复和改进中受益。升级的客户端保持与其他升级的 Kafka 组件的兼容性。客户端的性能和稳定性也可能有所改进。

考虑升级 Kafka 客户端和代理的最佳方法，以确保平稳过渡。所选升级策略取决于您是否首先升级代理或客户端。从 Kafka 3.0 开始，您可以以任何顺序独立和升级代理和客户端。升级客户端或代理的决定首先取决于几个因素，如需要升级的应用程序数量以及可以容忍的停机时间。

如果您在代理前升级客户端，一些新功能可能无法正常工作，因为它们还没有被代理支持。但是，代理可以处理使用不同版本运行的生产者和消费者，并支持不同的日志消息版本。

25.4. 在短短停机时间的情况下升级 OPENSHIFT

如果要升级 OpenShift，请参阅 OpenShift 升级文档，以检查升级路径以及正确升级节点的步骤。在升级 OpenShift 前，请检查您的 Apache Kafka 版本的 Streams 版本。

在执行升级时，按照以下步骤确保 Kafka 集群的可用性：

1. 配置 pod 中断预算
2. 使用以下方法之一滚动 pod：
   a. 使用 Apache Kafka Drain Cleaner 的 Streams （推荐）
   b. 将注解应用到 pod 以手动滚动

要使 Kafka 保持正常运行，还必须复制主题以实现高可用性。这要求主题配置指定至少 3 个复制因素，最小同步副本的数量为复制因素的数量减 1。

为高可用性复制 Kafka 主题

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

在高可用性环境中，Cluster Operator 在升级过程中为主题维护最少的同步副本，以便不会停机。

25.4.1. 使用 Drain Cleaner 的滚动 pod

当使用 Apache Kafka Drain Cleaner 的 Streams 在 OpenShift 升级过程中驱除节点时，它会使用手动滚动更新注解来注解 pod，以告知 Cluster Operator 执行应该被驱除的 pod，并从要升级的 OpenShift 节点移出。

如需更多信息，请参阅 *第 23 章 使用 Apache Kafka Drain Cleaner 的 Streams 驱除 pod*。

25.4.2. 手动滚动 pod （原生 Drain Cleaner）

作为使用 Drain Cleaner 滚动 pod 的替代选择，您可以通过 Cluster Operator 触发 pod 的手动滚动更新。使用 Pod 资源时，滚动更新会使用新 pod 重启资源的 pod。要通过保留主题来复制 Drain Cleaner 的操作，对于 pod 中断预算，还必须将 maxUnavailable 值设置为 0。将 pod 中断预算减少为零可防止 OpenShift 自动驱除 pod。

指定 pod 中断预算

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    template:
      podDisruptionBudget:
        maxUnavailable: 0
# ...
```

您需要监视需要排空的 pod。然后，您可以添加 pod 注解以进行更新。

在这里，注解会更新名为 my-cluster-pool-a-1 的 Kafka pod。

在 Kafka pod 上执行手动滚动更新

```
oc annotate pod my-cluster-pool-a-1 strimzi.io/manual-rolling-update="true"
```

其他资源

- 使用 Apache Kafka Drain Cleaner 的 Streams 排空 pod
- 使用 pod 注解执行滚动更新
- PodDisruptionBudgetTemplate 模式参考
- OpenShift 文档

25.5. 升级 CLUSTER OPERATOR

使用相同的方法升级 Cluster Operator，作为部署的初始方法。

25.5.1. 使用安装文件升级 Cluster Operator

这个步骤描述了如何升级 Cluster Operator 部署以使用 Streams for Apache Kafka 2.7。

如果您使用安装 YAML 文件部署 Cluster Operator，请按照以下步骤操作。

由 Cluster Operator 管理的 Kafka 集群的可用性不受升级操作的影响。

注意

有关如何升级到该版本的信息，请参阅支持 Apache Kafka 特定版本的文档。

先决条件

- 提供了现有 Cluster Operator 部署。
- 您已下载了 Apache Kafka 2.7 的 Streams 的发行工件。

流程

1. 记录对现有 Cluster Operator 资源所做的任何配置更改（在 /install/cluster-operator 目录中）。任何更改都会被 Cluster Operator 的新版本 覆盖。
2. 更新您的自定义资源，以反映 Apache Kafka 版本 2.7 的 Streams 支持的配置选项。
3. 更新 Cluster Operator。
   a. 根据 Cluster Operator 运行的命名空间，为新的 Cluster Operator 版本修改安装文件。

在 Linux 中，使用：

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

对于 MacOS，使用：

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

b. 如果您在现有 Cluster Operator Deployment 中修改了一个或多个环境变量，请编辑 install/cluster-operator/060-Deployment-strimzi-cluster-operator.yaml 文件以使用这些环境变量。

4. 当您有更新的配置时，将其与其余安装资源一起部署：

```
oc replace -f install/cluster-operator
```

等待滚动更新完成。

5. 如果新 Operator 版本不再支持您要从中升级的 Kafka 版本，Cluster Operator 会返回错误消息，表示不支持该版本。否则，不会返回任何错误消息。

- 如果返回错误消息，请升级到新 Cluster Operator 版本支持的 Kafka 版本：
  a. 编辑 Kafka 自定义资源。
  b. 将 spec.kafka.version 属性改为支持的 Kafka 版本。
- *如果没有* 返回错误消息，请转到下一步。稍后您将升级 Kafka 版本。

6. 获取 Kafka pod 的镜像以确保升级成功：

```
oc get pods my-cluster-kafka-0 -o jsonpath='{.spec.containers[0].image}'
```

- 镜像标签显示 Apache Kafka 版本的新 Streams，后跟 Kafka 版本：

```
registry.redhat.io/amq-streams/strimzi-kafka-37-rhel9:2.7.0
```

您还可以从 Kafka 资源的状态检查升级是否已成功完成。

Cluster Operator 升级到 2.7 版本，但它管理的 Kafka 版本不会改变。

25.5.2. 使用 OperatorHub 升级 Cluster Operator

如果您从 OperatorHub 部署 Apache Kafka 的 Streams，请使用 Operator Lifecycle Manager (OLM)将 Apache Kafka operator 的 Streams 的更新频道改为 Apache Kafka 版本的新流。

根据您选择的升级策略，更新频道会启动以下类型之一：

- 启动自动升级
- 在安装开始前需要批准的手动升级

注意

如果您订阅了 ***stable*** 频道，可以在不更改频道的情况下获得自动更新。但是，不建议启用自动更新，因为缺少任何预安装升级步骤。仅在特定于版本的频道中使用自动升级。

有关使用 OperatorHub 升级 Operator 的更多信息，请参阅 升级已安装的 Operator (OpenShift 文档)。

25.5.3. 迁移到单向主题管理

当部署主题 Operator 来管理主题时，Cluster Operator 默认启用单向主题管理。如果您要从使用双向主题管理的 Apache Kafka 版本切换，在升级 Cluster Operator 后需要执行一些清理任务。如需更多信息，请参阅 第 10.9 节 “在主题 Operator 模式间切换”。

25.5.4. 升级 Cluster Operator 返回 Kafka 版本错误

如果将 Cluster Operator 升级到不支持您使用的 Kafka 当前版本的版本，则会出现一个 *不支持的 Kafka 版本* 错误。这个错误适用于所有安装方法，这意味着您必须将 Kafka 升级到支持的 Kafka 版本。将 Kafka 资源中的 spec.kafka.version 更改为支持的版本。

您可以使用 oc 检查错误信息，如包括在 Kafka 资源的 status 中的信息。

检查 Kafka 状态中的错误

```
oc get kafka <kafka_cluster_name> -n <namespace> -o jsonpath='{.status.conditions}'
```

将 <kafka_cluster_name> 替换为 Kafka 集群的名称，将 <namespace> 替换为运行 pod 的 OpenShift 命名空间。

25.5.5. 使用 OperatorHub 从 Apache Kafka 1.7 或更早版本的流升级

使用 OperatorHub 从 Apache Kafka 1.7 或更早版本的 Streams 升级所需的操作

在将 Apache Kafka Operator 的 Streams 升级到 2.7 版本前，您需要进行以下更改：

- 将自定义资源和 CRD 转换为 v1beta2
- 升级到禁用了 ControlPlaneListener 功能门的 Apache Kafka 的 Streams 版本

这些要求在 第 25.2.2 节 “从 1.7 之前的 Apache Kafka 版本升级” 中进行了描述。

如果您要从 Apache Kafka 1.7 或更早版本的 Streams 升级，请执行以下操作：

1. 升级到 Apache Kafka 1.7 的流。

2. 从 Streams for Apache Kafka 软件下载页面，从 Streams for Apache Kafka 1.8 下载 Red Hat Streams for Apache Kafka API Conversion Tool。

3. 将自定义资源和 CRD 转换为 v1beta2。

   如需更多信息，请参阅 Apache Kafka 1.7 升级文档的流。

4. 在 OperatorHub 中，删除 Apache Kafka Operator Streams 的 Streams 版本 1.7。

5. 如果存在，删除 Apache Kafka Operator 的 Streams 版本 2.7。

   如果不存在，请转到下一步。

   如果 Apache Kafka Operator 的 Streams 的批准策略被设置为 Automatic，则集群中可能已存在 Operator 版本 2.7。如果您在发行版本 *前没有* 将自定义资源和 CRD 转换为 v1beta2 API 版本，Operator 管理的自定义资源和 CRD 将使用旧的 API 版本。因此，2.7 Operator 处于 *Pending* 状态。在这种情况下，您需要删除 Apache Kafka Operator 的 Streams 版本 2.7 以及版本 1.7。

   如果同时删除这两个 Operator，协调将暂停，直到安装了新的 Operator 版本。立即遵循后续步骤，对自定义资源的任何更改都不会延迟。

6. 在 OperatorHub 中，执行以下操作之一：

   - 升级到 Apache Kafka Operator 的 Streams 版本 1.8 （默认禁用 ControlPlaneListener 功能门）。
   - 在禁用了 ControlPlaneListener 功能门的情况下，升级到 Apache Kafka Operator 的 Streams 版本 2.0 或 2.2 （其中 ControlPlaneListener 功能门被默认启用）。

7. 立即升级到 Apache Kafka Operator 的 Streams 版本 2.7。

安装的 2.7 operator 开始监控集群并执行滚动更新。您可能会注意到在此过程中，集群性能可能会临时降低。

25.6. 升级基于 KRAFT 的 KAFKA 集群和客户端应用程序

将基于 KRaft 的 Apache Kafka 集群的流升级到较新的支持的 Kafka 版本和 KRaft 元数据版本。

您还应选择升级客户端的策略。在此流程的第 6 步中升级 Kafka 客户端。

注意

有关基于 KRaft 的升级支持的最新信息，请参阅 Apache Kafka 文档。

先决条件

- Cluster Operator 已启动并在运行。
- 在升级 Apache Kafka 集群的 Streams 前，请检查 Kafka 资源的属性 *不包含* 新的 Kafka 版本不支持的配置选项。

流程

1. 更新 Kafka 集群配置：

```
oc edit kafka <kafka_configuration_file>
```

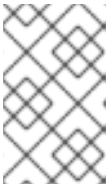
2. 如果配置，请检查当前 spec.kafka.metadataVersion 是否已设置为您要升级到的 Kafka 版本支持的版本。

例如，如果从 Kafka 版本 3.6.0 升级到 3.7.0，则当前版本为 3.6-IV2：

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

```
    replicas: 3
    metadataVersion: 3.6-IV2
    version: 3.6.0
    # ...
```

如果没有配置 metadataVersion，则 Apache Kafka 的 Streams 会在在下一步中更新到 Kafka 版本后自动更新到当前默认值。



注意

metadataVersion 的值必须是字符串，以防止它被解释为浮点数。

3.

更改 Kafka.spec.kafka.version 以指定新的 Kafka 版本；将 metadataVersion 保留为 *当前* Kafka 版本的默认值。

注意

更改 kafka.version 可确保升级集群中的所有代理，以使用新的代理二进制文件。在此过程中，一些代理使用旧的二进制文件，而其他代理已升级到新的二进制文件。将 metadataVersion 保持不变到当前设置中，可确保 Kafka 代理和控制器可以在升级过程中继续相互通信。

例如，如果从 Kafka 3.6.0 升级到 3.7.0 ：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3
    metadataVersion: 3.6-IV2 (1)
    version: 3.7.0 (2)
    # ...
```

(1) 元数据版本没有改变

(2) Kafka 版本已改为新版本。

4. 如果在 Kafka 自定义资源的 Kafka.spec.kafka.image 中定义了 Kafka 集群 的镜像，请更新该镜像以指向新的 Kafka 版本的容器镜像。

   请参阅 Kafka 版本和镜像映射

5. 保存并退出编辑器，然后等待滚动更新升级 Kafka 节点完成。

   通过观察 pod 状态转换来检查滚动更新的进度：

   ```
   oc get pods my-cluster-kafka-0 -o jsonpath='{.spec.containers[0].image}'
   ```

   滚动更新可确保每个 pod 都使用 Kafka 的新版本的代理二进制文件。

6. 根据您选择的 策略来升级客户端，升级所有客户端应用程序以使用客户端二进制文件的新版本。

   如果需要，将 Kafka Connect 和 MirrorMaker 的 version 属性设置为 Kafka 的新版本：

   a. 对于 Kafka Connect，更新 KafkaConnect.spec.version。

   b. 对于 MirrorMaker，更新 KafkaMirrorMaker.spec.version。

   c. 对于 MirrorMaker 2，更新 KafkaMirrorMaker2.spec.version。

   注意

   如果使用手动构建的自定义镜像，您必须重建这些镜像以确保它们与 Apache Kafka 基础镜像的最新流最新。例如，如果您从 基础 Kafka Connect 镜像创建了容器镜像，请更新 Dockerfile 以指向最新的基础镜像和构建配置。

7. 验证升级的客户端应用程序是否与新的 Kafka 代理正常工作。

8. 如果配置，将 Kafka 资源更新为使用新的 metadataVersion 版本。否则，请转到第 9 步。

   例如，如果升级到 Kafka 3.7.0 ：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3
    metadataVersion: 3.7-IV2
    version: 3.7.0
    # ...
```

警告

更改 metadataVersion 时请小心，因为可能无法降级。如果新 Kafka 版本的 metadataVersion 高于您要降级的 Kafka 版本，则无法降级 Kafka。但是，了解维护旧版本时对支持和兼容性的潜在影响。

9. 等待 Cluster Operator 更新集群。

   您可以从 Kafka 资源的状态检查升级是否已成功完成。

25.7. 使用 ZOOKEEPER 升级 KAFKA

如果您使用基于 ZooKeeper 的 Kafka 集群，升级需要更新 Kafka 版本和 inter-broker 协议版本。

如果要将 Kafka 集群从使用 ZooKeeper 切换，以便元数据管理在 KRaft 模式中操作，则必须独立于升级执行这些步骤。有关迁移到基于 KRaft 的集群的详情，请参考 第 8 章 *迁移到 KRaft 模式*。

25.7.1. 更新 Kafka 版本

当使用 ZooKeeper 进行集群管理时，在 Kafka 资源的配置中需要更新 Kafka 版本 (Kafka.spec.kafka.version)及其 inter-broker 协议版本(inter.broker.protocol.version)。Kafka 的每个

版本都有了一个内部代理协议的兼容版本。inter-broker 协议用于代理间通信。协议的次要版本通常会增加以匹配 Kafka 的次要版本，如上表中所示。inter-broker 协议版本在 Kafka 资源中设置 cluster wide。要更改它，您可以编辑 Kafka.spec.kafka.config 中的 inter.broker.protocol.version 属性。

下表显示了 Kafka 版本之间的区别：

表 25.1. Kafka 版本的不同

Apache Kafka 版本流	Kafka 版本	inter-broker 协议版本	日志消息格式版本	ZooKeeper 版本
2.7	3.7.0	3.7	3.7	3.8.3
2.6	3.6.0	3.6	3.6	3.8.3

- Kafka 3.7.0 支持在生产环境中使用。
- Kafka 3.6.0 仅支持升级到 Apache Kafka 2.7 的 Streams 的目的。

日志消息格式版本

当制作者发送消息到 Kafka 代理时，消息会使用特定的格式进行编码。格式可能会在 Kafka 发行版本之间改变，因此消息指定它们编码的消息格式版本。

用于设置特定消息格式版本的属性如下：

- 主题的 message.format.version 属性
- Kafka 代理的 log.message.format.version 属性

从 Kafka 3.0.0，消息格式版本值被假定为与 inter.broker.protocol.version 匹配，且不需要设置。该值反映了使用的 Kafka 版本。

当升级到 Kafka 3.0.0 或更高版本时，您可以在更新 inter.broker.protocol.version 时删除这些设置。否则，您可以根据您要升级到的 Kafka 版本设置消息格式版本。

由在 Kafka 代理中设置的 log.message.format.version 定义的 message.format.version 的默认值。您可以通过修改主题配置来手动设置主题的 message.format.version。

Kafka 版本的滚动更新更改

当 Kafka 版本被更新时，Cluster Operator 会启动对 Kafka 代理的滚动更新。进一步的滚动更新依赖于 inter.broker.protocol.version 和 log.message.format.version 的配置。

如果 Kafka.spec.kafka.config 包含...	Cluster Operator 启动...
inter.broker.protocol.version 和 **log.message.format.version**。	单个滚动更新。更新后，必须手动更新 **inter.broker.protocol.version**，后跟 **log.message.format.version**。更改每个将触发进一步的滚动更新。
inter.broker.protocol.version 或 **log.message.format.version**。	两个滚动更新。
没有 **inter.broker.protocol.version** 或 **log.message.format.version** 的配置。	两个滚动更新。

重要

从 Kafka 3.0.0，当 inter.broker.protocol.version 设置为 3.0 或更高版本时，log.message.format.version 选项会被忽略，且不需要设置。代理的 log.message.format.version 属性和主题的 message.format.version 属性已弃用，并将在以后的 Kafka 发行版本中删除。

作为 Kafka 升级的一部分，Cluster Operator 为 ZooKeeper 启动滚动更新。

- 即使 ZooKeeper 版本没有改变，也会进行单个滚动更新。
- 如果 Kafka 的新版本需要新的 ZooKeeper 版本，则会进行额外的滚动更新。

25.7.2. 升级带有旧消息格式的客户端

在 Kafka 3.0 之前，您可以使用 log.message.format.version 属性（或主题级别上的 message.format.version 属性）为代理配置特定的消息格式。这允许代理适应使用过时的消息格式的旧

Kafka 客户端。虽然 Kafka 在没有显式设置此属性的情况下支持旧的客户端，但代理需要从旧的客户端转换信息，这会显著降低性能成本。

自 0.11 版以来，Apache Kafka Java 客户端支持最新的消息格式版本。如果您的所有客户端都使用最新的消息版本，您可以在升级代理时删除 log.message.format.version 或 message.format.version 覆盖。

但是，如果您仍然有使用较旧的消息格式版本的客户端，我们建议首先升级您的客户端。从消费者开始，然后在升级代理时删除 log.message.format.version 或 message.format.version 覆盖前升级制作者。这将确保您的所有客户端都支持最新的消息格式版本，且升级过程平稳。

您可以使用此指标跟踪 Kafka 客户端名称和版本：

- kafka.server:type=socket-server-metrics,clientSoftwareName=<name>,clientSoftwareVersion=<version>,listener=<listener>,networkProcessor=<processor>

提示

以下 Kafka 代理指标帮助监控消息 down-conversion 的性能：

- kafka.network:type=RequestMetrics,name=MessageConversionsTimeMs,request={Produce|Fetch} 提供了执行消息转换的时间的指标。
- kafka.server:type=BrokerTopicMetrics,name={Produce|Fetch}MessageConversionsPerSec,topic=([-.\w]+) 在一段时间内转换的信息数量上提供指标。

25.7.3. 升级基于 ZooKeeper 的 Kafka 集群和客户端应用程序

将基于 ZooKeeper 的 Apache Kafka 集群的流升级到较新的支持的 Kafka 版本和 inter-broker 协议版本。

您还应选择升级客户端的策略。在此流程的第 6 步中升级 Kafka 客户端。

先决条件

- Cluster Operator 已启动并在运行。
- 在升级 Apache Kafka 集群的 Streams 前，请检查 Kafka 资源的属性 *不包含* 新的 Kafka 版本不支持的配置选项。

流程

1. 更新 Kafka 集群配置：

```
oc edit kafka <kafka_configuration_file>
```

2. 如果配置，请检查 inter.broker.protocol.version 和 log.message.format.version 属性是否已设置为 *当前版本*。

例如，如果从 Kafka 版本 3.6.0 升级到 3.7.0，则当前版本是 3.6。

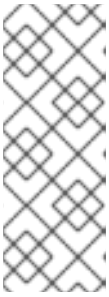
```
kind: Kafka
spec:
  # ...
  kafka:
    version: 3.6.0
    config:
      log.message.format.version: "3.6"
      inter.broker.protocol.version: "3.6"
      # ...
```

如果没有配置 log.message.format.version 和 inter.broker.protocol.version，则 Apache Kafka 的 Streams 会在在下一步中更新到 Kafka 版本后自动将这些版本更新为当前默认值。

注意

log.message.format.version 和 inter.broker.protocol.version 的值必须是字符串，以防止它们被解释为浮点号。

3. 更改 Kafka.spec.kafka.version 以指定新的 Kafka 版本；将 log.message.format.version 和 inter.broker.protocol.version 保留为 *当前* Kafka 版本的默认值。



注意

更改 kafka.version 可确保升级集群中的所有代理，以使用新的代理二进制文件。在此过程中，一些代理使用旧的二进制文件，而其他代理已升级到新的二进制文件。将 inter.broker.protocol.version 保持不变在当前设置中，可确保代理可以在升级过程中继续相互通信。

例如，如果从 Kafka 3.6.0 升级到 3.7.0 ：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # ...
  kafka:
    version: 3.7.0 (1)
    config:
      log.message.format.version: "3.6" (2)
      inter.broker.protocol.version: "3.6" (3)
      # ...
```

(1) Kafka 版本已改为新版本。

(2) 消息格式版本保持不变。

(3) inter-broker 协议版本保持不变。

警告

如果新的 Kafka 版本更改的 inter.broker.protocol.version，则无法降级 Kafka。inter-broker 协议版本决定用于代理存储的持久性元数据的模式，包括写入 __consumer_offsets 的消息。降级的集群不了解消息。

4. 如果在 Kafka 自定义资源的 Kafka.spec.kafka.image 中定义了 Kafka 集群 的镜像，请更新

该镜像以指向新的 Kafka 版本的容器镜像。

请参阅 Kafka 版本和镜像映射

5. 保存并退出编辑器，然后等待滚动更新完成。

通过观察 pod 状态转换来检查滚动更新的进度：

```
oc get pods my-cluster-kafka-0 -o jsonpath='{.spec.containers[0].image}'
```

滚动更新可确保每个 pod 都使用 Kafka 的新版本的代理二进制文件。

6. 根据您选择的策略来升级客户端，升级所有客户端应用程序以使用客户端二进制文件的新版本。

如果需要，将 Kafka Connect 和 MirrorMaker 的 version 属性设置为 Kafka 的新版本：

a. 对于 Kafka Connect，更新 KafkaConnect.spec.version。

b. 对于 MirrorMaker，更新 KafkaMirrorMaker.spec.version。

c. 对于 MirrorMaker 2，更新 KafkaMirrorMaker2.spec.version。

注意

如果使用手动构建的自定义镜像，您必须重建这些镜像以确保它们与 Apache Kafka 基础镜像的最新流最新。例如，如果您从 基础 Kafka Connect 镜像创建了容器镜像，请更新 Dockerfile 以指向最新的基础镜像和构建配置。

7. 验证升级的客户端应用程序是否与新的 Kafka 代理正常工作。

8. 如果配置，将 Kafka 资源更新为使用新的 inter.broker.protocol.version 版本。否则，请转到第 9 步。

例如，如果升级到 Kafka 3.7.0 ：

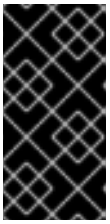
```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # ...
  kafka:
    version: 3.7.0
    config:
      log.message.format.version: "3.6"
      inter.broker.protocol.version: "3.7"
      # ...
```

9. 等待 Cluster Operator 更新集群。

10. 如果配置，将 Kafka 资源更新为使用新的 log.message.format.version 版本。否则，请转到第 10 步。

例如，如果升级到 Kafka 3.7.0 ：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # ...
  kafka:
    version: 3.7.0
    config:
      log.message.format.version: "3.7"
      inter.broker.protocol.version: "3.7"
      # ...
```



重要

从 Kafka 3.0.0，当 inter.broker.protocol.version 设置为 3.0 或更高版本时，log.message.format.version 选项会被忽略，且不需要设置。

11. 等待 Cluster Operator 更新集群。

您可以从 Kafka 资源的状态检查升级是否已成功完成。

25.8. 检查升级的状态

在执行升级（或降级）时，您可以检查 Kafka 自定义资源的状态成功完成。该状态提供有关使用 Apache Kafka 和 Kafka 版本的 Streams 的信息。

要确保在升级完成后有正确的版本，请验证 Kafka 状态中的 kafkaVersion 和 operatorLastSuccessfulVersion 值。

- operatorLastSuccessfulVersion 是 Apache Kafka operator 的 Streams 版本，最后一次执行成功协调。
- kafkaVersion 是 Kafka 集群使用的 Kafka 版本。
- kafkaMetadataVersion 是 KRaft 型 Kafka 集群使用的元数据版本

您可以使用这些值来检查 Apache Kafka 或 Kafka 的 Streams 升级已完成。

从 Kafka 状态检查升级

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
spec:
  # ...
status:
  # ...
  kafkaVersion: 3.7.0
  operatorLastSuccessfulVersion: 2.7
  kafkaMetadataVersion: 3.7
```

25.9. 在升级 APACHE KAFKA 的 STREAMS 时切换到 FIPS 模式

升级 Apache Kafka 的流，以便在启用了 FIPS 的 OpenShift 集群上以 FIPS 模式运行。until Streams for Apache Kafka 2.3，只能在启用了 FIPS 的 OpenShift 集群上运行，才能使用 FIPS_MODE 环境变量禁用 FIPS 模式。从 2.3 版本中，Apache Kafka 的 Streams 支持 FIPS 模式。如果您在启用了

FIPS 的 OpenShift 集群上运行 Streams for Apache Kafka，且将 FIPS_MODE 设置为 禁用，您可以按照以下流程启用它。

先决条件

- 启用 FIPS 的 OpenShift 集群
- 将 FIPS_MODE 环境变量设置为 disabled的现有 Cluster Operator 部署

流程

1. 将 Cluster Operator 升级到 2.3 或更新版本，但保持 FIPS_MODE 环境变量设置为 disabled。
2. 如果您最初为早于 2.3 的 Apache Kafka 版本部署了流，它可能会在其 PKCSMemcached 存储中使用旧的加密和摘要算法，这些算法在启用了 FIPS 的情况下不被支持。要使用更新的算法重新创建证书，请续订集群和客户端 CA 证书。
   a. 要续订 Cluster Operator 生成的 CA，请将 force-renew 注解添加到 CA secret 中以触发续订。
   b. 要续订您自己的 CA，将新证书添加到 CA secret，并使用更高的增量值更新 ca-cert-generation 注解，以捕获更新。
3. 如果使用 SCRAM-SHA-512 身份验证，请检查用户的密码长度。如果它们小于 32 个字符，请使用以下方法之一生成新密码：
   a. 删除用户 secret，以便 User Operator 生成一个新的密码，密码为 sufficient length。
   b. 如果您使用 KafkaUser 自定义资源的 .spec.authentication.password 属性提供密码，请更新同一密码配置中引用的 OpenShift secret 中的密码。不要忘记更新您的客户端以使用新密码。
4. 确保 CA 证书使用正确的算法，SCRAM-SHA-512 密码就足够长。然后，您可以启用 FIPS 模式。

5. **从 Cluster Operator 部署中删除 FIPS_MODE 环境变量。这会重启 Cluster Operator 并推出所有操作对象来启用 FIPS 模式。重启完成后，所有 Kafka 集群现在都会在启用了 FIPS 模式的情况下运行。**

第 26 章 降级 APACHE KAFKA 的流

如果您在升级到 Apache Kafka 的 Streams 版本时遇到问题，您可以将安装恢复到之前的版本。

如果您使用 YAML 安装文件为 Apache Kafka 安装 Streams，您可以使用上一发行版本中的 YAML 安装文件来执行降级过程。您可以通过更新 Cluster Operator 和您使用的 Kafka 版本来降级 Apache Kafka 的 Streams。Kafka 版本降级由 Cluster Operator 执行。

警告

只有在使用安装文件为 Apache Kafka 安装 Streams 时，以下降级指令才适用。如果您使用其他方法（如 OperatorHub ）安装 Apache Kafka 的 Streams，则这个方法可能不支持降级，除非在文档中另有指定。为确保成功降级过程，需要使用受支持的方法。

26.1. 将 CLUSTER OPERATOR 降级到以前的版本

如果您在 Apache Kafka 的 Streams 时遇到问题，您可以恢复安装。

此流程描述了如何将 Cluster Operator 部署降级到以前的版本。

先决条件

- 提供了现有 Cluster Operator 部署。
- 您已下载了上一版本的安装文件。

开始前

检查流的 Apache Kafka 功能门 的降级要求。如果永久启用功能门，您可能需要降级到允许您禁用它的版本，然后再降级到目标版本。

流程

1. 记录对现有 Cluster Operator 资源所做的任何配置更改（在 /install/cluster-operator 目录中）。任何更改都会被 Cluster Operator 的早期版本 覆盖。

2. 恢复您的自定义资源，以反映 Apache Kafka 的 Streams 版本支持的配置选项。

3. 更新 Cluster Operator。

   a. 根据 Cluster Operator 运行的命名空间修改之前的版本的安装文件。

      在 Linux 中，使用：

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

      对于 MacOS，使用：

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

   b. 如果您在现有 Cluster Operator Deployment 中修改了一个或多个环境变量，请编辑 install/cluster-operator/060-Deployment-strimzi-cluster-operator.yaml 文件以使用这些环境变量。

4. 当您有更新的配置时，将其与其余安装资源一起部署：

   ```
   oc replace -f install/cluster-operator
   ```

   等待滚动更新完成。

5. 获取 Kafka pod 的镜像以确保降级成功：

   ```
   oc get pod my-cluster-kafka-0 -o jsonpath='{.spec.containers[0].image}'
   ```

   镜像标签显示 Apache Kafka 版本的新 Streams，后跟 Kafka 版本。例如，<strimzi_version>-kafka-<kafka_version>。

您还可以从 Kafka 资源的状态检查降级是否已成功完成。

26.2. 降级基于 KRAFT 的 KAFKA 集群和客户端应用程序

将基于 KRaft 的 Apache Kafka 集群的流降级为早期版本。当将 KRaft-based Streams for Apache Kafka 集群降级到较低版本时，如从 3.7.0 移到 3.6.0 时，请确保 Kafka 集群使用的元数据版本是您要降级的 Kafka 版本支持的元数据版本。您从中降级的 Kafka 版本的元数据版本不能高于您要降级的版本。

注意

有关基于 KRaft 的降级支持和限制的详情，请参考 Apache Kafka 文档。

先决条件

- Cluster Operator 已启动并在运行。
- 在降级 Apache Kafka 集群的 Streams 之前，请检查以下 Kafka 资源：
  - Kafka 自定义资源不包含被降级的 Kafka 版本不支持的选项。
  - spec.kafka.metadataVersion 设置为一个版本，由要降级的 Kafka 版本支持。

流程

1. 更新 Kafka 集群配置。

```
oc edit kafka <kafka_configuration_file>
```

2. 将 metadataVersion 版本改为您要降级的 Kafka 版本支持的 metadataVersion 版本；使 Kafka.spec.kafka.version 保持不变。

例如，如果从 Kafka 3.7.0 降级到 3.6.0 ：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3
    metadataVersion: 3.6-IV2 1
    version: 3.7.0 2
    # ...
```

1 元数据版本被改为早期 Kafka 版本支持的版本。

2 Kafka 版本没有改变。

注意

metadataVersion 的值必须是字符串，以防止它被解释为浮点数。

3. 保存更改，并等待 Cluster Operator 为 Kafka 资源更新 .status.kafkaMetadataVersion。

4. 将 Kafka.spec.kafka.version 更改为之前的版本。

例如，如果从 Kafka 3.7.0 降级到 3.6.0 ：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3
    metadataVersion: 3.6-IV2 1
    version: 3.6.0 2
    # ...
```

1 Kafka 版本支持元数据版本。

2

Kafka 版本已改为新版本。

5. 如果 Kafka 版本的镜像与 Cluster Operator 的 STRIMZI_KAFKA_IMAGES 中定义的镜像不同，请更新 Kafka.spec.kafka.image。

   请参阅 第 25.2.3 节 “Kafka 版本和镜像映射”。

6. 等待 Cluster Operator 更新集群。

   您可以从 Kafka 资源的状态中检查降级是否已成功完成。

7. 降级所有客户端应用程序（使用者）以使用之前的客户端二进制文件版本。

   Kafka 集群和客户端现在使用以前的 Kafka 版本。

26.3. 使用 ZOOKEEPER 时降级 KAFKA

如果您在 ZooKeeper 模式中使用 Kafka，降级过程涉及更改 Kafka 版本以及相关的 log.message.format.version 和 inter.broker.protocol.version 属性。

26.3.1. 降级的 Kafka 版本兼容性

Kafka 降级取决于兼容的当前和目标 Kafka 版本，以及记录消息的状态。

如果以前的一个版本不支持在集群中已使用过的任何 inter.broker.protocol.version 设置，或者消息已被添加到使用较新的 log.message.format.version 的消息日志中时，则无法恢复到这个以前的 Kafka 版本。

inter.broker.protocol.version 决定用于代理存储的持久性元数据的 schema，如写入 __consumer_offsets 的消息的 schema。如果您降级到一个 Kafka 版本，它不知道之前在集群中使用的 inter.broker.protocol.version，则代理将遇到它无法理解的数据。

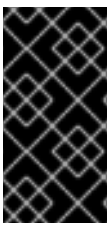
如果目标降级版本 Kafka 具有：

- 与当前版本 相同的 log.message.format.version，Cluster Operator 通过执行单个代理滚动重启来降级。
- 一个不同的 log.message.format.version，只有运行的集群始终有 log.message.format.version 设置为降级版本使用的版本时，才可以下载。这通常只有在更改 log.message.format.version 前中止升级过程时才会出现这种情况。在这种情况下，降级需要：
  - 如果两个版本的 interbroker 协议不同，则代理的两个滚动重启
  - 单个滚动重启（如果相同）

如果新版本已使用之前版本 不支持的 log.message.format.version，则无法降级，包括在使用 log.message.format.version 的默认值时。例如，此资源可以降级到 Kafka 版本 3.6.0，因为 log.message.format.version 没有改变：

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # ...
  kafka:
    version: 3.7.0
    config:
      log.message.format.version: "3.6"
      # ...
```

如果 log.message.format.version 设置为 "3.7" 或没有值，则无法降级，因此该参数会为 3.7 的 3.7.0 代理使用默认值。



重要

从 Kafka 3.0.0，当 inter.broker.protocol.version 设置为 3.0 或更高版本时，log.message.format.version 选项会被忽略，且不需要设置。

26.3.2. 降级基于 ZooKeeper 的 Kafka 集群和客户端应用程序

将 Apache Kafka 集群的基于 ZooKeeper 的流降级到早期版本。当将基于 ZooKeeper 的流降级到较

低版本时，比如从 3.7.0 移到 3.6.0，请确保 Kafka 集群使用的 inter-broker 协议版本是您要降级的 Kafka 版本支持的版本。从中降级的 Kafka 版本的 inter-broker 协议版本不能超过您要降级的版本。

注意

有关基于 ZooKeeper 的降级的支持和限制的详情，请参考 Apache Kafka 文档。

先决条件

- Cluster Operator 已启动并在运行。
- 在降级 Apache Kafka 集群的 Streams 之前，请检查以下 Kafka 资源：
  - 重要信息：Kafka 版本的兼容性。
  - Kafka 自定义资源不包含被降级的 Kafka 版本不支持的选项。
  - Kafka.spec.kafka.config 有一个 log.message.format.version 和 inter.broker.protocol.version，被 Kafka 版本降级为。

    从 Kafka 3.0.0，当 inter.broker.protocol.version 设置为 3.0 或更高版本时，log.message.format.version 选项会被忽略，且不需要设置。

流程

1. 更新 Kafka 集群配置。

```
oc edit kafka <kafka_configuration_file>
```

2. 将 inter.broker.protocol.version 版本（和 log.message.format.version ）改为您降级的 Kafka 版本支持的版本；将 Kafka.spec.kafka.version 保持不变。

   例如，如果从 Kafka 3.7.0 降级到 3.6.0 ：

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

```
metadata:
  name: my-cluster
spec:
  # ...
  kafka:
    version: 3.7.0 (1)
    config:
      inter.broker.protocol.version: "3.6" (2)
      log.message.format.version: "3.6"
      # ...
```

(1) Kafka 版本没有改变。

(2) inter-broker 协议版本被改为早期 Kafka 版本支持的版本。

注意

log.message.format.version 和 inter.broker.protocol.version 的值必须是字符串，以防止它们被解释为浮点号。

3. 保存并退出编辑器，然后等待滚动更新完成。

通过观察 pod 状态转换来检查滚动更新的进度：

```
oc get pods my-cluster-kafka-0 -o jsonpath='{.spec.containers[0].image}'
```

滚动更新可确保每个 pod 使用指定的 Kafka inter-broker 协议版本。

4. 将 Kafka.spec.kafka.version 更改为之前的版本。

例如，如果从 Kafka 3.7.0 降级到 3.6.0 ：

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

```
# ...
kafka:
  version: 3.6.0 (1)
  config:
    inter.broker.protocol.version: "3.6" (2)
    log.message.format.version: "3.6"
    # ...
```

(1) Kafka 版本已改为新版本。

(2) Kafka 版本支持 inter-broker 协议版本。

5. 如果 Kafka 版本的镜像与 Cluster Operator 的 STRIMZI_KAFKA_IMAGES 中定义的镜像不同，请更新 Kafka.spec.kafka.image。

   请参阅 第 25.2.3 节 “Kafka 版本和镜像映射”。

6. 等待 Cluster Operator 更新集群。

   您可以从 Kafka 资源的状态中检查降级是否已成功完成。

7. 降级所有客户端应用程序（使用者）以使用之前的客户端二进制文件版本。

   Kafka 集群和客户端现在使用以前的 Kafka 版本。

8. 如果您要恢复到之前早于 1.7 的 Apache Kafka 的 Streams 版本，它使用 ZooKeeper 进行主题元数据存储，请从 Kafka 集群中删除内部主题存储主题。

```
oc run kafka-admin -ti --image=registry.redhat.io/amq-streams/kafka-37-rhel9:2.7.0 --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
```

第 27 章 卸载 APACHE KAFKA 的流

您可以使用 OpenShift Container Platform Web 控制台或 CLI 从 OpenShift 4.12 上卸载 Apache Kafka 的 Streams 到 4.15。

使用与为 Apache Kafka 安装流相同的方法。

当您卸载 Apache Kafka 的流时，您需要识别专门为部署创建的资源，并从 Streams for Apache Kafka 资源引用。

这些资源包括：

- secret （自定义 CA 和证书、Kafka Connect secret 和其他 Kafka secret）
- 日志记录 ConfigMap （类型为 external）

这些是 Kafka,KafkaConnect,KafkaMirrorMaker, 或 KafkaBridge 配置引用的资源。

警告

删除 CRD 和相关的自定义资源

删除 CustomResourceDefinition 时，该类型的自定义资源也会被删除。这包括 Kafka、KafkaConnect、KafkaMirrorMaker 和 KafkaBridge 由 Apache Kafka 的 Streams 管理的资源，以及用于 Apache Kafka 的 StrimziPodSet 资源流来管理 Kafka 组件的 pod。另外，由这些自定义资源创建的任何 OpenShift 资源（如 Deployment、Pod、Service 和 ConfigMap 资源）也会被删除。删除这些资源时要小心，以避免意外数据丢失。

27.1. 使用 WEB 控制台从 OPERATORHUB 卸载 APACHE KAFKA 的流

此流程描述了如何从 OperatorHub 卸载 Apache Kafka 的流，并删除与部署相关的资源。

您可以从控制台执行步骤或使用替代 CLI 命令。

先决条件

- *使用具有 cluster-admin 或 strimzi-admin 权限的账户访问 OpenShift Container Platform Web 控制台。*
- *您已确定了要删除的资源。*

  您可以使用以下 oc CLI 命令查找资源，并在为 Apache Kafka 卸载流时验证它们是否已被删除。

  查找与 Apache Kafka 部署的流相关的资源的命令

  ```
  oc get <resource_type> --all-namespaces | grep <kafka_cluster_name>
  ```

  将 <resource_type > 替换为您要检查的资源类型，如 secret 或 configmap。

流程

1. *在 OpenShift Web 控制台中进入到 Operators > Installed Operators。*
2. *对于安装的 Apache Kafka operator，选择选项图标（三个垂直点），然后点击 Uninstall Operator。*

   Operator 从 Installed Operators 中删除。
3. *进入 Home > Projects，再选择安装用于 Apache Kafka 和 Kafka 组件的项目。*

4. 点 Inventory 下的选项删除相关资源。

   资源包括以下内容：

   - 部署
   - StatefulSets
   - Pods
   - 服务
   - ConfigMaps
   - Secrets

   提示

   使用搜索来查找以 Kafka 集群名称开头的相关资源。您还可以在 Workloads 下找到资源。

其他 CLI 命令

您可以使用 CLI 命令从 OperatorHub 卸载 Apache Kafka 的 Streams。

1. 删除 Apache Kafka 订阅的 Streams。

```
oc delete subscription amq-streams -n openshift-operators
```

2. 删除集群服务版本(CSV)。

```
oc delete csv amqstreams.<version>  -n openshift-operators
```

3. 删除相关的 CRD。

```
oc get crd -l app=strimzi -o name | xargs oc delete
```

27.2. 使用 CLI 卸载 APACHE KAFKA 的流

此流程描述了如何使用 oc 命令行工具卸载 Apache Kafka 的流，并删除与部署相关的资源。

先决条件

- 使用具有 cluster-admin 或 strimzi-admin 权限的账户访问 OpenShift 集群。
- 您已确定了要删除的资源。

  您可以使用以下 oc CLI 命令查找资源，并在为 Apache Kafka 卸载流时验证它们是否已被删除。

  查找与 Apache Kafka 部署的流相关的资源的命令

```
oc get <resource_type> --all-namespaces | grep <kafka_cluster_name>
```

  将 <resource_type > 替换为您要检查的资源类型，如 secret 或 configmap。

流程

1. 删除 Cluster Operator Deployment、相关的 CustomResourceDefinitions 和 RBAC 资源。

   指定用于部署 Cluster Operator 的安装文件。

```
oc delete -f install/cluster-operator
```

2. 删除您在先决条件中标识的资源。

```
oc delete <resource_type> <resource_name> -n <namespace>
```

将 <resource_type > 替换为您要删除的资源类型，将 <resource_name > 替换为资源名称。

删除 secret 的示例

```
oc delete secret my-cluster-clients-ca-cert -n my-project
```

第 28 章 查找 KAFKA 重启的信息

当 Cluster Operator 重启了 OpenShift 集群中的一个 Kafka pod 后，它会将 OpenShift 事件发送到 pod 的命名空间中，解释 pod 重启的原因。为了帮助了解集群行为，您可以从命令行检查重启事件。

提示

您可以使用 Prometheus 等指标集合工具导出和监控重启事件。使用带有 事件导出器 的指标工具，该导出可以以合适的格式导出输出。

28.1. 重启事件的原因

Cluster Operator 因特定原因启动重启事件。您可以通过获取重启事件的信息来检查原因。

表 28.1. 重启原因

事件	描述
CaCertHasOldGeneration	pod 仍然使用使用旧 CA 签名的服务器证书，因此需要作为证书更新的一部分重启。
CaCertRemoved	已过期的 CA 证书已被删除，pod 被重启来使用当前证书运行。
CaCertRenewed	CA 证书已更新，pod 被重启来使用更新的证书运行。
ClientCaCertKeyReplaced	用于为客户端 CA 证书签名的密钥已被替换，pod 作为 CA 续订服务器的一部分被重启。
ClusterCaCertKeyReplaced	用于为集群的 CA 证书签名的密钥已被替换，pod 作为 CA 续订服务器过程的一部分被重启。
ConfigChangeRequiresRestart	有些 Kafka 配置属性会动态更改，但有些 Kafka 配置属性需要重启代理。
FileSystemResizeNeeded	增加文件系统大小，需要重启来应用它。
KafkaCertificatesChanged	Kafka 代理使用的一个或多个 TLS 证书已更新，需要使用重启。
ManualRollingUpdate	用户标注了 pod，或 **StrimziPodSet** 设置它所属的用户，以触发重启。
PodForceRestartOnError	发生一个错误，需要 pod 重启才能重新显示。
PodHasOldRevision	在 Kafka 卷中添加或删除磁盘，需要重启来应用更改。使用 **StrimziPodSet** 资源时，如果需要重新创建 pod，则会提供相同的原因。

事件	描述
PodHasOldRevision	pod 是更新成员的 **StrimziPodSet**，因此需要重新创建 pod。当使用 **StrimziPodSet** 资源时，如果从 Kafka 卷中添加或删除磁盘，则会给出相同的原因。
PodStuck	pod 仍然处于待处理状态，且不会调度或无法调度，因此 Operator 会在最终尝试运行时重启 pod。
PodUnresponsive	Apache Kafka 的 Streams 无法连接到 pod，这可能代表一个代理无法正确启动，因此 Operator 会在尝试解决这个问题时重启它。

28.2. 重启事件过滤器

从命令行检查重启事件时，您可以指定一个 field-selector 来过滤 OpenShift 事件字段。

在使用 field-selector 过滤事件时，可以使用以下字段。

regardingObject.kind

重启的对象以及重启事件时，kind 始终为 Pod。

regarding.namespace

pod 所属的命名空间。

regardingObject.name

pod 的名称，如 strimzi-cluster-kafka-0。

regardingObject.uid

pod 的唯一 ID。

reason

pod 重启的原因，如 JbodVolumesChanged。

reportingController

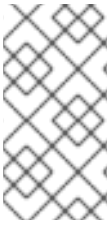
报告组件始终是 strimzi.io/cluster-operator for Streams for Apache Kafka restart 事件。

source

源是 reportingController 的旧版本。报告组件始终是 strimzi.io/cluster-operator for Streams for Apache Kafka restart 事件。

type

事件类型，可以是 Warning 或 Normal。对于 Apache Kafka 重启事件的 Streams，type 为 Normal。



注意

在 OpenShift 的旧版本中，使用前缀 的字段 可能会改用 involvedObject 前缀。reportingController 之前被称为 reportingComponent。

28.3. 检查 KAFKA 重启

使用 oc 命令列出 Cluster Operator 启动的重启事件。使用 reportingController 或 source 事件字段将 Cluster Operator 设置为报告组件来过滤 Cluster Operator 发出的重启事件。

先决条件

- *Cluster Operator 在 OpenShift 集群中运行。*

流程

1. *获取 Cluster Operator 发出的所有重启事件：*

```
oc -n kafka get events --field-selector reportingController=strimzi.io/cluster-operator
```

显示返回的事件示例

```
LAST SEEN   TYPE     REASON                 OBJECT                      MESSAGE
2m          Normal   CaCertRenewed          pod/strimzi-cluster-kafka-0   CA certificate renewed
58m         Normal   PodForceRestartOnError   pod/strimzi-cluster-kafka-1   Pod needs to be forcibly restarted due to an error
5m47s       Normal   ManualRollingUpdate      pod/strimzi-cluster-kafka-2   Pod was manually annotated to be rolled
```

您还可以指定 reason 或其他 field-selector 选项来限制返回的事件。

在这里，添加了具体原因：

```
oc -n kafka get events --field-selector reportingController=strimzi.io/cluster-operator,reason=PodForceRestartOnError
```

2. *使用 YAML 等输出格式返回有关一个或多个事件的更多详细信息。*

```
oc -n kafka get events --field-selector reportingController=strimzi.io/cluster-operator,reason=PodForceRestartOnError -o yaml
```

显示详细事件输出示例

```
apiVersion: v1
items:
- action: StrimziInitiatedPodRestart
  apiVersion: v1
  eventTime: "2022-05-13T00:22:34.168086Z"
  firstTimestamp: null
  involvedObject:
    kind: Pod
    name: strimzi-cluster-kafka-1
    namespace: kafka
  kind: Event
  lastTimestamp: null
  message: Pod needs to be forcibly restarted due to an error
  metadata:
    creationTimestamp: "2022-05-13T00:22:34Z"
    generateName: strimzi-event
    name: strimzi-eventwppk6
    namespace: kafka
    resourceVersion: "432961"
    uid: 29fcdb9e-f2cf-4c95-a165-a5efcd48edfc
  reason: PodForceRestartOnError
  reportingController: strimzi.io/cluster-operator
  reportingInstance: strimzi-cluster-operator-6458cfb4c6-6bpdp
  source: {}
  type: Normal
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

以下字段已弃用，因此这些事件不会填充它们：

- *firstTimestamp*
- *lastTimestamp*
- *source*

第 29 章 管理 APACHE KAFKA 的流

为 Apache Kafka 管理流需要执行各种任务来保持 Kafka 集群和相关资源平稳运行。使用 oc 命令检查资源的状态，为滚动更新配置维护窗口，并利用 Apache Kafka Drain Cleaner 和 Kafka Static Quota 插件等工具来有效地管理部署。

29.1. 滚动更新的维护时间窗

通过维护时间窗口，您可以调度 Kafka 和 ZooKeeper 集群的某些滚动更新，以便在方便的时间启动。

29.1.1. 维护时间窗概述

在大多数情况下，Cluster Operator 只更新您的 Kafka 或 ZooKeeper 集群，以响应对应的 Kafka 资源更改。这可让您规划何时对 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，如果您的 Kafka 和 ZooKeeper 集群的一些更新可能会在没有与 Kafka 资源对应的更改的情况下发生。例如，如果它管理的 CA （证书颁发机构）证书接近到期，Cluster Operator 将需要执行滚动重启。

虽然 pod 的滚动重启应该不会影响服务的可用性 （假设正确的代理和主题配置），但它可能会影响 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 集群的此类滚动更新，以便在方便的时间启动。如果没有为集群配置维护时间窗，则此类滚动更新可能会在不方便的时间内发生，如高负载的可预测期间。

29.1.2. 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。每个字符串都是以 UTC （协调世界时间）的 cron 表达式，它用于实际目的与 Greenwich Mean Time 相同。

以下示例配置了一个维护时间窗，它在午夜开始，并在 01:59am (UTC)结束，在 Sundays, Mondays, Tuesdays, Wednesdays, 和 Thursdays:

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

在实践中，维护窗口应当与 Kafka 资源的 Kafka.spec.clusterCa.renewalDays 和

Kafka.spec.clientsCa.renewalDays 属性一起设置，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

Apache Kafka 的流不会根据给定的窗口完全调度维护操作。相反，对于每个协调，它会检查维护窗口当前是否为 "open"。这意味着，在一个给定时间窗内开始维护操作会延迟到 Cluster Operator 协调间隔。因此，维护时间窗必须至少为长。

29.1.3. 配置维护时间窗

您可以为由支持的进程触发的滚动更新配置维护时间窗。

先决条件

- 一个 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 和 1059 之间维护 1400 到 1559，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_configuration_file>
```

其他资源

- 第 29.2.1 节 “使用 pod 管理注解执行滚动更新”
- 第 29.2.2 节 “使用 pod 注解执行滚动更新”

29.2. 使用注解启动 KAFKA 和其他操作对象的滚动更新

Apache Kafka 的流支持使用注解通过 Cluster Operator 手动触发 Kafka 和其他操作对象的滚动更新。使用注解来启动 Kafka、Kafka Connect、MirrorMaker 2 和 ZooKeeper 集群的滚动更新。

在特殊情况下，通常只需要手动对特定 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，您可以确保以下内容，而不是直接删除 pod：

- 手动删除 pod 不会与 Cluster Operator 操作冲突，如同时删除其他 pod。
- Cluster Operator 逻辑处理 Kafka 配置规格，如 in-sync 副本的数量。

29.2.1. 使用 pod 管理注解执行滚动更新

此流程描述了如何触发 Kafka、Kafka Connect、MirrorMaker 2 或 ZooKeeper 集群的滚动更新。要触发更新，您可以在管理集群中运行的 pod 的 StrimziPodSet 中添加注解。

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。您更新的组件的集群（无论是 Kafka、Kafka Connect、MirrorMaker 2 或 ZooKeeper）也必须正在运行。

流程

1. 查找控制您要手动更新的 pod 的资源名称。

   例如，如果您的 Kafka 集群名为 my-cluster，则对应的名称为 my-cluster-kafka 和 my-cluster-zookeeper。对于名为 my-connect-cluster 的 Kafka Connect 集群，对应的名称为 my-connect-cluster-connect。对于名为 my-mm2-cluster 的 MirrorMaker 2 集群，对应的名称为 my-mm2-cluster-mirrormaker2。

2. 使用 oc annotate 注解 OpenShift 中的相应资源。

Annotating a StrimziPodSet

```
oc annotate strimzipodset <cluster_name>-kafka strimzi.io/manual-rolling-update="true"

oc annotate strimzipodset <cluster_name>-zookeeper strimzi.io/manual-rolling-update="true"

oc annotate strimzipodset <cluster_name>-connect strimzi.io/manual-rolling-update="true"

oc annotate strimzipodset <cluster_name>-mirrormaker2 strimzi.io/manual-rolling-update="true"
```

3. 等待下一个协调发生（默认为两分钟）。只要协调过程检测到注解，就会触发被注解资源中的所有 pod 的滚动更新。当所有 pod 的滚动更新完成后，注解会自动从资源中删除。

29.2.2. 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、Kafka Connect、MirrorMaker 2 或 ZooKeeper 集群的滚动更新。注解多个 pod 时，会在同一协调运行中执行连续的滚动更新。

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。您更新的组件的集群（无论是 Kafka、Kafka Connect、MirrorMaker 2 或 ZooKeeper）也必须正在运行。

您可以在 Kafka 集群上执行滚动更新，无论所使用的主题复制因素是什么。但是，要让 Kafka 在更新过程中正常工作，您需要以下内容：

- 使用您要更新的节点运行高可用性 Kafka 集群部署。
- 为高可用性复制的主题。

主题配置指定至少 3 个复制因素，最小同步副本的数量为复制因素的数量减 1。

为高可用性复制 Kafka 主题

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

流程

1. 查找您要手动更新的 Pod 的名称。

   Pod 命名惯例如下：

   - <cluster_name>-kafka-<index_number> 用于 Kafka 集群
   - <cluster_name>-zookeeper-<index_number> 用于 ZooKeeper 集群
   - <cluster_name>-connect-<index_number> 用于 Kafka Connect 集群
   - <cluster_name>-mirrormaker2-<index_number> 用于 MirrorMaker 2 集群

   分配给 pod 的 <index_number> 从零开始，并以副本总数减一 结束。

2. *使用 oc annotate 注解 OpenShift 中的 Pod 资源：*

```
oc annotate pod <cluster_name>-kafka-<index_number> strimzi.io/manual-rolling-update="true"

oc annotate pod <cluster_name>-zookeeper-<index_number> strimzi.io/manual-rolling-update="true"

oc annotate pod <cluster_name>-connect-<index_number> strimzi.io/manual-rolling-update="true"

oc annotate pod <cluster_name>-mirrormaker2-<index_number> strimzi.io/manual-rolling-update="true"
```

3. *等待下一个协调发生（默认为两分钟）。当在协调过程检测到注解时，就会触发被注解的 Pod 的滚动更新。当 Pod 的滚动更新完成后，注解会自动从 Pod 中删除。*

29.3. 从持久性卷中恢复集群

如果 Kafka 集群仍然存在，您可以从持久性卷(PV)中恢复 Kafka 集群。

例如，您可能想要进行此操作，例如：

- *命名空间被意外删除*
- *整个 OpenShift 集群都会丢失，但 PV 保留在基础架构中*

29.3.1. 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，可以从命名空间删除中恢复。PersistentVolume (PV) 是位于命名空间外的存储资源。PV 使用一个 PersistentVolumeClaim (PVC)挂载到 Kafka pod 中，该 PVC 在命名空间中存在。

PV 的重新声明策略会告知集群在删除命名空间时如何操作。如果重新声明策略被设置为：

- *删除 （默认），当 PVC 在命名空间中删除时会删除 PV*

- Retain，当删除命名空间时 PV 不会删除。

如果确保在命名空间被意外删除时可以从 PV 中进行恢复，需要使用 persistentVolumeReclaimPolicy 属性在 PV 规格中将策略从 Delete 重置为 Retain：

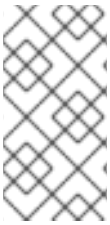
```
apiVersion: v1
kind: PersistentVolume
# ...
spec:
  # ...
  persistentVolumeReclaimPolicy: Retain
```

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，但您想要删除整个集群，则需要手动删除 PV。否则，它们不会被删除，并可能会对资源造成不必要的成本。

29.3.2. 恢复丢失 OpenShift 集群

当集群丢失时，您可以使用 disk/volumes 中的数据在基础架构中保留时恢复集群。恢复过程与删除命名空间时相同，假设可以恢复 PV，并手动创建它们。

29.3.3. 从持久性卷中恢复已删除的集群

此流程描述了如何从持久性卷(PV)中恢复已删除的集群。

在这种情况下，主题 Operator 会标识 Kafka 中存在主题，但 KafkaTopic 资源不存在。

当进入重新创建集群时，有两个选项：

1. 当您可以恢复所有 KafkaTopic 资源时，请使用 选项 1。

   因此，在启动集群前必须恢复 KafkaTopic 资源，以便主题 Operator 不会删除对应的主题。

2. 当您无法恢复所有 KafkaTopic 资源时，请使用选项 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，以便它可以从对应的主题重新创建 KafkaTopic 资源。

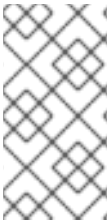
注意

如果没有部署 Topic Operator，您只需要恢复 PersistentVolumeClaim (PVC)资源。

开始前

在此过程中，PV 被挂载到正确的 PVC 中非常重要，以避免数据崩溃。为 PVC 指定 volumeName，它必须与 PV 的名称匹配。

如需更多信息，请参阅 持久性存储。



注意

该流程不包括对 KafkaUser 资源的恢复，这些资源必须手动重新创建。如果需要保留密码和证书，必须在创建 KafkaUser 资源前重新创建 secret。

流程

1. 检查集群中 PV 的信息：

```
oc get pv
```

会显示与数据相关的 PV 的信息。

显示此过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 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。

根据您是否有重新创建集群所需的所有 KafkaTopic 资源，请按照以下步骤操作。

选项 1：如果您在丢失 集群前存在所有 KafkaTopic 资源，包括内部主题，如来自 __consumer_offsets 的提交偏移：

1. 重新创建所有 KafkaTopic 资源。

   在部署集群前重新创建资源非常重要，否则主题 Operator 将删除主题。

2. 部署 Kafka 集群。

   例如：

```
oc apply -f kafka.yaml
```

选项 2：如果您在丢失集群前没有所有 KafkaTopic 资源：

1. 在部署前，部署 Kafka 集群，如第一个选项一样，但没有 Topic Operator，方法是从 Kafka 资源中删除 topicOperator 属性。

   如果您在部署中包含主题 Operator，则主题 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/amq-streams/kafka-37-rhel9:2.7.0
--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
```

   该命令必须与监听程序的类型以及用于访问 Kafka 集群的身份验证对应。

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 在这里，我们显示没有附加属性的默认配置。您可以使用 EntityTopicOperatorSpec schema reference 中所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
```

29.4. 常见问题解答

29.4.1. 与 Cluster Operator 相关的问题

29.4.1.1. 为什么需要集群管理员特权才能为 Apache Kafka 安装 Streams？

要安装 Apache Kafka 的 Streams，您需要创建以下集群范围的资源：

- 自定义资源定义(CRD)来指示 OpenShift 关于特定于 Apache Kafka Streams 的资源，如 Kafka 和 KafkaConnect
- cluster roles 和 ClusterRoleBindings

集群范围的资源（不限定于特定 OpenShift 命名空间），通常 需要集群管理员特权 来安装。

作为集群管理员，您可以检查正在安装的所有资源（在 /install/ 目录中），以确保 ClusterRole 不会授予不必要的特权。

安装后，Cluster Operator 以常规部署的形式运行，因此具有访问 Deployment 的任何标准（非 admin）OpenShift 用户都可以进行配置。集群管理员可以授予标准用户管理 Kafka 自定义资源所需的权限。

另请参阅：

- 为什么 Cluster Operator 需要创建 ClusterRoleBindings？
- 标准 OpenShift 用户能否创建 Kafka 自定义资源？

29.4.1.2. 为什么 Cluster Operator 需要创建 ClusterRoleBindings？

OpenShift 具有内置的 特权升级防止，这意味着 Cluster Operator 无法授予它本身没有权限，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 必须具有它编排的所有组件所需的权限。

Cluster Operator 需要能够授予访问权限，以便：

- 主题 Operator 可以通过在 Operator 运行的命名空间中创建 Role 和 RoleBindings 来管理 KafkaTopics
- User Operator 可以通过在 Operator 运行的命名空间中创建 Role 和 RoleBindings 来管理 KafkaUsers
- 通过创建一个 ClusterRoleBinding，通过 Streams for Apache Kafka 发现节点的故障域

使用机架感知分区分配时，代理 pod 需要能够获取它 在其上运行的节点 的信息，例如 Amazon AWS 中的可用区。Node 是一个集群范围的资源，因此只能通过 ClusterRoleBinding 而非命名空间范围的 RoleBinding 授予它。

29.4.1.3. 标准 OpenShift 用户能否创建 Kafka 自定义资源？

默认情况下，标准 OpenShift 用户没有管理 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 资源授予用户必要的特权。

如需更多信息，请参阅 第 4.6 节 “为 Apache Kafka 管理员设计流”。

29.4.1.4. 在 日志中获取锁警告失败 意味着什么？

对于每个集群，Cluster Operator 一次只执行一个操作。Cluster Operator 使用锁定来确保没有为同一集群运行两个并行操作。其他操作必须等到当前操作在锁定被释放前完成。

INFO

集群操作示例包括 集群创建、滚动更新、缩减，以及 扩展。

如果锁定的等待时间用时过长，则操作会超时，并将以下警告信息输出到日志：

```
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，这个警告信息偶尔可能会出现，而不会指示任何底层问题。下一次定期协调中获取超时的操作，以便操作可以获取锁定并再次执行。

如果此消息会定期出现，即使没有为给定集群运行其他操作，这可能表示因为错误而没有正确释放锁定。如果出现这种情况，请尝试重启 Cluster Operator。

29.4.1.5. 在使用 TLS 连接到 NodePort 时，为什么主机名验证失败？

目前，使用启用了 TLS 加密的 NodePort 的非集群访问不支持 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，但有以下例外：

```
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 设置为空字符串来实现。

当使用属性文件配置客户端时，您可以使用以下方法进行：

```
ssl.endpoint.identification.algorithm=
```

在直接在 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 30 章 在 APACHE KAFKA 的 STREAMS 中使用 METERING

您可以使用 OpenShift 上可用的 Metering 工具从不同的数据源生成 metering 报告。作为集群管理员，您可使用 Metering 来分析集群中的情况。您可以自行编写报告，也可以使用预定义的 SQL 查询来定义如何处理来自现有不同数据源的数据。使用 Prometheus 作为默认数据源，您可以生成 pod、命名空间和大多数其他 OpenShift 资源的报告。

您还可以使用 OpenShift Metering Operator 分析已安装的 Apache Kafka 组件的 Streams，以确定您是否符合红帽订阅。

要将 metering 与 Apache Kafka 的 Streams 搭配使用，您必须首先在 OpenShift Container Platform 上安装和配置 Metering Operator。

30.1. METERING 资源

Metering 具有很多资源，可用于管理 Metering 的部署与安装以及 Metering 提供的报告功能。Metering 使用以下 CRD 进行管理：

表 30.1. Metering 资源

名称	Description
MeteringConfig	为部署配置 metering 堆栈。包含用于控制 metering 堆栈各个组件的自定义和配置选项。
Reports	控制要使用的查询、查询运行时间、运行频率以及查询结果的存储位置。
ReportQueries	包含用于对 **ReportDataSources** 中包含的数据进行分析的 SQL 查询。
ReportDataSources	控制 ReportQueries 和 Reports 可用数据。支持配置 metering 中使用的不同数据库的访问权限。

30.2. APACHE KAFKA 的 STREAMS 的 METERING 标签

下表列出了 Apache Kafka 基础架构组件和集成的 Streams 的 metering 标签。

表 30.2. metering Labels

标签	可能的值
com.company	Red_Hat
rht.prod_name	Red_Hat_Application_Foundations
rht.prod_ver	2024.Q2
rht.comp	AMQ_Streams
rht.comp_ver	2.7
rht.subcomp	基础架构 cluster-operator entity-operator topic-operator user-operator zookeeper
	Application（应用程序） kafka-broker kafka-connect kafka-connect-build kafka-mirror-maker2 kafka-mirror-maker cruise-control kafka-bridge kafka-exporter drain-cleaner
rht.subcomp_t	infrastructure application

例子

- *基础架构示例（其中基础架构组件是 entity-operator）*
  -

```
com.company=Red_Hat
rht.prod_name=Red_Hat_Application_Foundations
rht.prod_ver=2024.Q2
rht.comp=AMQ_Streams
rht.comp_ver=2.7
rht.subcomp=entity-operator
rht.subcomp_t=infrastructure
```

- *应用程序示例（集成部署名称为 kafka-bridge）*

```
com.company=Red_Hat
rht.prod_name=Red_Hat_Application_Foundations
rht.prod_ver=2024.Q2
rht.comp=AMQ_Streams
rht.comp_ver=2.7
rht.subcomp=kafka-bridge
rht.subcomp_t=application
```

附录 A. 使用您的订阅

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

访问您的帐户

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

激活订阅

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

下载 Zip 和 Tar 文件

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

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

4. *单击组件的 Download 链接。*

使用 DNF 安装软件包

要安装软件包以及所有软件包的依赖软件包，请使用：

```
dnf install <package_name>
```

要从本地目录中安装之前下载的软件包，请使用：

```
dnf install <path_to_download_package>
```

更新于 2024-06-26