Kafka 文档（翻译版本）

Strimzi 简化了在 Kubernetes 集群中运行 Apache Kafka 的过程。

本指南提供了有关配置 Kafka 组件和使用 Strimzi Operators 的说明。这些过程与您可能想要如何修改部署并引入其他功能（例如 Cruise Control 或分布式跟踪）有关。

您可以使用 Strimzi 自定义资源 配置部署。在 自定义资源的 API 参考 描述您可以在配置中使用的属性。

注意	想要开始使用Strimzi？有关逐步部署说明，请参阅《*部署 Strimzi 指南*》。

1.1。Kafka功能

Kafka的基础数据流处理功能和组件体系结构可以提供：

- 微服务和其他应用程序以极高的吞吐量和低延迟共享数据
- 消息订购保证
- 从数据存储中倒回/重放消息以重建应用程序状态
- 使用键值日志时，压缩邮件以删除旧记录
- 集群配置中的水平可伸缩性
- 复制数据以控制容错
- 保留大量数据以立即访问

1.2。Kafka用例

Kafka的功能使其适用于：

- 事件驱动架构
- 事件源以捕获应用程序状态的更改作为事件日志
- 消息代理
- 网站活动跟踪
- 通过指标进行运营监控
- 日志收集与汇总
- 提交分布式系统的日志
- 流处理，以便应用程序可以实时响应数据

1.3。Strimzi如何支持Kafka

Strimzi提供了容器镜像和 Operator ，用于在Kubernetes上运行Kafka。Strimzi Operator 是Strimzi运行的基础。Strimzi提供的 Operator 是专门构建的，具有专业的操作知识，可以有效地管理Kafka。

Operator 简化了以下过程：

- 部署和运行Kafka集群
- 部署和运行Kafka组件
- 配置对Kafka的访问
- 确保对Kafka的访问
- 升级Kafka
- 管理broker
- 创建和管理 Topic
- 创建和管理用户

1.4。Strimzi operator

Strimzi使用 *Operator* 支持Kafka 来部署和管理Kafka到Kubernetes的组件和依赖项。

Operator 是打包、部署和管理Kubernetes应用程序的一种方法。Strimzi Operators扩展了Kubernetes功能，自动执行与Kafka部署相关的常见和复杂任务。通过在代码中实现对Kafka操作的了解，简化了Kafka管理任务，并减少了人工干预。

Operators

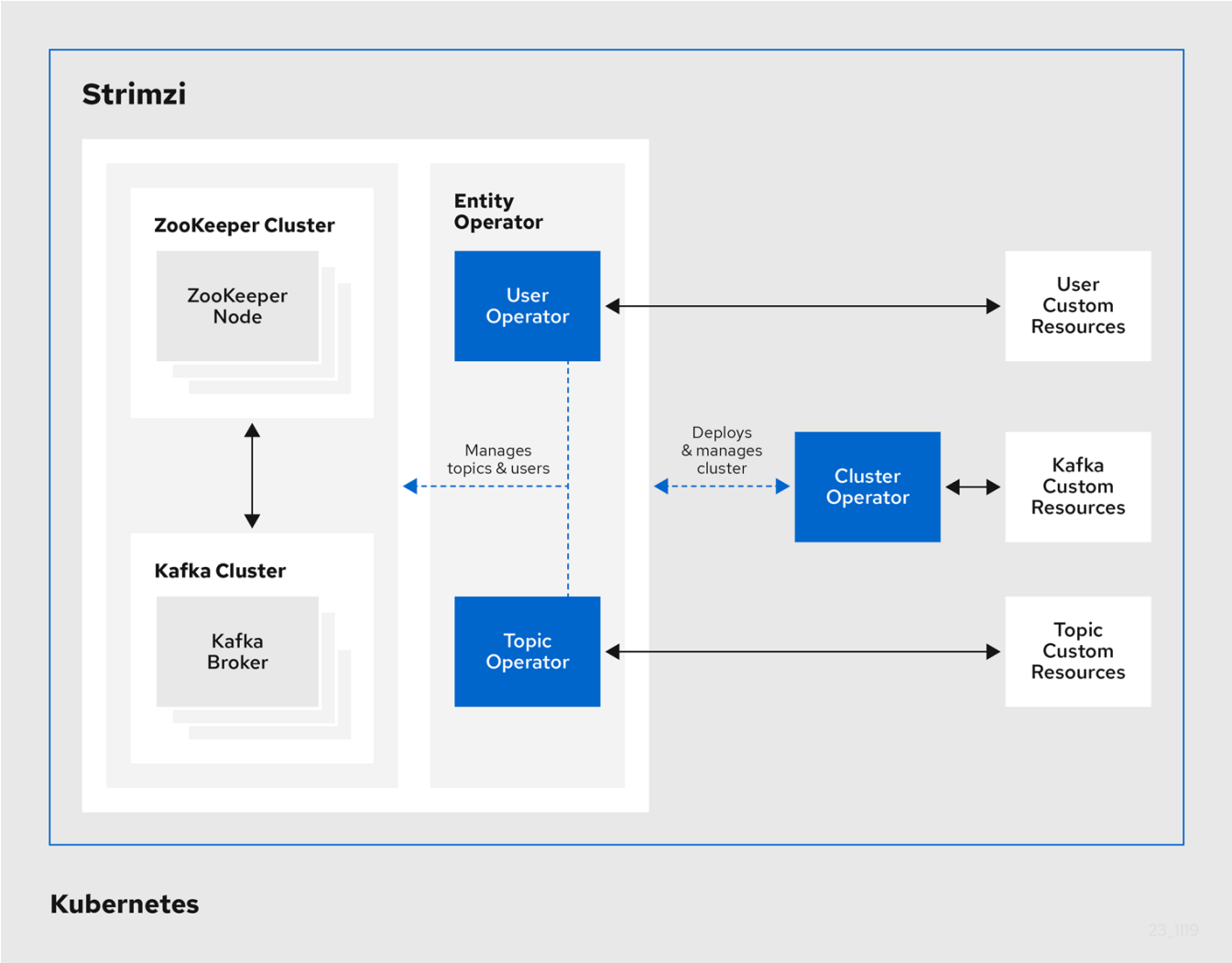
Strimzi为 Operator 提供了管理在Kubernetes集群中运行的Kafka集群的功能。

群集 Operator 部署和管理Apache Kafka群集，Kafka Connect，Kafka MirrorMaker，Kafka Bridge，Kafka Exporter和实体 Operator

实体 Operator 包括 Topic Operator 和user Operator Topic Operator 管理Kafka Topic 用户 Operator 管理Kafka用户

群集 Operator 可以与Kafka群集同时将 Topic Operator 和user Operator 部署为**实体 Operator** 配置的一部分。

Strimzi架构下的 Operator



1.4.1。集群 Operator

Strimzi使用群集 Operator 为以下方面部署和管理群集：

- Kafka（包括ZooKeeper，实体 Operator ，Kafka导出程序和巡航控制）
- Kafka Connect
- Kafka MirrorMaker
- Kafka Bridge

自定义资源用于部署群集。

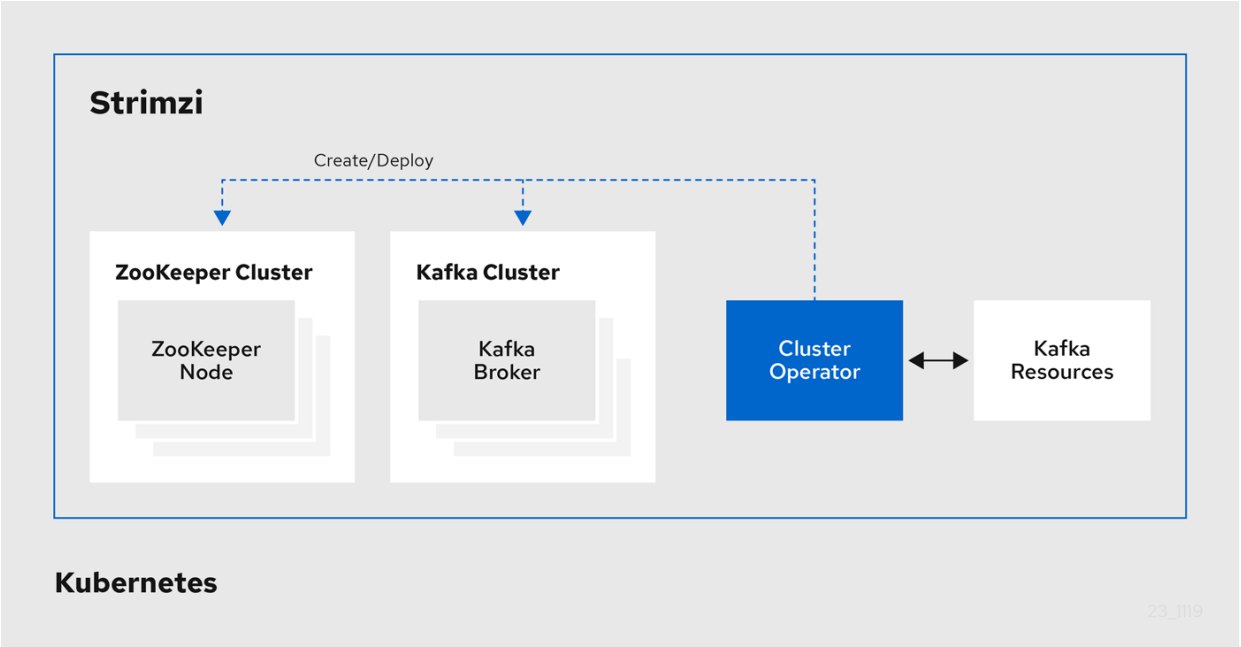
例如，要部署Kafka集群：

- 在Kubernetes集群中创建了具有集群配置的Kafka资源。
- 群集 Operator 根据Kafka资源中声明的内容部署相应的Kafka群集。

集群Operator还可以部署（通过配置Kafka资源）：

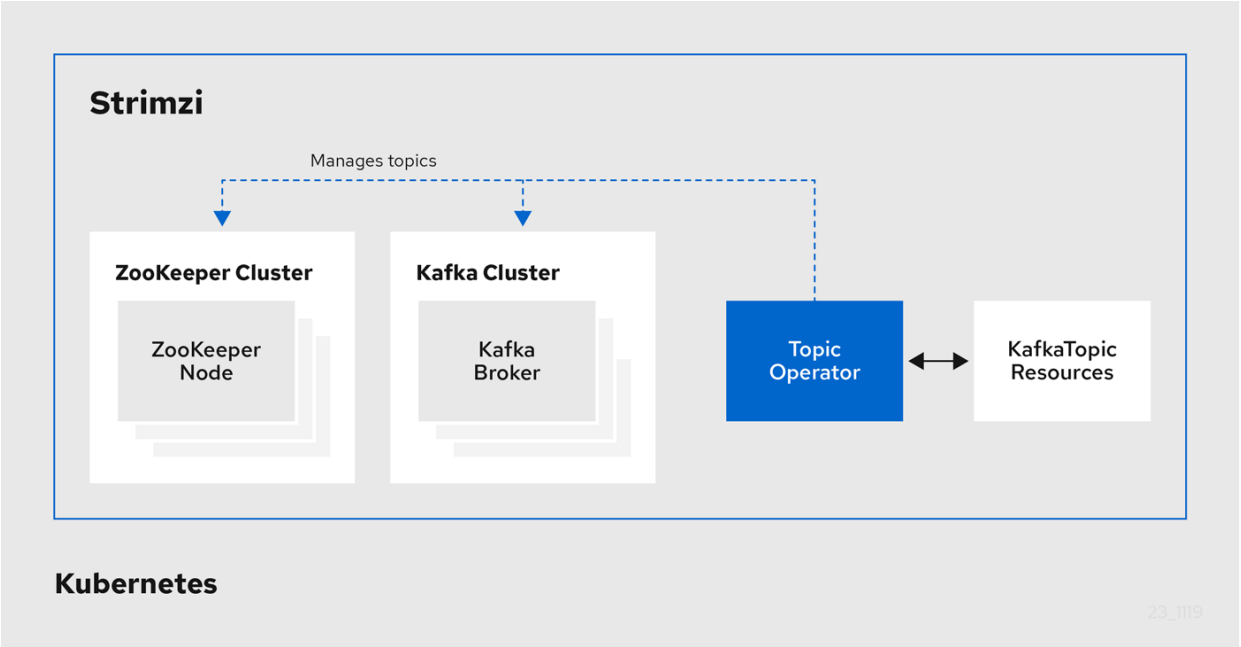
- Topic operator 通过自定义资源提供 operator 样式的 Topic 管理 `KafkaTopic`
- user Operator 通过自定义资源提供 Operator 风格的用户管理 `KafkaUser`

部署时，实体 operator 中的 Topic operator 和user operator 起作用。
集群 Operator 的示例架构



1.4.2。 Topic operator

Topic operator 提供了一种通过Kubernetes资源在Kafka集群中管理 Topic 的方法。
Topic operator 的示例架构



Topic operator 的作用是保持一组描述Kafka Topic 的Kubernetes资源与相应的Kafka Topic 同步。

具体来说，如果 `KafkaTopic`

- 创建后， Topic operator 将创建 Topic
- 删除后， Topic operator 删除 Topic
- 更改后， Topic operator 将更新 Topic

另一方面，如果 Topic 是：

- 在Kafka集群中创建的， Operator 会创建一个 `KafkaTopic`
- 从Kafka群集中删除后， Operator 将删除 `KafkaTopic`
- 在Kafka集群中更改后， Operator 会更新 `KafkaTopic`

这使您可以声明KafkaTopic作为应用程序部署的一部分，并且Topic Operator将负责为您创建 Topic 。您的应用程序只需要处理必要 Topic 的产生或使用。

如果将 Topic 重新配置或重新分配给其他Kafka节点，则该 Topic 将始终是最新的。

1.4.3。用户 Operator

用户 Operator 通过观察描述Kafka用户的资源，并确保在Kafka集群中正确配置了这些资源，来管理Kafka集群的Kafka用户。 `KafkaUser`

例如，如果`KafkaUser` ：

- 创建后，用户 Operator 将创建其描述的用户
- 删除后，用户 Operator 将删除其描述的用户
- 更改后，用户 Operator 将更新它描述的用户

与 Topic operator 不同，用户 operator 不会将来自Kafka集群的任何更改与Kubernetes资源同步。可以通过应用程序直接在Kafka中创建Kafka Topic ，但是不希望与User Operator并行地在Kafka集群中直接管理用户。

用户 Operator 允许您将资源声明为应用程序部署的一部分。您可以为用户指定身份验证和授权机制。您还可以配置用于控制Kafka资源使用情况的*用户配额*，以确保（例如）用户不垄断对代理的访问。

创建用户时，将在中创建用户凭据`Secret`。您的应用程序需要使用用户及其凭据进行身份验证，并生成或使用消息。

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

1.5。Strimzi定制资源

使用Strimzi将Kafka组件部署到Kubernetes集群是可以通过应用自定义资源进行高度配置的。自定义资源被创建为自定义资源定义（CRD）添加的API实例，以扩展Kubernetes资源。

CRD用作描述Kubernetes集群中自定义资源的配置指令，并随Strimzi一起提供给部署中使用的每个Kafka组件以及 user 和 Topic 。CRD和自定义资源定义为YAML文件。Strimzi发行版随附了示例YAML文件。

CRD还允许Strimzi资源受益于本地Kubernetes功能，例如CLI可访问性和配置验证。
额外资源

- 使用CustomResourceDefinitions扩展Kubernetes API

1.5.1。Strimzi自定义资源示例

CRD需要在集群中一次性安装，以定义用于实例化和管理Strimzi特定资源的架构。

通过安装CRD将新的自定义资源类型添加到群集后，您可以根据其规范创建资源实例。

根据群集设置，安装通常需要群集管理员权限。

注意	Strimzi管理员只能访问管理自定义资源的权限。有关更多信息，请参阅《*部署Strimzi*》指南中的“ 指定Strimzi管理员 ” 。

CRD 在Kubernetes集群中定义了新的资源，例如。 `kind` `kind:Kafka`

Kubernetes API服务器允许基于来创建自定义资源，并从CRD了解到将自定义资源添加到Kubernetes集群后如何验证和存储自定义资源。

警告	删除CRD时，该类型的自定义资源也将被删除。此外，由自定义资源创建的资源（例如pod和statefulsets）也将被删除。

每个特定于Strimzi的自定义资源都符合CRD为该资源的定义的架构。Strimzi组件的自定义资源具有通用的配置属性，这些属性在下定义。

要了解CRD和自定义资源之间的关系，我们来看一个Kafka Topic 的CRD示例。
Kafka Topic CRD

```
apiVersion: kafka.strimzi.io/v1beta1
kind: CustomResourceDefinition
metadata: (1)
  name: kafkatopics.kafka.strimzi.io
  labels:
    app: strimzi
spec: (2)
  group: kafka.strimzi.io
  versions:
    v1beta1
  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. Topic CRD的metadata，其name和label标识CRD。
2. CRD的规范，包括group（domain）名称，复数名称和受支持的架构版本，它们在URL中用于访问Topic的API。 其他名称用于在CLI中标识实例资源。 例如，`kubectl get kafkatopic my-topic` 或 `kubectl get kafkatopics`.
3. 简称可以在CLI命令中使用。例如：`kubectl get kt`可以用作 `kubectl get kafkatopic`的缩写
4. 在自定义资源上使用命令时显示的信息。
5. 资源的架构参考中描述的CRD的当前状态。
6. openAPIV3Schema验证为创建 Topic 自定义资源提供验证。例如，一个 Topic 至少需要一个分区和一个副本。

注意	您可以识别Strimzi安装文件随附的CRD YAML文件，因为文件名包含一个索引号，后跟'Crd'。

这是自定义资源的相应示例。Kafka Topic 自定义资源 `KafkaTopic`

```
apiVersion: kafka.strimzi.io/v1beta1
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标识自定义资源为其实例的CRD。
2. 仅适用于KafkaTopic和KafkaUser资源的标签，用于定义topic或user所属的Kafka群集的名称（与Kafka资源的名称相同）。
3. 该规范显示了该Topic的分区和副本的数量，以及该Topic本身的配置参数。 在此示例中，指定了消息保留在Topic中的保留时间以及日志的段文件大小；
4. 资源的状态条件。在条件改变的。 `KafkaTopic` `type` `Ready` `lastTransitionTime`

可以通过平台CLI将自定义资源应用于群集。创建自定义资源时，它使用与Kubernetes API的内置资源相同的验证。

创建自定义资源后，将通知 Topic operator ，并在Strimzi中创建相应的Kafka Topic 。 `KafkaTopic`

1.6。文件约定

替代品

在本文档中，可替换文本的样式为，用斜体，大写和连字符表示。 `monospace`

例如，在以下代码中，您将要替换为名称空间的名称： *`MY-NAMESPACE`*

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

2. 部署配置

本章介绍如何配置支持的部署的不同方面：

- Kafka集群
- Kafka Connect集群
- 具有*Source2Image*支持的Kafka Connect集群
- KafkaMirrorMaker
- Kafka桥
- 巡航控制

2.1。Kafka集群配置

模式参考中描述了资源的完整模式。应用于所需资源的所有标签也将应用于构成Kafka集群的Kubernetes资源。这提供了一种方便的机制，可以根据需要标记资源。 `Kafka` `Kafka` `Kafka`

2.1.1。示例Kafka YAML配置

为了帮助您了解可用于Kafka部署的配置选项，请参阅此处提供的示例YAML文件。

该示例仅显示一些可能的配置选项，但是特别重要的选项包括：

- 资源请求（CPU /内存）
- 用于最大和最小内存分配的JVM选项
- 侦听器（和身份验证）
- 认证方式
- 存储
- 机架意识
- 指标

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3 (1)
    version: latest (2)
    resources: (3)
      requests:
        memory: 64Gi
        cpu: "8"
      limits: (4)
        memory: 64Gi
        cpu: "12"
    jvmOptions: (5)
      -Xms: 8192m
      -Xmx: 8192m
    listeners: (6)
      tls:
        authentication:(7)
          type: tls
      external: (8)
        type: route
        authentication:
          type: tls
        configuration:
          brokerCertChainAndKey: (9)
            secretName: my-secret
            certificate: my-certificate.crt
            key: my-key.key
    authorization: (10)
      type: simple
    config: (11)
      auto.create.topics.enable: "false"
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
      ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" (12)
      ssl.enabled.protocols: "TLSv1.2"
      ssl.protocol: "TLSv1.2"
```

```
    storage: (13)
      type: persistent-claim (14)
      size: 10000Gi (15)
    rack: (16)
      topologyKey: topology.kubernetes.io/zone
    metrics: (17)
      lowercaseOutputName: true
      rules: (18)
      # Special cases and very specific rules
      - pattern : kafka.server<type=(.+), name=(.+), clientId=(.+), topic=(.+), partition=(.*)><>Value
        name: kafka_server_$1_$2
        type: GAUGE
        labels:
          clientId: "$3"
          topic: "$4"
          partition: "$5"
        # ...
  zookeeper: (19)
    replicas: 3
    resources:
      requests:
        memory: 8Gi
        cpu: "2"
      limits:
        memory: 8Gi
        cpu: "2"
    jvmOptions:
      -Xms: 4096m
      -Xmx: 4096m
    storage:
      type: persistent-claim
      size: 1000Gi
    metrics:
      # ...
  entityOperator: (20)
    topicOperator:
      resources:
        requests:
          memory: 512Mi
          cpu: "1"
        limits:
          memory: 512Mi
          cpu: "1"
    userOperator:
      resources:
        requests:
          memory: 512Mi
          cpu: "1"
        limits:
          memory: 512Mi
          cpu: "1"
  kafkaExporter: (21)
    # ...
  cruiseControl: (22)
    # ...
```

1. 副本指定代理节点的数量。
2. Kafka版本，可以按照升级步骤进行更改。
3. 资源请求指定要为给定容器保留的资源。
4. 资源限制指定了容器可以消耗的最大资源。
5. JVM选项可以指定 JVM 的最小（）和最大（）内存分配。 -Xms-Xmx
6. 侦听器配置客户端如何通过引导程序地址连接到Kafka群集。侦听器配置为（不加密）或。 plain tls external
7. 可以为每个侦听器配置侦听器身份验证机制，并将其指定为相互TLS或SCRAM-SHA。
8. 外部侦听配置指定Kafka簇是如何Kubernetes外部露出，如通过一个，或。 route loadbalancer nodeport
9. 由外部证书颁发机构管理的Kafka侦听器证书的可选配置。该属性指定一个，其中包含服务器证书和私钥。还可以为TLS侦听器配置Kafka侦听器证书。 brokerCertChainAndKey Secret
10. 授权使用Kafka插件在Kafka代理上启用授权。 simple AclAuthorizer
11. Config指定代理配置。可以提供标准的Apache Kafka配置，限于不受Strimzi直接管理的那些属性。
12. 外部侦听器的SSL属性可与TLS版本的特定*密码套件*一起运行 。
13. 存储被配置为，或。 ephemeral persistent-claim jbod
14. 永久卷的存储大小可能会增加，并且其他卷可能会添加到JBOD存储中。
15. 永久存储具有其他配置选项，例如存储和用于动态卷配置。 id class
16. 机架感知配置为将副本分布在不同的机架上。一键必须群集节点的标签匹配。 topology
17. 用于Prometheus的 Kafka 指标配置。
18. Kafka规则通过JMX Exporter将指标导出到Grafana仪表板。Strimzi提供的一组规则可以复制到您的Kafka资源配置中。
19. ZooKeeper特定的配置，其中包含类似于Kafka配置的属性。
20. 实体 Operator 配置，它指定 Topic Operator 和用户 Operator 的配置。
21. Kafka Exporter配置，用于将数据公开为Prometheus指标。
22. 巡航控制配置，用于重新平衡Kafka集群。

2.1.2。数据存储注意事项

高效的数据存储基础架构对于Strimzi的最佳性能至关重要。

需要块存储。文件存储（例如NFS）不适用于Kafka。

对于块存储，可以选择例如：

- 基于云的块存储解决方案，例如Amazon Elastic Block Store（EBS）
- 本地永久卷
- 可通过*光纤通道*或*iSCSI*等协议访问的存储区域网络（SAN）卷

注意	Strimzi不需要Kubernetes原始块卷。

文件系统

建议您将存储系统配置为使用*XFS*文件系统。Strimzi也与*ext4*文件系统兼容，但这可能需要附加配置才能获得最佳效果。

Apache Kafka和ZooKeeper存储

将单独的磁盘用于Apache Kafka和ZooKeeper。

支持三种类型的数据存储：

- 临时的（仅推荐用于开发）
- 持久的
- JBOD（仅一堆磁盘，仅适用于Kafka）

有关更多信息，请参见Kafka和ZooKeeper存储。

固态驱动器（SSD）尽管不是必需的，但可以在大型集群中提高Kafka的性能，在大型集群中，数据是异步发送到多个 Topic 或从多个 Topic 接收的。SSD对ZooKeeper尤其有效，它需要快速，低延迟的数据访问。

注意	您无需置备复制的存储，因为Kafka和ZooKeeper都具有内置的数据复制。

2.1.3。Kafka和ZooKeeper存储类型

作为有状态应用程序，Kafka和ZooKeeper需要在磁盘上存储数据。Strimzi支持三种存储类型的数据：

- 短暂的
- 持久的
- JBOD存储

注意	JBOD存储仅支持Kafka，不支持ZooKeeper。

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

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

存储类型在字段中配置。 `type`

警告	部署Kafka集群后，无法更改存储类型。

额外资源

- 有关临时存储的更多信息，请参阅临时存储架构参考。
- 有关持久性存储的更多信息，请参阅持久性存储架构参考。
- 有关JBOD存储的更多信息，请参见JBOD模式参考。
- 有关架构的更多信息，请参见架构参考。 `Kafka` `Kafka`

临时存储

临时存储使用卷来存储数据。要使用临时存储，该字段应设置为。 `emptyDir` `type` `ephemeral`

重要	`emptyDir` 卷不是永久性的，重新启动Pod时，存储在其中的数据将丢失。启动新的Pod之后，它必须从群集的其他节点恢复所有数据。临时存储不适用于单节点ZooKeeper群集以及复制因子为1的Kafka Topic ，因为它会导致数据丢失。

临时存储的示例

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    storage:
      type: ephemeral
    # ...
  zookeeper:
    # ...
    storage:
      type: ephemeral
    # ...
```

日志目录

临时卷将由Kafka代理用作安装在以下路径中的日志目录：

/ var / lib / kafka / data / kafka-log_idx_Where idx是Kafka经纪人pod索引。例如/ var / lib / kafka / data / kafka-log0。

永久储存

持久存储使用“ 持久卷声明”来设置持久卷以存储数据。持久卷声明可用于提供许多不同类型的卷，具体取决于将提供该卷的存储类。可用于永久卷声明的数据类型包括许多类型的SAN存储以及本地永久卷。

要使用永久性存储，必须将设置为。永久性存储支持其他配置选项： `type` `persistent-claim`

id（可选）存储标识号。对于JBOD存储声明中定义的存储卷，此选项是必需的。默认值为0.size（必需）定义持久卷声明的大小，例如“ 1000Gi”。class（可选）用于动态卷供应的Kubernetes存储类。选择器（可选）允许选择特定的持久卷以用。它包含key：value对，代表用于选择此类卷的标签。
deleteClaim（可选）布尔值，指定在取消部署集群时是否必须删除持久卷声明。默认为false。

警告	仅在支持调整持久卷大小的Kubernetes版本中才支持增加现有Strimzi群集中的持久卷的大小。要调整大小的持久卷必须使用支持卷扩展的存储类。对于不支持卷扩展的其他版本的Kubernetes和存储类，必须在部署集群之前确定所需的存储大小。不可能减小现有持久卷的大小。

具有1000Gi的持久性存储配置的示例片段 `size`

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

以下示例演示了存储类的用法。
具有特定存储类的持久性存储配置的示例片段

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

最后，可以使用来选择特定的标记持久卷以提供所需的功能（例如SSD）。使用选择器的持久性存储配置的示例片段 `selector`

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

存储类覆盖

您可以为一个或多个Kafka代理指定不同的存储类，而不是使用默认存储类。例如，如果存储类别仅限于不同的可用性区域或数据中心，则此功能很有用。您可以将字段用于此目的。 `overrides`

在此示例中，默认存储类名为：使用存储类替代的示例Strimzi集群 `my-storage-class`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  labels:
    app: my-cluster
  name: my-cluster
  namespace: myproject
spec:
  # ...
  kafka:
    replicas: 3
    storage:
      deleteClaim: true
      size: 100Gi
      type: persistent-claim
      class: my-storage-class
      overrides:
        - broker: 0
          class: my-storage-class-zone-1a
        - broker: 1
          class: my-storage-class-zone-1b
        - broker: 2
          class: my-storage-class-zone-1c
  # ...
```

由于配置了属性，代理卷使用以下存储类： `overrides`

- 代理0的永久卷将使用。 `my-storage-class-zone-1a`
- 代理1的持久卷将使用。 `my-storage-class-zone-1b`
- 代理2的持久卷将使用。 `my-storage-class-zone-1c`

该属性当前仅用于覆盖存储类配置。当前不支持覆盖其他存储配置字段。当前不支持存储配置中的其他字段。 `overrides`

持久卷声明命名

使用持久性存储时，它将使用以下名称创建持久性容量声明：

data-cluster-name-kafka-idxPersistent Volume Claim，用于存储Kafka代理pod idx的数据。data-cluster-name-zookeeper-idxPersistentVolume Claim，用于存储ZooKeeper节点pod idx的数据。

日志目录

Kafka代理会将永久卷用作装入以下路径的日志目录：

/ var / lib / kafka / data / kafka-log_idx_Where idx是Kafka经纪人pod索引。例如/ var / lib / kafka / data / kafka-log0。

调整持久卷的大小

您可以通过增加现有Strimzi群集使用的永久卷的大小来配置增加的存储容量。在JBOD存储配置中使用单个永久卷或多个永久卷的群集中，支持调整永久卷的大小。

注意	您可以增加但不能减少永久卷的大小。Kubernetes当前不支持减小持久卷的大小。

先决条件

- 支持卷大小调整的Kubernetes集群
- 群集 Operator 正在运行。
- 使用持久卷的Kafka集群，持久卷使用支持卷扩展的存储类创建。

程序

1. 在资源中，增加分配给Kafka群集，ZooKeeper群集或两者的持久卷的大小。 `Kafka`

   - 要增加分配给Kafka群集的卷大小，请编辑属性。 `spec.kafka.storage`
   - 要增加分配给ZooKeeper群集的卷大小，请编辑该属性。 `spec.zookeeper.storage`

   例如，要将卷大小从增大到： `1000Gi` `2000Gi`

```
apiVersion: kafka.strimzi.io/v1beta1
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。

   用途： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

   Kubernetes响应集群 Operator 的请求，增加了选定持久卷的容量。调整大小完成后，群集 Operator 将重新启动所有使用调整后大小的持久卷的容器。这会自动发生。

额外资源

有关在Kubernetes中调整持久卷大小的更多信息，请参阅使用Kubernetes调整持久卷大小。

JBOD存储概述

您可以将Strimzi配置为使用JBOD，JBOD是多个磁盘或卷的数据存储配置。JBOD是为Kafka经纪人提供更多数据存储的一种方法。它还可以提高性能。

甲JBOD配置由一个或多个卷，其中的每一个可以是所描述的临时或持续的。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，在配置永久性存储卷后，您将无法更改其大小。

JBOD配置

要将JBOD与Strimzi一起使用，必须将存储设置为。该属性使您能够描述组成JBOD存储阵列或配置的磁盘。以下片段显示了示例JBOD配置： `type` `jbod` `volumes`

```
# ...
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，将无法更改ID。

用户可以从JBOD配置中添加或删除卷。

JBOD和持久量声明

当使用持久性存储声明JBOD卷时，生成的“持久性卷声明”的命名方案如下：

data-id-cluster-name-kafka-idx其中id是用于存储Kafka代理pod idx的数据的卷的ID。

日志目录

Kafka代理会将JBOD卷用作安装在以下路径中的日志目录：

/ var / lib / kafka / data-id / kafka-log_idx_Where id是用于为Kafka经纪人pod idx存储数据的卷的ID。例如/ var / lib / kafka / data-0 / kafka-log0。

将卷添加到JBOD存储

此过程描述了如何将卷添加到配置为使用JBOD存储的Kafka群集中。它不能应用于配置为使用任何其他存储类型的Kafka群集。

注意	在过去已使用过并已删除的卷下添加新卷时，必须确保已删除先前使用的卷。 `id` `PersistentVolumeClaims`

先决条件

- Kubernetes集群
- 正在运行的集群 Operator
- 具有JBOD存储的Kafka集群

程序

1. 编辑资源中的属性。将新卷添加到阵列。例如，添加具有ID的新卷： `spec.kafka.storage.volumes` `Kafka` `volumes` `2`

```
apiVersion: kafka.strimzi.io/v1beta1
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。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f KAFKA-CONFIG-FILE
   ```

3. 创建新 Topic 或将现有分区重新分配给新磁盘。

额外资源

有关重新分配 Topic 的更多信息，请参阅分区重新分配。

从JBOD存储中删除卷

此过程描述了如何从配置为使用JBOD存储的Kafka群集中删除卷。它不能应用于配置为使用任何其他存储类型的Kafka群集。JBOD存储器始终必须包含至少一个卷。

重要	为避免数据丢失，必须在删除卷之前移动所有分区。

先决条件

- Kubernetes集群
- 正在运行的集群 Operator
- 具有带有两个或更多卷的JBOD存储的Kafka集群

程序

1. 从要删除的磁盘上重新分配所有分区。仍分配给要删除的磁盘的分区中的所有数据都可能会丢失。
2. 编辑资源中的属性。从阵列中删除一个或多个卷。例如，删除具有id 和的卷： `spec.kafka.storage.volumes` `Kafka` `volumes` `2`

```
apiVersion: kafka.strimzi.io/v1beta1
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。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

额外资源

有关重新分配 Topic 的更多信息，请参阅分区重新分配。

2.1.4。Kafka经纪人副本

Kafka集群可以与许多代理一起运行。您可以在中配置用于Kafka集群的代理数量。必须根据您的特定用例来确定集群的最佳代理数量。 `Kafka.spec.kafka.replicas`

配置代理节点数

此过程描述了如何在新群集中配置Kafka代理节点的数量。它仅适用于没有分区的新群集。如果您的集群已经定义了 Topic ，请参阅扩展集群。先决条件

- Kubernetes集群
- 正在运行的集群 Operator
- 一个尚未定义 Topic 的Kafka集群

程序

1. 编辑资源中的属性。例如： `replicas` `Kafka`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
       replicas: 3
       # ...
     zookeeper:
       # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

额外资源

如果您的集群已经定义了 Topic ，请参阅扩展集群。

2.1.5。Kafka经纪人配置

Strimzi允许您自定义Kafka集群中Kafka代理的配置。您可以指定和配置Apache Kafka文档的“代理配置”部分中列出的大多数选项。您不能配置与以下区域相关的选项：

- 安全性（加密，认证和授权）
- 侦听器配置
- 经纪人ID配置
- 日志数据目录的配置
- 经纪人之间的沟通
- ZooKeeper连接

这些选项由Strimzi自动配置。

有关代理配置的更多信息，请参阅模式。授权访问Kafka `KafkaClusterSpec`

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

配置Kafka经纪人

您可以配置现有的Kafka代理，也可以使用指定的配置创建新的Kafka代理。
先决条件

- Kubernetes集群可用。
- 群集 Operator 正在运行。

程序

1. 打开YAML配置文件，该文件包含指定群集部署的资源。 `Kafka`
2. 在资源的属性中，输入一个或多个Kafka配置设置。例如： `spec.kafka.config` `Kafka`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  kafka:
    # ...
    config:
      default.replication.factor: 3
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 1
    # ...
  zookeeper:
    # ...
```

3. 应用新配置以创建或更新资源。

   用途： `kubectl apply`

   ```
   kubectl apply -f kafka.yaml
   ```

   您要配置的资源的YAML配置文件在哪里；例如，。 *`kafka.yaml`* `kafka-persistent.yaml`

2.1.6。Kafka经纪人听众

您可以配置在Kafka代理中启用的侦听器。支持以下类型的侦听器：

- 端口9092上的纯侦听器（无TLS加密）
- 端口9093上的TLS侦听器（具有TLS加密）
- 端口9094上的外部侦听器，可从Kubernetes外部进行访问

有关侦听器配置的更多信息，请参阅架构参考。配置监听器以保护对Kafka代理的访问 `KafkaListeners`

您可以使用身份验证为安全连接配置侦听器。有关保护对Kafka经纪人的访问权的更多信息，请参阅管理对Kafka的访问。配置外部侦听器以在Kubernetes外部进行客户端访问

您可以使用指定的连接机制（例如负载均衡器）为Kubernetes环境外部的客户端访问配置外部侦听器。有关用于连接外部客户端的配置选项的更多信息，请参阅配置外部侦听器。侦听器证书

您可以为TLS侦听器或启用了TLS加密的外部侦听器提供自己的服务器证书，称为*Kafka侦听器证书*。有关更多信息，请参阅Kafka侦听器证书。

2.1.7。ZooKeeper副本

ZooKeeper群集或集合通常以奇数个节点（通常为三个，五个或七个）运行。

为了保持有效的仲裁，大多数节点必须可用。如果ZooKeeper集群失去其法定人数，它将停止对客户的响应，而Kafka经纪人将停止工作。拥有稳定且高度可用的ZooKeeper集群对于Strimzi至关重要。

三节点群集三节点ZooKeeper群集至少需要两个节点才能启动并运行，以维持仲裁。它只能容忍一个节点不可用。五节点群集五节点ZooKeeper群集至少需要启动三个节点才能运行并维持仲裁。它可以容忍两个节点不可用。七个节点的群集七个节点的ZooKeeper群集至少需要四个节点才能正常运行，以维持仲裁。它可以容忍三个节点不可用。

注意	出于开发目的，也可以在单个节点上运行ZooKeeper。

拥有更多的节点并不一定意味着更好的性能，因为维护仲裁的成本将随着集群中节点的数量而增加。根据您的可用性要求，您可以决定要使用的节点数。

ZooKeeper节点数

可以使用中的属性配置ZooKeeper节点的数量。显示副本配置的示例 `replicas` `Kafka.spec.zookeeper`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
    replicas: 3
    # ...
```

更改ZooKeeper副本的数量

先决条件

- Kubernetes集群可用。
- 群集 Operator 正在运行。

程序

1. 打开YAML配置文件，该文件包含指定群集部署的资源。 `Kafka`
2. 在资源的属性中，输入复制的ZooKeeper服务器的数量。例如： `spec.zookeeper.replicas` `Kafka`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
     zookeeper:
       # ...
       replicas: 3
       # ...
   ```

3. 应用新配置以创建或更新资源。

   用途： `kubectl apply`

   ```
   kubectl apply -f kafka.yaml
   ```

   您要配置的资源的YAML配置文件在哪里；例如，。 *`kafka.yaml`* `kafka-persistent.yaml`

2.1.8。ZooKeeper配置

Strimzi允许您自定义Apache ZooKeeper节点的配置。您可以指定和配置ZooKeeper文档中列出的大多数选项。

不能配置的选项是与以下区域相关的选项：

- 安全性（加密，认证和授权）
- 侦听器配置
- 数据目录配置
- ZooKeeper集群组成

这些选项由Strimzi自动配置。

ZooKeeper配置

ZooKeeper节点是使用中的属性配置的。此属性包含ZooKeeper配置选项作为键。可以使用以下JSON类型之一描述值： `config` `Kafka.spec.zookeeper`

- 串
- 数
- 布尔型

用户可以指定和配置ZooKeeper文档中列出的选项，但由Strimzi直接管理的选项除外。具体来说，禁止所有键等于或以下列字符串之一开头的配置选项：

- `server.`
- `dataDir`
- `dataLogDir`
- `clientPort`
- `authProvider`
- `quorum.auth`
- `requireClientAuthScheme`

如果属性中存在禁止选项之一，则将其忽略并将警告消息打印到Cluster Operator日志文件。所有其他选项都传递给ZooKeeper。 `config`

重要	群集 Operator 不验证提供的对象中的键或值。如果提供了无效的配置，则ZooKeeper群集可能无法启动或变得不稳定。在这种情况下，对象中的配置应该是固定的，群集 Operator 会将新配置推广到所有ZooKeeper节点。 `config` `Kafka.spec.zookeeper.config`

所选选项具有默认值：

- `timeTick` 具有默认值 `2000`
- `initLimit` 具有默认值 `5`
- `syncLimit` 具有默认值 `2`
- `autopurge.purgeInterval` 具有默认值 `1`

这些选项在属性中不存在时将被自动配置。 `Kafka.spec.zookeeper.config`

使用用于TLS版本的特定*密码套件*的三个允许的配置选项进行客户端连接。密码套件结合了用于安全连接和数据传输的算法。ZooKeeper配置示例 `ssl`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  kafka:
    # ...
  zookeeper:
    # ...
    config:
      autopurge.snapRetainCount: 3
      autopurge.purgeInterval: 1
      ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" (1)
      ssl.enabled.protocols: "TLSv1.2" (2)
      ssl.protocol: "TLSv1.2" (3)
    # ...
```

1. TLS的密码套件，使用密钥交换机制，认证算法，批量加密算法和MAC算法的组合。 `ECDHE` `RSA` `AES` `SHA384`
2. 启用SSl协议。 `TLSv1.2`
3. 指定生成SSL上下文的协议。允许的值为和。 `TLSv1.2` `TLSv1.1` `TLSv1.2`

配置ZooKeeper

先决条件

- Kubernetes集群可用。
- 群集 Operator 正在运行。

程序

1. 打开YAML配置文件，该文件包含指定群集部署的资源。 `Kafka`
2. 在资源的属性中，输入一个或多个ZooKeeper配置设置。例如： `spec.zookeeper.config` `Kafka`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   spec:
     kafka:
       # ...
     zookeeper:
       # ...
       config:
         autopurge.snapRetainCount: 3
         autopurge.purgeInterval: 1
       # ...
   ```

3. 应用新配置以创建或更新资源。

   用途： `kubectl apply`

   ```
   kubectl apply -f kafka.yaml
   ```

   您要配置的资源的YAML配置文件在哪里；例如，。 *`kafka.yaml`* `kafka-persistent.yaml`

2.1.9。ZooKeeper连接

ZooKeeper服务已通过加密和身份验证进行保护，不供不属于Strimzi的外部应用程序使用。

但是，如果要使用需要连接到ZooKeeper的Kafka CLI工具（例如该工具），则可以使用Kafka容器内的终端，并通过用作ZooKeeper地址来连接到ZooKeeper的TLS隧道的本地端。 `kafka-topics` `localhost:2181`

从终端连接到ZooKeeper

大多数Kafka CLI工具可以直接连接到Kafka。因此，在通常情况下，您无需连接到ZooKeeper。如果需要，可以按照以下步骤操作。打开ZooKeeper容器内的终端以使用需要ZooKeeper连接的Kafka CLI工具。
先决条件

- Kubernetes集群可用。
- Kafka集群正在运行。
- 群集 Operator 正在运行。

程序

1. 使用Kubernetes控制台打开终端，或从CLI 运行命令。 `exec`

   例如：

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

   一定要使用。 `localhost:12181`

   您现在可以对ZooKeeper运行Kafka命令。

2.1.10。实体 Operator

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

实体 Operator 包括：

- Topic operator ， 用于管理Kafka Topic
- 用户 Operator 来管理Kafka用户

通过资源配置，在部署Kafka群集时，群集 Operator 可以部署实体 Operator ，包括一个或两个 Operator 。 `Kafka`

注意	部署时，实体 Operator 将根据部署配置包含 Operator 。

Operator 会自动配置为管理Kafka集群的 Topic 和用户。

实体 Operator 配置属性

使用中的属性来配置Entity Operator。 `entityOperator` `Kafka.spec`

该属性支持几个子属性： `entityOperator`

- `tlsSidecar`
- `topicOperator`
- `userOperator`
- `template`

该属性包含用于与ZooKeeper进行通信的TLS sidecar容器的配置。有关配置TLS sidecar的更多信息，请参阅TLS sidecar。 `tlsSidecar`

该属性包含“实体 Operator ”窗格的配置，例如标签，注释，亲和力和容忍度。有关配置模板的更多信息，请参阅自定义Kubernetes资源。 `template`

该属性包含 Topic operator 的配置。当缺少此选项时，将部署没有 Topic operator 的实体 operator 。 `topicOperator`

该属性包含用户 Operator 的配置。如果缺少此选项，则将部署实体 Operator 而不使用用户 Operator 。 `userOperator`

有关配置实体 operator 的属性的更多信息，请参见模式参考。支持两个 Operator 的基本配置示例 `EntityUserOperatorSpec`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    topicOperator: {}
    userOperator: {}
```

如果和均使用空对象（`{}`），则所有属性均使用其默认值。 `topicOperator` `userOperator`

如果缺少和属性，则不会部署Entity Operator。 `topicOperator` `userOperator`

Topic Operator 配置属性

可以使用对象内的其他选项配置 Topic Operator 部署。支持以下属性： `topicOperator`

watchedNamespace Topic operator 在其中监视KafkaTopics的Kubernetes命名空间。默认是部署Kafka集群的名称空间。reconciliationIntervalSeconds定期对帐之间的间隔，以秒为单位。默认值90.zookeeperSessionTimeoutSeconds ZooKeeper会话超时（以秒为单位）。默认的20.topicMetadataMaxAttempts从Kafka获取 Topic 元数据的尝试次数。每次尝试之间的时间定义为指数补偿。由于分区或副本的数量，创建 Topic 可能需要更多时间时，请考虑增加此值。默认值6.imageimage属性可用于配置将要使用的容器图像。有关配置自定义容器镜像的更多详细信息，请参阅容器镜像。resourcesresources属性配置分配给 Topic operator 的资源量。有关资源请求和限制配置的更多详细信息，请参阅CPU和内存资源。日志记录属性配置 Topic operator 的日志记录。有关更多详细信息，请参阅 Operator 记录器。

Topic operator 配置示例

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    topicOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
    # ...
```

用户 Operator 配置属性

可以使用对象内的其他选项来配置用户 Operator 部署。支持以下属性： `userOperator`

watchedNamespace Kubernetes命名空间，用户 Operator 在该命名空间中监视KafkaUsers。默认是部署Kafka集群的名称空间。
reconciliationIntervalSeconds定期对帐之间的间隔，以秒为单位。默认值为120.zookeeperSessionTimeoutSeconds ZooKeeper会话超时（以秒为单位）。默认值6.imageimage属性可用于配置将要使用的容器图像。有关配置自定义容器镜像的更多详细信息，请参见Container images.resources resources属性配置分配给用户 Operator 的资源量。有关资源请求和限制配置的更多详细信息，请参阅CPU和内存资源。日志记录属性配置用户 Operator 的日志记录。有关更多详细信息，请参阅 Operator 记录器。

用户 Operator 配置示例

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    userOperator:
      watchedNamespace: my-user-namespace
      reconciliationIntervalSeconds: 60
    # ...
```

Operator 记录仪

Topic operator 和用户 operator 具有可配置的记录器：

- `rootLogger.level`

Operator 使用Apache 记录器实现。 `log4j2`

使用资源中的属性来配置记录器和记录器级别。 `logging` `Kafka`

您可以通过指定记录器和直接（内联）级别或使用自定义（外部）ConfigMap来设置日志级别。如果使用ConfigMap，则将属性设置为包含外部日志记录配置的ConfigMap的名称。在ConfigMap内部，使用来描述日志记录配置。 `logging.name` `log4j2.properties`

在这里，我们看到和的示例。内联记录 `inline` `external`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    topicOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
      logging:
        type: inline
        loggers:
          rootLogger.level: INFO
    # ...
    userOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
      logging:
        type: inline
        loggers:
          rootLogger.level: INFO
# ...
```

外部记录

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    topicOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
      logging:
        type: external
        name: customConfigMap
# ...
```

额外资源

- 垃圾收集器（GC）日志记录也可以启用（或禁用）。有关GC日志记录的更多信息，请参阅JVM配置。
- 有关日志级别的更多信息，请参阅Apache日志服务。

配置实体 operator

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性。例如： `entityOperator` `Kafka`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    topicOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
    userOperator:
      watchedNamespace: my-user-namespace
      reconciliationIntervalSeconds: 60
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.1.11。CPU和内存资源

对于每个部署的容器，Strimzi允许您请求特定资源并定义这些资源的最大消耗量。

Strimzi支持两种类型的资源：

- 中央处理器
- 记忆

Strimzi使用Kubernetes语法指定CPU和内存资源。

资源限制和要求

使用以下资源中的属性配置资源限制和请求： `resources`

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `Kafka.spec.entityOperator.topicOperator`
- `Kafka.spec.entityOperator.userOperator`
- `Kafka.spec.entityOperator.tlsSidecar`
- `Kafka.spec.KafkaExporter`
- `KafkaConnect.spec`
- `KafkaConnectS2I.spec`
- `KafkaBridge.spec`

额外资源

- 有关在Kubernetes上管理计算资源的更多信息，请参阅管理容器的计算资源。

资源要求

请求指定要为给定容器保留的资源。保留资源可确保它们始终可用。

重要	如果资源请求所需要的资源超出了Kubernetes集群中的可用空闲资源，则未安排Pod。

资源请求在属性中指定。Strimzi当前支持的资源请求： `requests`

- `cpu`
- `memory`

可以为一个或多个支持的资源配置请求。
所有资源的示例资源请求配置

```
# ...
resources:
  requests:
    cpu: 12
    memory: 64Gi
# ...
```

资源限制

限制指定给定容器可以消耗的最大资源。该限制不是保留的，可能并不总是可用。容器只有在可用时才能使用资源。资源限制应始终高于资源请求。

资源限制在属性中指定。Strimzi当前支持的资源限制： `limits`

- cpu
- memory

可以为一个或多个受支持的限制配置资源。
示例资源限制配置

```
# ...
resources:
  limits:
    cpu: 12
    memory: 64Gi
# ...
```

支持的CPU格式

支持以下格式的CPU请求和限制：

- CPU内核数，可以是整数（5 CPU内核）或十进制（2.5 CPU内核）。
- 数字或*毫微克 / 毫厘*（），其中1000 *毫欧*是同一CPU核心。 100m 1

示例CPU单元

```
# ...
resources:
  requests:
    cpu: 500m
  limits:
    cpu: 2.5
# ...
```

注意	1个CPU内核的计算能力可能会因所部署的Kubernetes平台而异。

额外资源

- 有关CPU规格的更多信息，请参见CPU的含义。

支持的内存格式

内存请求和限制以兆字节，千兆字节，兆字节和千兆字节指定。

- 要指定兆字节的内存，请使用后缀。例如。 M 1000M
- 要以GB为单位指定内存，请使用后缀。例如。 G 1G
- 要以兆字节为单位指定内存，请使用后缀。例如。 Mi 1000Mi
- 要以千兆字节指定内存，请使用后缀。例如。 Gi 1Gi

使用不同存储单元的示例

```
# ...
resources:
  requests:
    memory: 512Mi
  limits:
    memory: 2Gi
# ...
```

额外资源

- 有关内存规格和其他受支持单位的更多详细信息，请参阅内存的含义。

配置资源请求和限制

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性以指定集群部署。例如： resources

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  kafka:
    # ...
    resources:
      requests:
        cpu: "8"
        memory: 64Gi
      limits:
        cpu: "12"
        memory: 128Gi
    # ...
  zookeeper:
    # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

额外资源

- 有关架构的更多信息，请参见ResourceRequirements API参考。

2.1.12。Kafka记录器

Kafka有自己的可配置记录器：

- log4j.logger.org.I0Itec.zkclient.ZkClient
- log4j.logger.org.apache.zookeeper
- log4j.logger.kafka
- log4j.logger.org.apache.kafka
- log4j.logger.kafka.request.logger
- log4j.logger.kafka.network.Processor
- log4j.logger.kafka.server.KafkaApis
- log4j.logger.kafka.network.RequestChannel$
- log4j.logger.kafka.controller
- log4j.logger.kafka.log.LogCleaner
- log4j.logger.state.change.logger
- log4j.logger.kafka.authorizer.logger

ZooKeeper也有一个可配置的记录器：

- zookeeper.root.logger

Kafka和ZooKeeper使用Apache 记录器实现。 `log4j`

Operator 使用Apache 记录器实现，因此使用来在ConfigMap中描述记录配置。有关更多信息，请参阅 Operator 记录器。 `log4j2` `log4j2.properties`

使用该属性来配置记录器和记录器级别。 `logging`

您可以通过指定记录器和直接（内联）级别或使用自定义（外部）ConfigMap来设置日志级别。如果使用ConfigMap，则将属性设置为包含外部日志记录配置的ConfigMap的名称。在ConfigMap内部，使用来描述日志记录配置。 `logging.name` `log4j.properties`

在这里，我们看到和的示例。内联记录 `inline` `external`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  # ...
  kafka:
    # ...
    logging:
      type: inline
      loggers:
        log4j.logger.kafka: "INFO"
  # ...
  zookeeper:
    # ...
    logging:
      type: inline
      loggers:
        zookeeper.root.logger: "INFO"
  # ...
  entityOperator:
    # ...
    topicOperator:
      # ...
      logging:
        type: inline
        loggers:
          rootLogger.level: INFO
    # ...
    userOperator:
      # ...
      logging:
        type: inline
        loggers:
          rootLogger.level: INFO
    # ...
```

外部记录

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  # ...
  logging:
    type: external
    name: customConfigMap
  # ...
```

外部日志记录级别和内联日志记录级别的更改都将应用于Kafka代理，而无需重新启动。
额外资源

- 垃圾收集器（GC）日志记录也可以启用（或禁用）。有关垃圾回收的更多信息，请参阅JVM配置。
- 有关日志级别的更多信息，请参阅Apache日志服务。

2.1.13。Kafka货架意识

Strimzi中的机架感知功能有助于将Kafka经纪人吊舱和Kafka Topic 副本分布在不同机架上。启用机架意识有助于提高Kafka代理及其托管 Topic 的可用性。

注意	“机架”可能表示可用区，数据中心或数据中心中的实际机架。

在Kafka经纪人中配置机架意识

可以在的属性中配置Kafka机架感知。该对象具有一个名为的必填字段。该密钥需要匹配分配给Kubernetes集群节点的标签之一。在将Kafka代理容器调度到节点时，Kubernetes将使用该标签。如果Kubernetes集群在云提供商平台上运行，则该标签应代表节点正在运行的可用性区域。通常，节点被标记为标签（或在较旧的Kubernetes版本上），可以用作值。有关Kubernetes节点标签的更多信息，请参见知名标签，注释和污点。这具有将经纪人吊舱散布到各个区域的效果，并且还可以设置经纪人的 `rack` `Kafka.spec.kafka` `rack` `topologyKey` `topology.kubernetes.io/zone` `failure-domain.beta.kubernetes.io/zone` `topologyKey` `broker.rack` Kafka代理中的配置参数。
先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 有关代表节点部署所在的区域/机架的节点标签，请咨询您的Kubernetes管理员。
2. 使用标签作为拓扑键来编辑资源中的属性。 `rack` `Kafka`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    rack:
      topologyKey: topology.kubernetes.io/zone
    # ...
```

3. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

额外资源

- 有关为Kafka机架感知配置初始容器镜像的信息，请参阅容器镜像。

2.1.14。健康检查

运行状况检查是定期测试，可验证应用程序的运行状况。当运行状况检查探针失败时，Kubernetes会认为该应用程序运行状况不佳，并尝试对其进行修复。

Kubernetes支持两种类型的Healthcheck探针：

- 活力探针
- 准备就绪探针

有关探针的更多详细信息，请参阅“ 配置活动性和就绪性探针”。两种类型的探针都用于Strimzi组件中。

用户可以为活动和就绪探针配置选定的选项。

健康检查配置

可以使用以下资源中的和属性来配置活动性和就绪性探针： `livenessProbe` `readinessProbe`

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `Kafka.spec.entityOperator.tlsSidecar`
- `Kafka.spec.entityOperator.topicOperator`
- `Kafka.spec.entityOperator.userOperator`
- `Kafka.spec.KafkaExporter`
- `KafkaConnect.spec`
- `KafkaConnectS2I.spec`
- `KafkaMirrorMaker.spec`
- `KafkaBridge.spec`

双方并支持以下选项： `livenessProbe` `readinessProbe`

- `initialDelaySeconds`
- `timeoutSeconds`
- `periodSeconds`
- `successThreshold`
- `failureThreshold`

有关和选项的更多信息，请参见模式参考。活动和就绪探针配置示例 `livenessProbe` `readinessProbe` `Probe`

```
# ...
readinessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
# ...
```

配置健康检查

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑或财产的，或资源。例如： `livenessProbe` `readinessProbe` `Kafka` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
       readinessProbe:
         initialDelaySeconds: 15
         timeoutSeconds: 5
       livenessProbe:
         initialDelaySeconds: 15
         timeoutSeconds: 5
       # ...
     zookeeper:
       # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.1.15。普罗米修斯指标

Strimzi使用Prometheus JMX导出器支持Prometheus指标，以将Apache Kafka和ZooKeeper支持的JMX指标转换为Prometheus指标。启用度量后，它们将在端口9404上公开。

有关设置和部署Prometheus和Grafana的更多信息，请参阅*Deploying Strimzi*指南中的向Kafka引入度量。

指标配置

通过在以下资源中配置属性来启用Prometheus指标： `metrics`

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `KafkaConnect.spec`
- `KafkaConnectS2I.spec`

当资源中未定义属性时，将禁用Prometheus指标。要在不进行任何其他配置的情况下启用Prometheus指标导出，您可以将其设置为空对象（）。无需进一步配置即可启用指标的示例 `metrics {}`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    metrics: {}
    # ...
  zookeeper:
    # ...
```

该属性可能包含Prometheus JMX导出器的其他配置。使用其他Prometheus JMX Exporter配置启用指标的示例 `metrics`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    metrics:
      lowercaseOutputName: true
      rules:
        - pattern: "kafka.server<type=(.+), name=(.+)PerSec\\w*><>Count"
          name: "kafka_server_$1_$2_total"
        - pattern: "kafka.server<type=(.+), name=(.+)PerSec\\w*, topic=(.+)><>Count"
          name: "kafka_server_$1_$2_total"
          labels:
            topic: "$3"
    # ...
  zookeeper:
    # ...
```

配置Prometheus指标

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑在属性，或资源。例如： `metrics` `Kafka` `KafkaConnect` `KafkaConnectS2I`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
    metrics:
      lowercaseOutputName: true
    # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.1.16。JMX选项

Strimzi支持通过在9999上打开JMX端口来从Kafka代理获取JMX指标。您可以获取有关每个Kafka代理的各种指标，例如，使用数据，例如代理网络的值或请求率。Strimzi支持打开密码和用户名受保护的JMX端口或不受保护的JMX端口。 `BytesPerSecond`

配置JMX选项

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

您可以使用以下资源中的属性来配置JMX选项： `jmxOptions`

- `Kafka.spec.kafka`

您可以为在Kafka代理上打开的JMX端口配置用户名和密码保护。
保护JMX端口

您可以保护JMX端口，以防止未经授权的Pod访问该端口。当前，只能使用用户名和密码保护JMX端口。要为JMX端口启用安全性，请将字段中的参数设置为。： `type` `authentication` `password`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    jmxOptions:
      authentication:
        type: "password"
    # ...
  zookeeper:
    # ...
```

这使您可以在内部将Pod部署到集群中，并通过使用无头服务并指定要处理的代理来获取JMX指标。为了从代理*0*获得JMX指标，我们解决了无头服务，将代理*0*附加在无头服务之前：

"<*cluster-name*>-kafka-0-<*cluster-name*>-<*headless-service-name*>"

如果JMX端口是安全的，则可以通过在pod部署中从JMX机密引用用户名和密码来获取用户名和密码。
使用开放的JMX端口

要禁用JMX端口的安全性，请不要填写该字段 `authentication`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    jmxOptions: {}
    # ...
  zookeeper:
    # ...
```

这只会打开无头服务上的JMX端口，您可以按照上述类似的方法将Pod部署到集群中。唯一的区别是任何Pod都可以从JMX端口读取。

2.1.17。使用JMXTrans检索JMX指标

JmxTrans是一种从Java流程中检索JMX指标数据并将其以各种格式推送到集群内部或外部的远程接收器的方法。JmxTrans可以与安全的JMX端口进行通信。Strimzi支持使用JmxTrans从Kafka代理读取JMX数据。

跨界

JmxTrans从安全或不安全的Kafka代理读取JMX指标数据，并将数据以各种数据格式推送到远程接收器。Jmxtrans的一个示例用例将是获取有关每个Kafka经纪人网络的请求速率的JMX度量，并将其推送到Kubernetes集群之外的Logstash数据库。

配置JMXTrans部署

先决条件

- 正在运行的Kubernetes集群

您可以使用该属性配置JmxTrans部署。JmxTrans部署可以从安全或不安全的Kafka代理读取。要配置JmxTrans部署，请定义以下属性： `Kafka.spec.jmxTrans`

- `Kafka.spec.jmxTrans.outputDefinitions`
- `Kafka.spec.jmxTrans.kafkaQueries`

有关这些属性的更多信息，请参见JmxTransSpec模式参考。

注意	如果您在Kafka代理上指定JmxOptions，JmxTrans将不会出现。有关更多信息，请参见Kafka Jmx选项。配置JmxTrans输出定义

输出定义指定将JMX指标推送到何处以及采用哪种数据格式。有关支持的数据格式的信息，请参阅数据格式。可以通过该属性配置JmxTrans代理在推送新数据之前等待的秒数。的和属性定义被推到目标主机地址和目标端口的数据。该属性是该属性引用的必需属性。 `flushDelay` `host` `port` `name` `Kafka.spec.kafka.jmxOptions.jmxTrans.queries`

这是一个示例配置，每隔5秒将Graphite格式的JMX数据推送到http：// myLogstash：9999上的Logstash数据库，另一次推送到（标准输出）： `standardOut`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  jmxTrans:
    outputDefinitions:
      - outputType: "com.googlecode.jmxtrans.model.output.GraphiteWriter"
        host: "http://myLogstash"
        port: 9999
        flushDelay: 5
        name: "logstash"
      - outputType: "com.googlecode.jmxtrans.model.output.StdOutWriter"
        name: "standardOut"
        # ...
    # ...
  zookeeper:
    # ...
```

配置JmxTrans查询

JmxTrans查询指定从Kafka代理读取哪些JMX度量。目前，JmxTrans查询只能发送到Kafka经纪人。配置该属性以指定要寻址Kafka代理上的哪个目标MBean。配置该属性指定从目标MBean读取哪个MBean属性作为JMX度量。JmxTrans支持通配符从目标MBean读取，并通过指定进行过滤。该属性通过指定输出定义的名称来定义将度量推送到的位置。 `targetMBean` `attributes` `typenames` `outputs`

以下JmxTrans部署从与该模式匹配且具有目标MBean名称的所有MBean中读取。从这些Mbeans中，它获取有关该属性的JMX度量，并将该度量推送到所定义的标准输出。 `kafka.server:type=BrokerTopicMetrics,name=*` `name` `Count` `outputs`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  jmxTrans:
    kafkaQueries:
      - targetMBean: "kafka.server:type=BrokerTopicMetrics,*"
        typeNames: ["name"]
        attributes:  ["Count"]
        outputs: ["standardOut"]
  zookeeper:
    # ...
```

额外资源

有关Jmxtrans的更多信息，请参见Jmxtrans github。

2.1.18。JVM选项

Strimzi的以下组件在虚拟机（VM）中运行：

- 阿帕奇•Kafka
- Apache ZooKeeper
- Apache Kafka连接
- Apache Kafka MirrorMaker
- Strimzi Kafka桥

JVM配置选项可优化不同平台和体系结构的性能。Strimzi允许您配置其中一些选项。

JVM配置

使用该属性可以为运行组件的JVM配置受支持的选项。 `jvmOptions`

支持的JVM选项有助于优化不同平台和体系结构的性能。

配置JVM选项

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑在属性，，，，或资源。例如： `jvmOptions` `Kafka` `KafkaConnect` `KafkaConnectS2I` `KafkaMirrorMaker` `KafkaBridge`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
       jvmOptions:
         "-Xmx": "8g"
         "-Xms": "8g"
       # ...
     zookeeper:
       # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.1.19。容器图片

Strimzi允许您配置将用于其组件的容器镜像。仅在需要使用其他容器注册表的特殊情况下才建议覆盖容器镜像。例如，由于您的网络不允许访问Strimzi使用的容器存储库。在这种情况下，您应该复制Strimzi图像或从源代码构建它们。如果配置的图像与Strimzi图像不兼容，则可能无法正常工作。

容器镜像配置

使用该属性指定要使用的容器图像。 `image`

警告	仅在特殊情况下才建议覆盖容器镜像。

配置容器镜像

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑在属性，或资源。例如： `image` `Kafka` `KafkaConnect` `KafkaConnectS2I`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    image: my-org/my-image:latest
    # ...
  zookeeper:
    # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.1.20。TLS边车

边车是一种在吊舱中运行但用于支撑目的的容器。在Strimzi中，TLS边车使用TLS加密和解密各个组件与ZooKeeper之间的所有通信。

TLS Sidecar用于：

- 实体 Operator
- 巡航控制

TLS Sidecar配置

可以使用以下属性配置TLS Sidecar ： `tlsSidecar`

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `Kafka.spec.entityOperator`

TLS Sidecar支持以下其他选项：

- `image`
- `resources`
- `logLevel`
- `readinessProbe`
- `livenessProbe`

该属性可用于指定为TLS边车分配的内存和CPU资源。 `resources`

该属性可用于配置将要使用的容器镜像。有关配置自定义容器镜像的更多详细信息，请参阅容器镜像。 `image`

该属性用于指定日志记录级别。支持以下日志记录级别： `logLevel`

- 能源
- 警报
- 暴击
- 呃
- 警告
- 注意
- 信息
- 调试

默认值为*note*。

有关配置运行状况检查的和属性的更多信息，请参阅运行状况检查配置。TLS边车配置示例 `readinessProbe` `livenessProbe`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    tlsSidecar:
      image: my-org/my-image:latest
      resources:
        requests:
          cpu: 200m
          memory: 64Mi
        limits:
          cpu: 500m
          memory: 128Mi
      logLevel: debug
      readinessProbe:
        initialDelaySeconds: 15
        timeoutSeconds: 5
      livenessProbe:
        initialDelaySeconds: 15
        timeoutSeconds: 5
    # ...
  zookeeper:
    # ...
```

配置TLS Sidecar

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性。例如： `tlsSidecar` `Kafka`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    tlsSidecar:
      resources:
        requests:
          cpu: 200m
          memory: 64Mi
        limits:
          cpu: 500m
          memory: 128Mi
    # ...
  cruiseControl:
    # ...
    tlsSidecar:
      resources:
        requests:
          cpu: 200m
          memory: 64Mi
        limits:
          cpu: 500m
          memory: 128Mi
    # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.1.21。配置窗格调度

重要	当将两个应用程序安排到同一Kubernetes节点时，这两个应用程序可能会使用相同的资源（如磁盘I / O）并影响性能。这会导致性能下降。避免与其他关键工作负载共享节点，使用正确的节点或仅将一组节点专用于Kafka的方式来调度Kafka Pod是避免此类问题的最佳方法。

根据其他应用程序调度Pod

避免关键应用程序共享节点

可以使用Pod抗关联性来确保关键应用程序永远不会安排在同一磁盘上。在运行Kafka群集时，建议使用pod anti-affinity以确保Kafka代理不与其他工作负载（如数据库）共享节点。

亲和力

可以使用以下资源中的属性来配置亲和力： `affinity`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

相似性配置可以包括不同类型的相似性：

- Pod亲和力和反亲和力
- 节点亲和力

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes节点和pod关联文档。 `affinity`

在Kafka组件中配置Pod抗亲和力

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性以指定集群部署。使用标签指定不应在同一节点上安排的Pod。该应设置为指定所选荚不应该使用相同的主机节点进行调度。例如： `affinity` `topologyKey` `kubernetes.io/hostname`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   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。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

将Pod调度到特定节点

节点调度

Kubernetes集群通常由许多不同类型的工作程序节点组成。有些针对CPU繁重的工作负载进行了优化，有些针对内存进行了优化，而其他针对存储（快速本地SSD）或网络进行了优化。使用不同的节点有助于优化成本和性能。为了获得最佳性能，允许安排Strimzi组件使用正确的节点很重要。

Kubernetes使用节点关联性将工作负载调度到特定节点上。节点亲缘关系允许您为将在其上调度容器的节点创建调度约束。该约束被指定为标签选择器。您可以使用内置节点标签（例如）或自定义标签来指定标签，以选择正确的节点。 `beta.kubernetes.io/instance-type`

亲和力

可以使用以下资源中的属性来配置亲和力： `affinity`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

相似性配置可以包括不同类型的相似性：

- Pod亲和力和反亲和力
- 节点亲和力

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes节点和pod关联文档。 `affinity`

在Kafka组件中配置节点关联

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 标记应安排Strimzi组件的节点。

   可以使用： `kubectl label`

   ```
   kubectl label node your-node node-type=fast-network
   ```

   或者，某些现有标签可能会被重用。
2. 编辑资源中的属性以指定集群部署。例如： `affinity`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   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。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

使用专用节点

专用节点

群集管理员可以将选定的Kubernetes节点标记为已污染。带有污点的节点不包括在常规调度中，普通的吊舱也不会安排在它们上运行。只能在节点上安排可以容忍在节点上设置污点的服务。在此类节点上运行的唯一其他服务将是系统服务，例如日志收集器或软件定义的网络。

污渍可用于创建专用节点。在专用节点上运行Kafka及其组件可具有许多优势。在同一节点上不会运行其他任何可能引起干扰或消耗Kafka所需资源的应用程序。这可以提高性能和稳定性。

要在专用节点上安排Kafka Pod，请配置节点亲和力和容差。

亲和力

可以使用以下资源中的属性来配置亲和力： `affinity`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`

- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

相似性配置可以包括不同类型的相似性：

- Pod亲和力和反亲和力
- 节点亲和力

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes节点和pod关联文档。 `affinity`

公差范围

可以使用以下资源中的属性来配置公差： `tolerations`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes污点和公差。 `tolerations`

设置专用节点并在其上调度Pod

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 选择应用作专用节点。
2. 确保在这些节点上没有安排任何工作负载。
3. 在所选节点上设置污点：

   可以使用： `kubectl taint`

   ```
   kubectl taint node your-node dedicated=Kafka:NoSchedule
   ```

4. 此外，还向所选节点添加标签。

   可以使用： `kubectl label`

   ```
   kubectl label node your-node dedicated=Kafka
   ```

5. 编辑资源中的和属性以指定集群部署。例如： `affinity` `tolerations`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   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。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.1.22。Kafka出口商

您可以配置资源以在集群中自动部署Kafka Exporter。 `Kafka`

Kafka Exporter提取数据作为Prometheus指标进行分析，主要是与抵销，消费者群体，消费者滞后和 Topic 相关的数据。

有关设置Kafka Exporter的信息以及监视消费者的性能滞后的重要性，请参阅*Deploying Strimzi*指南中的Kafka Exporter。

2.1.23。对Kafka集群执行滚动更新

此过程介绍了如何使用Kubernetes批注手动触发现有Kafka集群的滚动更新。
先决条件

有关运行以下命令的说明，请参阅《*部署Strimzi*》指南：

- 集群 Operator
- Kafka集群

程序

1. 找到控制您要手动更新的Kafka Pod 的名称。 `StatefulSet`

   例如，如果您的Kafka群集名为*my-cluster*，则对应*的群集*名为*my-cluster-kafka*。 `StatefulSet`
2. 在Kubernetes中注释资源。例如，使用： `StatefulSet` `kubectl annotate`

   ```
   kubectl annotate statefulset cluster-name-kafka strimzi.io/manual-rolling-update=true
   ```

3. 等待下一次对帐（默认情况下，每两分钟进行一次）。只要对帐过程中已检测到注释，便会触发注释内所有Pod的滚动更新。当所有Pod的滚动更新完成后，注释将从中删除。 `StatefulSet` `StatefulSet`

2.1.24。对ZooKeeper集群执行滚动更新

此过程描述了如何使用Kubernetes批注手动触发现有ZooKeeper集群的滚动更新。
先决条件

有关运行以下命令的说明，请参阅《*部署Strimzi*》指南：

- 集群 Operator
- Kafka集群

程序

1. 找到控制您要手动更新的ZooKeeper窗格的名称。 `StatefulSet`

   例如，如果您的Kafka群集名为*my-cluster*，则对应*的群集*名为*my-cluster-zookeeper*。 `StatefulSet`
2. 在Kubernetes中注释资源。例如，使用： `StatefulSet` `kubectl annotate`

   ```
   kubectl annotate statefulset cluster-name-zookeeper strimzi.io/manual-rolling-update=true
   ```

3. 等待下一次对帐（默认情况下，每两分钟进行一次）。只要对帐过程中已检测到注释，便会触发注释内所有Pod的滚动更新。当所有Pod的滚动更新完成后，注释将从中删除。 `StatefulSet` `StatefulSet`

2.1.25。扩展集群

扩展Kafka集群

将代理添加到集群

增加 Topic 吞吐量的主要方法是增加该 Topic 的分区数量。之所以可行，是因为额外的分区允许在群集中的不同代理之间共享 Topic 的负载。但是，在每个代理都受特定资源（通常是I / O）约束的情况下，使用更多分区将不会导致吞吐量增加。相反，您需要将代理添加到集群。

当您向集群添加额外的代理时，Kafka不会自动为其分配任何分区。您必须确定从现有代理迁移到新代理的分区。

一旦在所有代理之间重新分配了分区，就应该降低每个代理的资源利用率。

从集群中删除代理

由于Strimzi用于管理代理容器，因此您不能从集群中删除*任何*容器。您只能从集群中删除一个或多个编号最高的窗格。例如，在由12个经纪人组成的集群中，吊舱的名称最多为。如果您决定由一名经纪人缩减规模，则将其删除。 `StatefulSets` *`cluster-name`*`-kafka-0` *`cluster-name`*`-kafka-11` *`cluster-name`*`-kafka-11`

从群集中删除代理之前，请确保未将其分配给任何分区。您还应该确定剩余的哪个代理负责将要停用的代理上的每个分区。一旦代理没有分配的分区，就可以安全地缩小群集。

分区重新分配

Topic operator 当前不支持将副本重新分配给不同的代理，因此有必要直接连接到代理pod以便将副本重新分配给代理。

在代理窗格中，该实用程序允许您将分区重新分配给其他代理。 `kafka-reassign-partitions.sh`

它具有三种不同的模式：

--generate接收一组 Topic 和代理，并生成一个重新分配JSON文件，该文件将导致将这些 Topic 的分区分配给这些代理。由于此操作适用于整个 Topic ，因此仅在需要重新分配某些 Topic 的某些分区时不能使用它。--execute接收重新分配的JSON文件并将其应用于集群中的分区和代理。结果获得分区的经纪人成为分区负责人的追随者。对于给定的分区，一旦新代理赶上并加入了ISR（同步副本），旧代理将停止成为追随者并删除其副本。--verify使用与--execute步骤相同的重新分配JSON文件，--verify检查文件中的所有分区是否已移至其预期的代理。如果重新分配完成，--verify还会删除所有有效的节流阀。除非删除，否则即使重新分配完成后，节流阀仍将继续影响群集。

在任何给定时间只能在集群中运行一个重新分配，并且不能取消正在运行的重新分配。如果您需要取消重新分配，请等待它完成，然后再执行另一次重新分配以还原第一次重新分配的效果。在将打印此回复改派JSON作为其输出的一部分。如果需要停止进行中的重新分配，则应将非常大的重新分配分解为许多较小的重新分配。 `kafka-reassign-partitions.sh`

重新分配JSON文件

在*重新分配JSON文件*具有特定的结构：

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

其中*<PartitionObjects>*是用逗号分隔的对象列表，例如：

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

注意	尽管Kafka也支持属性，但不应在Strimzi中使用它。 `"log_dirs"`

以下是一个例子再分配JSON文件指派 Topic ，分区经纪人，和，和 Topic 分区经纪人，和： `topic-a 4 2 4 7 topic-b 2 1 5`

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

JSON中未包括的分区不会更改。

在JBOD卷之间重新分配分区

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

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

该对象应包含与该对象中指定的副本数相同数量的日志目录。该值应该是日志目录的绝对路径，也可以是关键字。 `log_dirs` `replicas` `any`

例如：

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

生成重新分配JSON文件

此过程描述了如何使用该工具生成重新分配JSON文件，该文件将为给定 Topic 集重新分配所有分区。先决条件 `kafka-reassign-partitions.sh`

- 正在运行的集群 Operator
- 一个资源 `Kafka`
- 一组 Topic 以重新分配以下内容的分区

程序

1. 准备一个名为的JSON文件，其中列出了要移动的 Topic 。它必须具有以下结构： *topics.json*

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

其中*<TopicObjects>*是逗号分隔的对象列表，例如：

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

例如，如果你要重新分配的所有分区和，你需要准备一个文件是这样的： `topic-a` `topic-b` *topics.json*

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

2. 将文件复制到代理窗格之一： *topics.json*

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

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

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

例如，移动的所有分区和经纪人和 `topic-a` `topic-b` `4` `7`

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

手动创建重新分配JSON文件

如果要移动特定的分区，可以手动创建重新分配JSON文件。

重新分配油门

分区重新分配可能是一个缓慢的过程，因为它涉及在代理之间传输大量数据。为避免对客户端造成不利影响，您可以限制重新分配过程。这可能导致重新分配需要更长的时间才能完成。

- 如果限制太低，那么新分配的代理将无法跟上发布的记录，并且重新分配将永远不会完成。
- 如果节流阀过高，则会影响客户。

例如，对于生产者，这可能表现为比等待确认的正常等待时间更长。对于消费者而言，这可能表现为由于两次轮询之间的较高延迟而导致的吞吐量下降。

扩展Kafka集群

此过程描述了如何增加Kafka群集中的代理数量。
先决条件

- 现有的Kafka集群。
- 一个名为*JSON*的*重新分配文件*，描述了应如何将分区重新分配给扩大后的集群中的代理。 *reassignment.json*

程序

1. 通过增加配置选项，根据需要添加任意数量的新代理。 `Kafka.spec.kafka.replicas`
2. 验证新的Broker Pod已启动。
3. 将文件复制到代理容器，稍后将在其上执行命令： *reassignment.json*

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

例如：

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

4. 使用命令行工具从同一代理窗格执行分区重新分配。 `kafka-reassign-partitions.sh`

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

如果要限制复制，还可以通过代理间限制速率（以字节/秒为单位）传递该选项。例如： `--throttle`

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

该命令将打印出两个重新分配的JSON对象。第一个记录正在移动的分区的当前分配。如果以后需要还原重新分配，则应将其保存到本地文件（而不是Pod中的文件）。第二个JSON对象是您在重新分配JSON文件中传递的目标重新分配。

5. 如果您需要在重新分配期间更改油门，则可以使用相同的命令行使用不同的油门率。例如：

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

6. 使用任何经纪人吊舱中的命令行工具，定期验证重新分配是否已完成。该命令与上一步相同，但带有选项而不是选项。 `kafka-reassign-partitions.sh --verify --execute`

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

例如，

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

7. 当命令报告每个已成功移动的分区时，重新分配已完成。此决赛还将具有删除所有重新分配节流阀的作用。现在，如果您保存了用于将分配还原到其原始代理的JSON，则可以删除还原文件。 `--verify --verify`

缩减Kafka集群

额外资源

此过程描述了如何减少Kafka群集中的代理数量。

先决条件

- 现有的Kafka集群。
- 名为*JSON*的*重新分配文件*，描述了一旦编号最高的代理被删除，应如何将分区重新分配给集群中的代理。 `reassignment.json` `Pod(s)`

程序

1. 将文件复制到代理容器，稍后将在其上执行命令： `reassignment.json`

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

例如：

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

2. 使用命令行工具从同一代理窗格执行分区重新分配。 `kafka-reassign-partitions.sh`

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

如果要限制复制，还可以通过代理间限制速率（以字节/秒为单位）传递该选项。例如： `--throttle`

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

该命令将打印出两个重新分配的JSON对象。第一个记录正在移动的分区的当前分配。如果以后需要还原重新分配，则应将其保存到本地文件（而不是Pod中的文件）。第二个JSON对象是您在重新分配JSON文件中传递的目标重新分配。

3. 如果您需要在重新分配期间更改油门，则可以使用相同的命令行使用不同的油门率。例如：

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

4. 使用任何经纪人吊舱中的命令行工具，定期验证重新分配是否已完成。该命令与上一步相同，但带有选项而不是选项。 `kafka-reassign-partitions.sh --verify --execute`

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

例如，

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

5. 当命令报告每个已成功移动的分区时，重新分配已完成。此决赛还将具有删除所有重新分配节流阀的作用。现在，如果您保存了用于将分配还原到其原始代理的JSON，则可以删除还原文件。 `--verify --verify`
6. 一旦完成所有分区的重新分配，要删除的代理就不应对集群中的任何分区负责。您可以通过检查代理的数据日志目录中是否包含任何活动分区日志来验证这一点。如果代理上的日志目录包含与扩展的正则表达式不匹配的目录，则代理仍然具有活动分区，因此不应停止它。 `[a-zA-Z0-9.-]+\.[a-z0-9]+-delete$`

您可以通过执行以下命令进行检查：

```
kubectl 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*是要删除的号码。 `Pod(s)`

如果以上命令显示任何输出，则代理仍然具有活动分区。在这种情况下，或者重新分配未完成，或者重新分配JSON文件不正确。

7. 一旦你已经证实，经纪人就没有活的分区，您可以编辑您的资源，这将缩小的，删除编号最高的券商。 `Kafka.spec.kafka.replicas` `Kafka` `StatefulSet` `Pod(s)`

2.1.26。手动删除Kafka节点

额外资源

此过程描述了如何使用Kubernetes批注删除现有的Kafka节点。删除Kafka节点包括同时删除运行Kafka代理的节点和相关的节点（如果群集是使用持久性存储部署的）。删除后，和及其相关的文件会自动重新创建。 `Pod` `PersistentVolumeClaim` `Pod` `PersistentVolumeClaim`

警告	删除可能会导致永久性数据丢失。仅当遇到存储问题时，才应执行以下过程。 `PersistentVolumeClaim`

先决条件

有关运行以下命令的说明，请参阅《*部署Strimzi*》指南：

- 集群 Operator
- Kafka集群

程序

1. 查找您要删除的名称。 `Pod`

例如，如果群集名为*cluster-name*，则吊舱的名称为*cluster-name* `-kafka-` *index*，其中*index*始于零，终止于副本总数。

2. 在Kubernetes中注释资源。 `Pod`

用途： `kubectl annotate`

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

3. 当带有基础持久性卷声明的带注释的吊舱将被删除然后重新创建时，请等待下一个对帐。

2.1.27。手动删除ZooKeeper节点

此过程描述了如何使用Kubernetes批注删除现有的ZooKeeper节点。删除ZooKeeper节点包括删除正在运行ZooKeeper的节点和相关的节点（如果群集是通过持久性存储部署的）。删除后，和及其相关的文件会自动重新创建。 `Pod` `PersistentVolumeClaim` `Pod` `PersistentVolumeClaim`

警告	删除可能会导致永久性数据丢失。仅当遇到存储问题时，才应执行以下过程。 `PersistentVolumeClaim`

先决条件

有关运行以下命令的说明，请参阅《*部署Strimzi*》指南：

- 集群 Operator
- Kafka集群

程序

1. 查找您要删除的名称。 `Pod`

   例如，如果群集名为*cluster-name*，则吊舱的名称为*cluster-name* -zookeeper- *index*，其中*index*始于零，终止于副本总数。
2. 在Kubernetes中注释资源。 `Pod`

   用途： `kubectl annotate`

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

3. 当带有基础持久性卷声明的带注释的吊舱将被删除然后重新创建时，请等待下一个对帐。

2.1.28。维护时间窗口，用于滚动更新

维护时间窗口使您可以安排对Kafka和ZooKeeper群集的某些滚动更新，以便在方便的时间开始。

维护时间窗口概述

在大多数情况下，群集 Operator 仅响应于对相应资源的更改而更新您的Kafka或ZooKeeper群集。这使您能够计划何时将更改应用于资源，以最大程度地减少对Kafka客户端应用程序的影响。 `Kafka` `Kafka`

但是，可以对Kafka和ZooKeeper群集进行某些更新，而无需对资源进行任何相应更改。例如，如果集群 Operator 管理的CA（证书颁发机构）证书即将到期，则它将需要执行滚动重启。 `Kafka`

尽管Pod的滚动重启不会影响服务的*可用性*（假设正确的代理和 Topic 配置），但可能会影响Kafka客户端应用程序的*性能*。维护时间窗口使您可以安排对Kafka和ZooKeeper群集的此类自发滚动更新，以在方便的时间开始。如果未为群集配置维护时间窗口，则可能会在不方便的时间（例如在可预测的高负载期间）进行此类自发滚动更新。

维护时间窗口定义

您可以通过在属性中输入字符串数组来配置维护时间窗口。每个字符串都是cron表达式，解释为采用UTC（世界标准时间，实际上与格林威治标准时间相同）。 `Kafka.spec.maintenanceTimeWindows`

以下示例配置一个维护时间窗口，该窗口从午夜开始，在周日，周一，周二，周三和周四的凌晨01:59（UTC）结束：

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

实际上，应与资源的和属性一起设置维护窗口，以确保可以在配置的维护时间窗口中完成必要的CA证书续订。 `Kafka.spec.clusterCa.renewalDays` `Kafka.spec.clientsCa.renewalDays` `Kafka`

注意	Strimzi并未完全根据给定的窗口安排维护操作。相反，对于每个对帐，它都会检查维护窗口当前是否处于“打开”状态。这意味着在给定的时间范围内，维护操作的开始可以延迟最多Cluster Operator对帐间隔。因此，维护时间窗口必须至少为该时间长度。

额外资源

- 有关Cluster Operator配置的更多信息，请参阅Cluster Operator配置。

配置维护时间窗口

您可以配置维护时间窗口，以滚动支持的流程触发的更新。
先决条件

- Kubernetes集群
- 群集 Operator 正在运行。

程序

1. 在资源中添加或编辑属性。例如，要允许在0800和1059之间以及在1400和1559之间进行维护，可以如下所示进行设置： `maintenanceTimeWindows` `Kafka` `maintenanceTimeWindows`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
     zookeeper:
       # ...
     maintenanceTimeWindows:
       - "* * 8-10 * * ?"
       - "* * 14-15 * * ?"
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

额外资源

- 执行Kafka集群的滚动更新，请参阅执行 Kafka集群的滚动更新
- 执行ZooKeeper集群的滚动更新，请参阅执行 ZooKeeper集群的滚动更新

2.1.29。手动续订CA证书

群集和客户端CA证书会在其各自的证书续订期开始时自动续订。如果和设置为，则CA证书不会自动更新。 `Kafka.spec.clusterCa.generateCertificateAuthority` `Kafka.spec.clientsCa.generateCertificateAuthority` `false`

您可以在证书续订期开始之前手动续订这两个证书中的一个或两个。您可能出于安全原因，或者如果您更改了证书的续约或有效期，可能会这样做。

续订的证书使用与旧证书相同的私钥。
先决条件

- 群集 Operator 正在运行。
- 安装有CA证书和私钥的Kafka群集。

程序

1. 将注释应用于包含要续订的CA证书的。 `strimzi.io/force-renew` `Secret`

   表1.强制更新证书的机密注释

证书	秘密	注释命令
群集CA	*KAFKA群集名称* -cluster-ca-cert	kubectl annotate secret *KAFKA-CLUSTER-NAME*-cluster-ca-cert strimzi.io/force-renew=true
客户CA	*KAFKA群集名称* -clients-ca-cert	kubectl annotate secret *KAFKA-CLUSTER-NAME*-clients-ca-cert strimzi.io/force-renew=true

   在下一个对帐中，群集 Operator 将为您注释的生成新的CA证书。如果配置了维护时间窗口，则群集 Operator 将在下一个维护时间窗口内的第一次对帐中生成新的CA证书。 `Secret`

   客户端应用程序必须重新加载群集和由群集 Operator 更新的客户端CA证书。
2. 检查CA证书有效期：

   例如，使用命令： `openssl`

   ```
   kubectl get secret CA-CERTIFICATE-SECRET -o 'jsonpath={.data.CA-CERTIFICATE}' | base64 -d | openssl x509 -subject -issuer -startdate -enddate -noout
   ```

   CA证书秘密 是的名称，这是针对集群CA证书，并为客户CA证书。 `Secret` *KAFKA-CLUSTER-NAME*-cluster-ca-cert *KAFKA-CLUSTER-NAME*-clients-ca-cert

   CA-CERTIFICATE 是CA证书的名称，例如。 `jsonpath={.data.ca\.crt}`

   该命令返回和日期，这是CA证书的有效期。 `notBefore` `notAfter`

例如，对于群集CA证书：

```
subject=O = io.strimzi, CN = cluster-ca v0
issuer=O = io.strimzi, CN = cluster-ca v0
notBefore=Jun 30 09:43:54 2020 GMT
notAfter=Jun 30 09:43:54 2021 GMT
```

3. 从机密中删除旧证书。

   当组件使用新证书时，旧证书可能仍处于活动状态。删除旧证书以消除任何潜在的安全风险。

额外资源

- 机密
- 维护时间窗口，用于滚动更新
- `CertificateAuthority` 模式参考

2.1.30。更换私钥

您可以替换群集CA和客户端CA证书使用的私钥。替换私钥后，群集 Operator 将为新私钥生成一个新的CA证书。
先决条件

- 群集 Operator 正在运行。
- 安装有CA证书和私钥的Kafka群集。

程序

- 将注释应用到包含要更新的私钥的。 `strimzi.io/force-replace` `Secret`

  表2.替换私钥的命令

的私钥	秘密	注释命令
群集CA	*<群集名称>* -cluster-ca	`kubectl annotate secret <cluster-name>-cluster-ca strimzi.io/force-replace=true`
客户CA	*<群集名称>* -clients-ca	`kubectl annotate secret <cluster-name>-clients-ca strimzi.io/force-replace=true`

在下一次对帐中，集群 Operator 将：

- 为您注释的生成新的私钥 `Secret`
- 生成新的CA证书

如果配置了维护时间窗口，则群集 Operator 将在下一个维护时间窗口内的第一次对帐中生成新的私钥和CA证书。

客户端应用程序必须重新加载群集和由群集 Operator 更新的客户端CA证书。
额外资源

- 机密
- 维护时间窗口，用于滚动更新

2.1.31。作为Kafka集群的一部分创建的资源列表

Kubernetes集群中的集群 Operator 将创建以下资源：

cluster-name-kafkaStatefulSet负责管理Kafka代理pods.cluster-name-kafka-brokersService需要DNS直接解析Kafka代理pod的IP地址.cluster-name-kafka-bootstrapService可以用作引导服务器Kafkaclients.cluster-name-kafka-external-bootstrapBootstrap服务，用于从Kubernetes集群外部连接的客户端。仅在启用外部侦听器后才会创建此资源。cluster-name-kafka-pod-idService用于将流量从Kubernetes集群外部路由到单个Pod。仅在启用外部侦听器时才会创建此资源。从Kubernetes集群外部连接的客户端的集群名称-kafka-external-bootstrapBootstrap路由。仅当启用外部侦听器并将其设置为类型route时，才会创建此资源。cluster-name-kafka-pod-id路由，用于从Kubernetes集群外部到单个Pod的流量。仅在启用外部侦听器并将其设置为route.cluster-name-kafka-configConfigMap类型时，才会创建此资源，该路径包含Kafka辅助配置，并由Kafka代理pods.cluster-name-kafka-brokersSecret作为卷安装。 Kafka经纪人keys.cluster-name-kafka由Kafka brokers.cluster-name-kafkaPod中断服务预算配置的Kafka brokers.strimzi-namespace-name-cluster-name-kafka-initCluster角色绑定，由Kafka经纪人使用。 cluster-name-zookeeperStatefulSet负责管理ZooKeeper节点pods.cluster-name-zookeeper-nodesService需要DNS直接解析ZooKeeper Pod的IP地址。Kafka经纪人使用的cluster-name-zookeeper-clientService作为客户端连接到ZooKeeper节点.cluster-name-zookeeper-configConfigMap包含ZooKeeper辅助配置，并由ZooKeeper节点pods作为卷安装.cluster-name-zookeeper-使用ZooKeeper节点密钥秘密处理的节点。为ZooKeeper节点配置的集群名称-zookeeperPod中断预算。 Topic 名称和用户 operator 的集群名称-实体-Operator 部署。仅当集群 Operator 部署了带有 Topic Operator 辅助配置的Entity Operator.cluster-name-entity-topic-operator-configConfigmap时，才会创建此资源。仅当群集 Operator 部署了带有用户 Operator 辅助配置的Entity Operator.cluster-name-entity-user-operator-configConfigmap时，才会创建此资源。仅当Cluster Operator部署了带有用于与Kafka和ZooKeeper通信的Entity Operators密钥的Entity Operator.cluster-name-entity-operator-certsSecret时，才会创建此资源。仅当集群 Operator 部署了由Entity Operator.strimzi-cluster-name-topic-operator角色使用的Entity Operator.strimzi-cluster-name-topic-operator角色绑定时，才会创建此资源。实体Operator.cluster-name-cluster-caSecret与集群CA一起使用的-operatorRole绑定，用于对集群通信进行加密。cluster-name-cluster-ca-certSecret与Cluster CA公共密钥。此密钥可用于验证Kafka brokers.cluster-name-clients-caSecret与用于加密Kafka代理与Kafka客户端之间的通信的客户端CA的身份。使用客户端CA公钥的cluster-name-clients-ca-certSecret。此密钥可用于验证Kafka brokers.cluster-name-cluster-operator-certsSecret与用于与Kafka和ZooKeeper进行通信的集群 Operator 密钥的秘密.data-cluster-name-kafka-idx为Kafka经纪人pod idx存储数据。仅当选择持久性存储以供应持久性卷以存储数据时，才会创建此资源。data-id-cluster-name-kafka-idxPersistent Volume声明用于存储Kafka代理pod idx的数据的卷ID。仅当在配置持久卷以存储数据时为JBOD卷选择了持久存储时，才创建此资源。data-cluster-name-zookeeper-idx持久卷声明用于存储ZooKeeper节点pod idx的数据的卷。仅在选择持久性存储以供应持久性卷以存储data.cluster-name-jmxSecret时，才会创建此资源，其中JMX用户名和密码用于保护Kafka代理端口。

2.2。Kafka Connect集群配置

模式参考中描述了资源的完整模式。应用于所需资源的所有标签也将应用于构成Kafka Connect集群的Kubernetes资源。这提供了一种方便的机制，可以根据需要标记资源。 `KafkaConnect` `KafkaConnect` `KafkaConnect`

2.2.1。复制品

Kafka Connect群集可以包含一个或多个节点。节点的数量在和资源中定义。运行具有多个节点的Kafka Connect群集可以提供更好的可用性和可伸缩性。但是，在Kubernetes上运行Kafka Connect时，不必为了高可用性而运行Kafka Connect的多个节点。如果将部署Kafka Connect的节点崩溃，Kubernetes将自动将Kafka Connect pod的时间表重新安排到其他节点。但是，与多个节点一起运行Kafka Connect可以提供更快的故障转移时间，因为其他节点已经启动并正在运行。 `KafkaConnect` `KafkaConnectS2I`

配置节点数

Kafka连接的节点的数目是使用配置在属性和。先决条件 `replicas` `KafkaConnect.spec` `KafkaConnectS2I.spec`

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 在或资源中编辑属性。例如： `replicas` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: KafkaConnectS2I
   metadata:
     name: my-cluster
   spec:
     # ...
     replicas: 3
     # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.2.2。引导服务器

Kafka Connect群集始终与Kafka群集结合使用。Kafka群集被指定为引导服务器列表。在Kubernetes上，理想情况下，该列表必须包含名为的Kafka集群引导服务以及端口9092（用于普通流量）或9093（用于加密流量）。 *`cluster-name`*`-kafka-bootstrap`

引导服务器列表在和中的属性中配置。必须将服务器定义为指定一个或多个Kafka代理的逗号分隔列表，或者将服务指定为成对指向Kafka代理的服务。 `bootstrapServers` `KafkaConnect.spec` `KafkaConnectS2I.spec` *`hostname`*`:_port_`

将Kafka Connect与不受Strimzi管理的Kafka群集一起使用时，可以根据群集的配置指定引导服务器列表。

配置引导服务器

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 在或资源中编辑属性。例如： `bootstrapServers` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: KafkaConnect
   metadata:
     name: my-cluster
   spec:
     # ...
     bootstrapServers: my-cluster-kafka-bootstrap:9092
     # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.2.3。使用TLS连接到Kafka经纪人

默认情况下，Kafka Connect尝试使用纯文本连接连接到Kafka代理。如果您更喜欢使用TLS，则需要其他配置。

Kafka Connect中的TLS支持

TLS支持在和中的属性中配置。该属性包含密钥名称的机密列表，用于存储证书。证书必须以X509格式存储。显示带有多个证书的TLS配置的示例 `tls` `KafkaConnect.spec` `KafkaConnectS2I.spec` `tls`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-cluster
spec:
  # ...
  tls:
    trustedCertificates:
      - secretName: my-secret
        certificate: ca.crt
      - secretName: my-other-secret
        certificate: certificate.crt
  # ...
```

当多个证书存储在同一秘密中时，可以多次列出。
显示带有来自同一机密的多个证书的TLS配置的示例

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnectS2I
metadata:
  name: my-cluster
spec:
  # ...
  tls:
    trustedCertificates:
      - secretName: my-secret
        certificate: ca.crt
      - secretName: my-secret
        certificate: ca2.crt
  # ...
```

在Kafka Connect中配置TLS

先决条件

- Kubernetes集群
- 正在运行的集群 Operator
- 如果存在，则用于TLS服务器身份验证的证书的名称，以及用于将证书存储在其中的密钥。 `Secret` `Secret`

程序

1. （可选）如果尚不存在它们，请在文件中准备用于身份验证的TLS证书，并创建一个。 `Secret`

注意	由集群 Operator 为Kafka集群创建的机密可以直接使用。

   可以使用： `kubectl create`

   ```
   kubectl create secret generic my-secret --from-file=my-file.crt
   ```

2. 在或资源中编辑属性。例如： `tls` `KafkaConnect` `KafkaConnectS2I`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  tls:
    trustedCertificates:
      - secretName: my-cluster-cluster-cert
        certificate: ca.crt
  # ...
```

3. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.2.4。通过身份验证连接到Kafka经纪人

默认情况下，Kafka Connect将尝试在不进行身份验证的情况下连接到Kafka代理。通过和资源启用身份验证。 `KafkaConnect` `KafkaConnectS2I`

Kafka Connect中的身份验证支持

认证是通过配置在属性和。该属性指定应使用的身份验证机制的类型以及取决于该机制的其他配置详细信息。支持的身份验证类型为： `authentication` `KafkaConnect.spec` `KafkaConnectS2I.spec` `authentication`

- TLS客户端身份验证
- 使用SCRAM-SHA-512机制的基于SASL的身份验证
- 使用PLAIN机制的基于SASL的身份验证
- 基于OAuth 2.0令牌的身份验证

TLS客户端身份验证

要使用TLS客户端身份验证，请将属性设置为值。TLS客户端身份验证使用TLS证书进行身份验证。该证书在属性中指定，并且始终从Kubernetes机密中加载。秘密中，证书必须以X509格式存储在两个不同的密钥下：公用密钥和专用密钥。 `type` `tls` `certificateAndKey`

注意	TLS客户端身份验证只能与TLS连接一起使用。有关Kafka Connect中TLS配置的更多详细信息，请参阅使用TLS连接到Kafka代理。

TLS客户端身份验证配置示例

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-cluster
spec:
  # ...
  authentication:
    type: tls
    certificateAndKey:
      secretName: my-secret
      certificate: public.crt
      key: private.key
  # ...
```

基于SASL的SCRAM-SHA-512身份验证

要将Kafka Connect配置为使用基于SASL的SCRAM-SHA-512身份验证，请将属性设置为。此身份验证机制需要用户名和密码。 `type` `scram-sha-512`

- 在属性中指定用户名。 `username`
- 在属性中，指定指向包含密码的链接。该属性包含的名称和属性包含其下的密码存储在里面的键的名称。 `passwordSecret` `Secret` `secretName` `Secret` `password` `Secret`

重要	不要在字段中指定实际密码。 `password`

基于SASL的示例SCRAM-SHA-512客户端身份验证配置

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-cluster
spec:
  # ...
  authentication:
    type: scram-sha-512
    username: my-connect-user
    passwordSecret:
      secretName: my-connect-user
      password: my-connect-password-key
  # ...
```

基于SASL的PLAIN身份验证

要将Kafka Connect配置为使用基于SASL的PLAIN身份验证，请将属性设置为。此身份验证机制需要用户名和密码。 `type` `plain`

警告	SASL PLAIN机制将以明文形式在网络上传输用户名和密码。如果启用了TLS加密，则仅使用SASL PLAIN身份验证。

- 在属性中指定用户名。 `username`
- 在属性中，指定指向包含密码的链接。该属性包含这样的名称，该属性包含密钥的名称，在该密钥下密码存储在内。 `passwordSecret` `Secret` `secretName` `Secret` `password` `Secret`

重要	不要在字段中指定实际密码。 `password`

显示基于SASL的PLAIN客户端身份验证配置的示例

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-cluster
spec:
  # ...
  authentication:
    type: plain
    username: my-connect-user
    passwordSecret:
      secretName: my-connect-user
      password: my-connect-password-key
  # ...
```

在Kafka Connect中配置TLS客户端身份验证

先决条件

- Kubernetes集群
- 正在运行的集群 Operator
- 如果存在，则带有用于TLS客户端身份验证的公钥和私钥的的名称，以及将其存储在 `Secret` `Secret`

程序

1. （可选）如果尚不存在，请在文件中准备用于身份验证的密钥，然后创建。 `Secret`

注意	可以使用由用户 Operator 创建的机密。

   可以使用： `kubectl create`

   ```
   kubectl create secret generic my-secret --from-file=my-public.crt --from-file=my-private.key
   ```

2. 在或资源中编辑属性。例如： `authentication` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: KafkaConnect
   metadata:
     name: my-connect
   spec:
     # ...
     authentication:
       type: tls
       certificateAndKey:
         secretName: my-secret
         certificate: my-public.crt
         key: my-private.key
     # ...
   ```

3. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

在Kafka Connect中配置SCRAM-SHA-512身份验证

先决条件

- Kubernetes集群
- 正在运行的集群 Operator
- 用于验证的用户的用户名
- 如果存在，请输入的名称以及用于身份验证的密码，以及用于存储密码的密钥 `Secret` `Secret`

程序

1. （可选）如果尚不存在，请准备一个文件，其中包含用于身份验证的密码并创建。 `Secret`

注意	可以使用由用户 Operator 创建的机密。

   可以使用： `kubectl create`

   ```
   echo -n 'PASSWORD' > MY-PASSWORD.txt
   kubectl create secret generic MY-SECRET --from-file=MY-PASSWORD.txt
   ```

2. 在或资源中编辑属性。例如： `authentication` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: KafkaConnect
   metadata:
     name: my-connect
   spec:
     # ...
     authentication:
       type: scram-sha-512
       username: MY-USERNAME
       passwordSecret:
         secretName: MY-SECRET
         password: MY-PASSWORD.txt
     # ...
   ```

3. 创建或更新资源。

   在Kubernetes上可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.2.5。Kafka Connect配置

Strimzi允许您通过编辑Apache Kafka文档中列出的某些选项来自定义Apache Kafka Connect节点的配置。

无法配置的配置选项涉及：

- Kafka集群引导地址
- 安全性（加密，认证和授权）
- 侦听器/ REST接口配置
- 插件路径配置

这些选项由Strimzi自动配置。

Kafka Connect配置

Kafka连接是使用配置在属性和。该属性包含Kafka Connect配置选项作为键。值可以是以下JSON类型之一： `config` `KafkaConnect.spec` `KafkaConnectS2I.spec`

- 串
- 数
- 布尔型

您可以指定和配置Apache Kafka文档中列出的选项，但由Strimzi直接管理的选项除外。特别是，禁止使用键等于或以下列字符串之一开头的配置选项：

- `ssl.`
- `sasl.`
- `security.`
- `listeners`
- `plugin.path`
- `rest.`
- `bootstrap.servers`

当属性中存在禁止选项时，它将被忽略，并且警告消息将打印到Cluster Operator日志文件中。所有其他选项都传递给Kafka Connect。 config

重要	群集 operator 不验证提供的对象中的键或值。提供无效的配置时，Kafka Connect群集可能无法启动或变得不稳定。在这种情况下，请在或对象中修复配置，然后集群 Operator 可以将新配置推广到所有Kafka Connect节点。 config KafkaConnect.spec.config KafkaConnectS2I .spec.config

某些选项具有默认值：

- group.id 具有默认值 connect-cluster
- offset.storage.topic 具有默认值 connect-cluster-offsets
- config.storage.topic 具有默认值 connect-cluster-configs
- status.storage.topic 具有默认值 connect-cluster-status
- key.converter 具有默认值 org.apache.kafka.connect.json.JsonConverter
- value.converter 具有默认值 org.apache.kafka.connect.json.JsonConverter

如果或属性中不存在这些选项，则会自动进行配置。 KafkaConnect.spec.config KafkaConnectS2I.spec.config

禁止选项也有例外。对于特定于TLS版本的*密码套件*，可以使用三个允许的配置选项进行客户端连接。密码套件结合了用于安全连接和数据传输的算法。您还可以配置该属性以启用或禁用主机名验证。Kafka Connect示例配置 ssl ssl.endpoint.identification.algorithm

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    group.id: my-connect-cluster
    offset.storage.topic: my-connect-cluster-offsets
    config.storage.topic: my-connect-cluster-configs
    status.storage.topic: my-connect-cluster-status
    key.converter: org.apache.kafka.connect.json.JsonConverter
    value.converter: org.apache.kafka.connect.json.JsonConverter
    key.converter.schemas.enable: true
    value.converter.schemas.enable: true
    config.storage.replication.factor: 3
    offset.storage.replication.factor: 3
    status.storage.replication.factor: 3
    ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" (1)
    ssl.enabled.protocols: "TLSv1.2" (2)
    ssl.protocol: "TLSv1.2" (3)
    ssl.endpoint.identification.algorithm: HTTPS (4)
  # ...
```

1. TLS的密码套件，使用密钥交换机制，认证算法，批量加密算法和MAC算法的组合。 ECDHE RSA AES SHA384
2. 启用SSl协议。 TLSv1.2
3. 指定生成SSL上下文的协议。允许的值为和。 TLSv1.2 TLSv1.1 TLSv1.2
4. 通过设置为启用主机名验证。空字符串将禁用验证。 HTTPS

多个实例的Kafka Connect配置

如果您正在运行Kafka Connect的多个实例，则必须更改以下属性的默认配置： config

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    group.id: connect-cluster (1)
    offset.storage.topic: connect-cluster-offsets (2)
    config.storage.topic: connect-cluster-configs (3)
    status.storage.topic: connect-cluster-status  (4)
    # ...
# ...
```

1. 实例所属的Kafka Connect群集组。
2. 存储连接器偏移量的Kafka Topic 。
3. 存储连接器和任务状态配置的Kafka Topic 。
4. 存储连接器和任务状态更新的Kafka Topic 。

注意	对于所有具有相同的Kafka Connect实例，这三个 Topic 的值必须相同。 group.id

除非您更改默认设置，否则连接到同一Kafka群集的每个Kafka Connect实例将使用相同的值进行部署。实际上，发生的事情是所有实例都耦合在一起在集群中运行并使用相同的 Topic 。

如果多个Kafka Connect群集尝试使用相同的 Topic ，则Kafka Connect将无法按预期工作并产生错误。

如果您希望运行多个Kafka Connect实例，请为每个实例更改这些属性的值。

配置Kafka Connect

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 在或资源中编辑属性。例如： `config` `KafkaConnect` `KafkaConnectS2I`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    group.id: my-connect-cluster
    offset.storage.topic: my-connect-cluster-offsets
    config.storage.topic: my-connect-cluster-configs
    status.storage.topic: my-connect-cluster-status
    key.converter: org.apache.kafka.connect.json.JsonConverter
    value.converter: org.apache.kafka.connect.json.JsonConverter
    key.converter.schemas.enable: true
    value.converter.schemas.enable: true
    config.storage.replication.factor: 3
    offset.storage.replication.factor: 3
    status.storage.replication.factor: 3
  # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

3. 如果为Kafka Connect启用了授权，请配置Kafka Connect用户以启用对Kafka Connect使用者组和 Topic 的访问。

2.2.6。Kafka Connect用户授权

如果为Kafka Connect启用了授权，则必须配置Kafka Connect用户以提供对Kafka Connect使用者组和内部 Topic 的读/写访问权限。

消费者组和内部 Topic 的属性由Strimzi自动配置，或者可以在for 或配置中显式指定。 `spec` `KafkaConnect` `KafkaConnectS2I`

以下示例显示了资源中属性的配置，需要在Kafka Connect用户配置中表示。 `KafkaConnect`

```
apiVersion: kafka.strimzi.io/v1beta1
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. 实例所属的Kafka Connect群集组。
2. 存储连接器偏移量的Kafka Topic 。
3. 存储连接器和任务状态配置的Kafka Topic 。
4. 存储连接器和任务状态更新的Kafka Topic 。

配置Kafka Connect用户授权

此过程描述了如何授权用户访问Kafka Connect。

在Kafka中使用任何类型的授权时，Kafka Connect用户都需要对使用者组的访问权限以及Kafka Connect的内部 Topic 。

此过程显示使用授权时如何提供访问。 `simple`

简单授权使用由Kafka 插件处理的ACL规则来提供正确的访问级别。有关配置资源以使用简单授权的更多信息，请参见模式参考。 `AclAuthorizer` `KafkaUser` `AclRule`

注意	运行多个实例时，使用者组和 Topic 的默认值将有所不同。

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性以向用户提供访问权限。 `authorization` `KafkaUser`

   在以下示例中，使用名称值为Kafka Connect Topic 和使用者组配置了访问权限： `literal`

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

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: KafkaUser
   metadata:
     name: my-user
     labels:
       strimzi.io/cluster: my-cluster
   spec:
     # ...
     authorization:
       type: simple
       acls:
         # access to offset.storage.topic
         - resource:
             type: topic
             name: connect-cluster-offsets
             patternType: literal
           operation: Write
           host: "*"
         - resource:
             type: topic
             name: connect-cluster-offsets
             patternType: literal
           operation: Create
           host: "*"
         - resource:
             type: topic
             name: connect-cluster-offsets
             patternType: literal
           operation: Describe
           host: "*"
         - resource:
             type: topic
             name: connect-cluster-offsets
             patternType: literal
           operation: Read
           host: "*"
         # access to status.storage.topic
         - resource:
             type: topic
             name: connect-cluster-status
             patternType: literal
           operation: Write
           host: "*"
         - resource:
             type: topic
             name: connect-cluster-status
             patternType: literal
           operation: Create
           host: "*"
         - resource:
             type: topic
             name: connect-cluster-status
             patternType: literal
           operation: Describe
           host: "*"
         - resource:
   ```

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

2. 创建或更新资源。

```
kubectl apply -f your-file
```

2.2.7。CPU和内存资源

对于每个部署的容器，Strimzi允许您请求特定资源并定义这些资源的最大消耗量。

Strimzi支持两种类型的资源：

- 中央处理器
- 记忆

Strimzi使用Kubernetes语法指定CPU和内存资源。

资源限制和要求

使用以下资源中的属性配置资源限制和请求： `resources`

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `Kafka.spec.entityOperator.topicOperator`
- `Kafka.spec.entityOperator.userOperator`
- `Kafka.spec.entityOperator.tlsSidecar`
- `Kafka.spec.KafkaExporter`
- `KafkaConnect.spec`
- `KafkaConnectS2I.spec`
- `KafkaBridge.spec`

额外资源

- 有关在Kubernetes上管理计算资源的更多信息，请参阅管理容器的计算资源。

资源要求

请求指定要为给定容器保留的资源。保留资源可确保它们始终可用。

重要	如果资源请求所需要的资源超出了Kubernetes集群中的可用空闲资源，则未安排Pod。

资源请求在属性中指定。Strimzi当前支持的资源请求： `requests`

- `cpu`

- memory

可以为一个或多个支持的资源配置请求。
所有资源的示例资源请求配置

```
# ...
resources:
  requests:
    cpu: 12
    memory: 64Gi
# ...
```

资源限制

限制指定给定容器可以消耗的最大资源。该限制不是保留的，可能并不总是可用。容器只有在可用时才能使用资源。资源限制应始终高于资源请求。

资源限制在属性中指定。Strimzi当前支持的资源限制： `limits`

- cpu
- memory

可以为一个或多个受支持的限制配置资源。
示例资源限制配置

```
# ...
resources:
  limits:
    cpu: 12
    memory: 64Gi
# ...
```

支持的CPU格式

支持以下格式的CPU请求和限制：

- CPU内核数，可以是整数（`5` CPU内核）或十进制（`2.5` CPU内核）。
- 数字或*毫微克* / *毫厘*（），其中1000 *毫欧*是同一CPU核心。 `100m` `1`

示例CPU单元

```
# ...
resources:
  requests:
    cpu: 500m
  limits:
    cpu: 2.5
# ...
```

注意	1个CPU内核的计算能力可能会因所部署的Kubernetes平台而异。

额外资源

- 有关CPU规格的更多信息，请参见CPU的含义。

支持的内存格式

内存请求和限制以兆字节，千兆字节，兆字节和千兆字节指定。

- 要指定兆字节的内存，请使用后缀。例如。 `M` `1000M`
- 要以GB为单位指定内存，请使用后缀。例如。 `G` `1G`
- 要以兆字节为单位指定内存，请使用后缀。例如。 `Mi` `1000Mi`
- 要以千兆字节指定内存，请使用后缀。例如。 `Gi` `1Gi`

使用不同存储单元的示例

```
# ...
resources:
  requests:
    memory: 512Mi
  limits:
    memory: 2Gi
# ...
```

额外资源

- 有关内存规格和其他受支持单位的更多详细信息，请参阅内存的含义。

配置资源请求和限制

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性以指定集群部署。例如： `resources`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  kafka:
    # ...
    resources:
      requests:
        cpu: "8"
        memory: 64Gi
      limits:
        cpu: "12"
        memory: 128Gi
    # ...
  zookeeper:
    # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   `kubectl apply -f your-file`

额外资源

- 有关架构的更多信息，请参见ResourceRequirements API参考。

2.2.8。Kafka Connect记录器

使用该属性来配置记录器。 `logging`

您可以通过指定记录器和直接（嵌入式）级别或使用自定义（外部）ConfigMap来设置日志级别。

2.2.9。健康检查

运行状况检查是定期测试，可验证应用程序的运行状况。当运行状况检查探针失败时，Kubernetes会认为该应用程序运行状况不佳，并尝试对其进行修复。

Kubernetes支持两种类型的Healthcheck探针：

- 活力探针
- 准备就绪探针

有关探针的更多详细信息，请参阅“ 配置活动性和就绪性探针”。两种类型的探针都用于Strimzi组件中。

用户可以为活动和就绪探针配置选定的选项。

健康检查配置

可以使用以下资源中的和属性来配置活动性和就绪性探针： `livenessProbe` `readinessProbe`

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `Kafka.spec.entityOperator.tlsSidecar`
- `Kafka.spec.entityOperator.topicOperator`
- `Kafka.spec.entityOperator.userOperator`
- `Kafka.spec.KafkaExporter`
- `KafkaConnect.spec`
- `KafkaConnectS2I.spec`
- `KafkaMirrorMaker.spec`
- `KafkaBridge.spec`

双方并支持以下选项： `livenessProbe` `readinessProbe`

- `initialDelaySeconds`
- `timeoutSeconds`
- `periodSeconds`
- `successThreshold`
- `failureThreshold`

有关和选项的更多信息，请参见模式参考。活动和就绪探针配置示例 `livenessProbe` `readinessProbe` `Probe`

```
# ...
readinessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
# ...
```

配置健康检查

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑或财产的，或资源。例如： `livenessProbe` `readinessProbe` `Kafka` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
       readinessProbe:
         initialDelaySeconds: 15
         timeoutSeconds: 5
       livenessProbe:
         initialDelaySeconds: 15
         timeoutSeconds: 5
       # ...
     zookeeper:
       # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.2.10。普罗米修斯指标

Strimzi使用Prometheus JMX导出器支持Prometheus指标，以将Apache Kafka和ZooKeeper支持的JMX指标转换为Prometheus指标。启用度量后，它们将在端口9404上公开。

有关设置和部署Prometheus和Grafana的更多信息，请参阅*Deploying Strimzi*指南中的向Kafka引入度量。

指标配置

通过在以下资源中配置属性来启用Prometheus指标： `metrics`

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `KafkaConnect.spec`
- `KafkaConnectS2I.spec`

当资源中未定义属性时，将禁用Prometheus指标。要在不进行任何其他配置的情况下启用Prometheus指标导出，您可以将其设置为空对象（）。无需进一步配置即可启用指标的示例 `metrics {}`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    metrics: {}
    # ...
  zookeeper:
    # ...
```

该属性可能包含Prometheus JMX导出器的其他配置。使用其他Prometheus JMX Exporter配置启用指标的示例 `metrics`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    metrics:
      lowercaseOutputName: true
      rules:
        - pattern: "kafka.server<type=(.+), name=(.+)PerSec\\w*><>Count"
          name: "kafka_server_$1_$2_total"
        - pattern: "kafka.server<type=(.+), name=(.+)PerSec\\w*, topic=(.+)><>Count"
          name: "kafka_server_$1_$2_total"
          labels:
            topic: "$3"
    # ...
  zookeeper:
    # ...
```

配置Prometheus指标

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑在属性，或资源。例如： `metrics` `Kafka` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
     zookeeper:
       # ...
       metrics:
         lowercaseOutputName: true
       # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.2.11。JVM选项

Strimzi的以下组件在虚拟机（VM）中运行：

- 阿帕奇·Kafka
- Apache ZooKeeper
- Apache Kafka连接
- Apache Kafka MirrorMaker
- Strimzi Kafka桥

JVM配置选项可优化不同平台和体系结构的性能。Strimzi允许您配置其中一些选项。

JVM配置

使用该属性可以为运行组件的JVM配置受支持的选项。 `jvmOptions`

支持的JVM选项有助于优化不同平台和体系结构的性能。

配置JVM选项

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑在属性，，，，或资源。例如： `jvmOptions` `Kafka` `KafkaConnect` `KafkaConnectS2I` `KafkaMirrorMaker` `KafkaBridge`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    jvmOptions:
      "-Xmx": "8g"
      "-Xms": "8g"
    # ...
  zookeeper:
    # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.2.12。容器图片

Strimzi允许您配置将用于其组件的容器镜像。仅在需要使用其他容器注册表的特殊情况下才建议覆盖容器镜像。例如，由于您的网络不允许访问Strimzi使用的容器存储库。在这种情况下，您应该复制Strimzi图像或从源代码构建它们。如果配置的图像与Strimzi图像不兼容，则可能无法正常工作。

容器镜像配置

使用该属性指定要使用的容器图像。 `image`

警告	仅在特殊情况下才建议覆盖容器镜像。

配置容器镜像

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑在属性，或资源。例如： `image` `Kafka` `KafkaConnect` `KafkaConnectS2I`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    image: my-org/my-image:latest
    # ...
  zookeeper:
    # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.2.13。配置窗格调度

重要	当将两个应用程序安排到同一Kubernetes节点时，这两个应用程序可能会使用相同的资源（如磁盘I / O）并影响性能。这会导致性能下降。避免与其他关键工作负载共享节点，使用正确的节点或仅将一组节点专用于Kafka的方式来调度Kafka Pod是避免此类问题的最佳方法。

根据其他应用程序调度Pod

避免关键应用程序共享节点

可以使用Pod抗关联性来确保关键应用程序永远不会安排在同一磁盘上。在运行Kafka群集时，建议使用pod anti-affinity以确保Kafka代理不与其他工作负载（如数据库）共享节点。

亲和力

可以使用以下资源中的属性来配置亲和力： `affinity`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

相似性配置可以包括不同类型的相似性：

- Pod亲和力和反亲和力
- 节点亲和力

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes节点和pod关联文档。 `affinity`

在Kafka组件中配置Pod抗亲和力

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性以指定集群部署。使用标签指定不应在同一节点上安排的Pod。该应设置为指定所选荚不应该使用相同的主机节点进行调度。例如： `affinity` `topologyKey` `kubernetes.io/hostname`

```
apiVersion: kafka.strimzi.io/v1beta1
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。

   可以使用： `kubectl apply`

```
kubectl apply -f your-file
```

将Pod调度到特定节点

节点调度

Kubernetes集群通常由许多不同类型的工作程序节点组成。有些针对CPU繁重的工作负载进行了优化，有些针对内存进行了优化，而其他针对存储（快速本地SSD）或网络进行了优化。使用不同的节点有助于优化成本和性能。为了获得最佳性能，允许安排Strimzi组件使用正确的节点很重要。

Kubernetes使用节点关联性将工作负载调度到特定节点上。节点亲缘关系允许您为将在其上调度容器的节点创建调度约束。该约束被指定为标签选择器。您可以使用内置节点标签（例如）或自定义标签来指定标签，以选择正确的节点。 `beta.kubernetes.io/instance-type`

亲和力

可以使用以下资源中的属性来配置亲和力： `affinity`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

相似性配置可以包括不同类型的相似性：

- Pod亲和力和反亲和力
- 节点亲和力

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes节点和pod关联文档。 `affinity`

在Kafka组件中配置节点关联

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 标记应安排Strimzi组件的节点。

   可以使用： `kubectl label`

   ```
   kubectl label node your-node node-type=fast-network
   ```

   或者，某些现有标签可能会被重用。
2. 编辑资源中的属性以指定集群部署。例如： `affinity`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   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。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

使用专用节点

专用节点

群集管理员可以将选定的Kubernetes节点标记为已污染。带有污点的节点不包括在常规调度中，普通的吊舱也不会安排在它们上运行。只能在节点上安排可以容忍在节点上设置污点的服务。在此类节点上运行的唯一其他服务将是系统服务，例如日志收集器或软件定义的网络。

污渍可用于创建专用节点。在专用节点上运行Kafka及其组件可具有许多优势。在同一节点上不会运行其他任何可能引起干扰或消耗Kafka所需资源的应用程序。这可以提高性能和稳定性。

要在专用节点上安排Kafka Pod，请配置节点亲和力和容差。

亲和力

可以使用以下资源中的属性来配置亲和力： `affinity`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

相似性配置可以包括不同类型的相似性：

- Pod亲和力和反亲和力
- 节点亲和力

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes节点和pod关联文档。 `affinity`

公差范围

可以使用以下资源中的属性来配置公差： `tolerations`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`

- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes污点和公差。 `tolerations`

设置专用节点并在其上调度Pod

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 选择应用作专用节点。
2. 确保在这些节点上没有安排任何工作负载。
3. 在所选节点上设置污点：

   可以使用： `kubectl taint`

   ```
   kubectl taint node your-node dedicated=Kafka:NoSchedule
   ```

4. 此外，还向所选节点添加标签。

   可以使用： `kubectl label`

   ```
   kubectl label node your-node dedicated=Kafka
   ```

5. 编辑资源中的和属性以指定集群部署。例如： `affinity` `tolerations`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   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。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.2.14。使用外部配置和机密

使用Kafka Connect HTTP REST接口或使用创建，重新配置和删除连接器。有关这些方法的更多信息，请参阅*Deploying Strimzi*指南中的创建和管理连接器。连接器配置作为HTTP请求的一部分传递到Kafka Connect，并存储在Kafka本身中。 `KafkaConnectors`

ConfigMap和Secrets是用于存储配置和机密数据的标准Kubernetes资源。无论使用哪种方法来管理连接器，都可以使用ConfigMaps和Secrets来配置连接器的某些元素。然后，您可以在HTTP REST命令中引用配置值（如果需要，这可以使配置独立且更安全）。此方法尤其适用于机密数据，例如用户名，密码或证书。

从外部存储连接器配置

您可以将ConfigMap或Secrets作为卷或环境变量安装到Kafka Connect窗格中。在和中的属性中配置卷和环境变量。 `externalConfiguration` `KafkaConnect.spec` `KafkaConnectS2I.spec`

外部配置作为环境变量

该属性用于指定一个或多个环境变量。这些变量可以包含ConfigMap或Secret中的值。 `env`

注意	用户定义的环境变量的名称不能以或开头。 `KAFKA_` `STRIMZI_`

要将值从Secret装入到环境变量，请使用属性和下例所示。将环境变量设置为Secret中的值的示例 `valueFrom` `secretKeyRef`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  externalConfiguration:
    env:
      - name: MY_ENVIRONMENT_VARIABLE
        valueFrom:
          secretKeyRef:
            name: my-secret
            key: my-key
```

将Secrets装入环境变量的常见用例是连接器需要与Amazon AWS通信并需要使用凭证读取和环境变量时。 `AWS_ACCESS_KEY_ID` `AWS_SECRET_ACCESS_KEY`

要将值从ConfigMap挂载到环境变量，请在属性中使用，如以下示例所示。将环境变量设置为ConfigMap中的值的示例 `configMapKeyRef` `valueFrom`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  externalConfiguration:
    env:
      - name: MY_ENVIRONMENT_VARIABLE
        valueFrom:
          configMapKeyRef:
            name: my-config-map
            key: my-key
```

外部配置为卷

您还可以将ConfigMap或Secrets作为卷安装到Kafka Connect吊舱。在以下情况下，使用卷代替环境变量非常有用：

- 使用TLS证书挂载信任库或密钥库
- 挂载用于配置Kafka Connect连接器的属性文件

在资源的属性中，列出将作为卷挂载的ConfigMaps或Secrets。每个卷都必须在属性中指定一个名称以及对ConfigMap或Secret的引用。具有外部配置的卷示例 `volumes` `externalConfiguration` `name`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  externalConfiguration:
    volumes:
      - name: connector1
        configMap:
          name: connector1-configuration
      - name: connector1-certificates
        secret:
          secretName: connector1-certificates
```

这些卷将安装在该路径中的Kafka Connect容器内。例如，名为的卷中的文件将出现在directory中。 `/opt/kafka/external-configuration/<volume-name>` `connector1` `/opt/kafka/external-configuration/connector1`

在必须被用于读取在连接器结构从所安装的属性文件中的值。 `FileConfigProvider`

将Secrets挂载为环境变量

您可以创建一个Kubernetes Secret并将其作为环境变量安装到Kafka Connect。
先决条件

- 正在运行的集群 Operator 。

程序

1. 创建一个包含将作为环境变量装入的信息的机密。例如：

```
apiVersion: v1
kind: Secret
metadata:
  name: aws-creds
type: Opaque
data:
  awsAccessKey: QUtJQVhYWFhYWFhYWFhYWFg=
  awsSecretAccessKey: Ylhsd1lYTnpkMjl5WkE=
```

2. 创建或编辑Kafka Connect资源。配置资源或自定义资源的部分以引用机密。例如： `externalConfiguration` `KafkaConnect` `KafkaConnectS2I`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  externalConfiguration:
    env:
      - name: AWS_ACCESS_KEY_ID
        valueFrom:
          secretKeyRef:
            name: aws-creds
            key: awsAccessKey
      - name: AWS_SECRET_ACCESS_KEY
        valueFrom:
          secretKeyRef:
            name: aws-creds
            key: awsSecretAccessKey
```

3. 将更改应用于您的Kafka Connect部署。

   用途： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

现在，开发连接器时可以使用环境变量。
额外资源

- 有关Kafka Connect中外部配置的更多信息，请参阅模式参考。 `ExternalConfiguration`

将Secrets作为卷进行连接器配置

您可以创建一个Kubernetes Secret，将其作为卷安装到Kafka Connect，然后使用它来配置Kafka Connect连接器。

配置提供程序允许您从外部源加载配置值。

在此过程中，将名为的Secret 安装到名为的卷上。Kafka 的别名为，用于从文件读取和提取数据库*用户名*和*密码*属性值，以用于连接器配置。先决条件 `mysecret` `connector-config` `FileConfigProvider` `file`

- 正在运行的集群 Operator 。

程序

1. 创建一个包含用于定义连接器配置的配置选项的属性的机密。

   例如：

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: mysecret
   type: Opaque
   stringData:
     connector.properties: |- (1)
       dbUsername: my-user (2)
       dbPassword: my-password
   ```

   a. 属性文件格式的连接器配置。
   b. 配置中使用的数据库用户名和密码属性。
2. 创建或编辑Kafka Connect资源。

   在或自定义资源的部分和部分中，配置配置提供程序以引用机密。 `config` `externalConfiguration` `KafkaConnect` `KafkaConnectS2I`

   例如：

```
apiVersion: kafka.strimzi.io/v1beta1
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)
```

a. 配置提供程序的别名，用于定义其他配置参数。如果要添加多个提供程序，请使用逗号分隔的列表。
b. 的是配置提供商，其提供从属性文件的值。参数使用from的别名，格式为。 `FileConfigProvider` `config.providers` `config.providers.${alias}.class`
c. 包含机密的卷的名称。
d. 秘密的名称。

3. 将更改应用于您的Kafka Connect部署。

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

4. 配置您的连接器

- 如果您使用的是Kafka Connect HTTP REST接口，请使用带有连接器配置的JSON有效负载中的已安装属性文件中的值。例如：

```
{
   "name":"my-connector",
   "config":{
      "connector.class":"MyDbConnector",
      "tasks.max":"3",
      "database": "my-postgresql:5432",
      "username":"${file:/opt/kafka/external-configuration/connector-config/connector.properties:dbUsername}", (1)
      "password":"${file:/opt/kafka/external-configuration/connector-config/connector.properties:dbPassword}", (2)
      # ...
   }
}
```

a. 用户名属性值的路径。路径采用形式。 `/opt/kafka/external-configuration/SECRET-VOLUME-NAME/PROPERTIES-FILENAME:USERNAME-PROPERTY`
b. 密码属性值的路径。路径采用形式。 `/opt/kafka/external-configuration/SECRET-VOLUME-NAME/PROPERTIES-FILENAME:PASSWORD-PROPERTY`

- 如果使用的是资源，请使用自定义资源部分中已装入的属性文件中的值。 `KafkaConnector` `spec.config`

例如：

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnector
metadata:
  name: my-connector
  # ...
spec:
  class: "MyDbConnector"
  tasksMax: 3
  config:
    database: "my-postgresql:5432"
    username: "${file:/opt/kafka/external-configuration/connector-config/connector.properties:dbUsername}"
    password: "${file:/opt/kafka/external-configuration/connector-config/connector.properties:dbPassword}"
```

额外资源

- 有关Kafka Connect中外部配置的更多信息，请参阅模式参考。 `ExternalConfiguration`

2.2.15。启用资源 KafkaConnector

要启用Kafka Connect集群，请将注释添加到或自定义资源。先决条件 `KafkaConnectors` `strimzi.io/use-connector-resources` `KafkaConnect` `KafkaConnectS2I`

- 正在运行的集群 Operator

程序

1. 编辑或资源。添加注释。例如： `KafkaConnect` `KafkaConnectS2I` `strimzi.io/use-connector-resources`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  # ...
```

2. 使用以下命令创建或更新资源： `kubectl apply`

```
kubectl apply -f kafka-connect.yaml
```

额外资源

- 创建和管理连接器
- 将资源部署到Kafka Connect `KafkaConnector`
- `KafkaConnect` 模式参考
- `KafkaConnectS2I` 模式参考

2.2.16。作为Kafka Connect集群的一部分创建的资源列表

Kubernetes集群中的集群 Operator 将创建以下资源：

connect-cluster-name-connectDeployment负责创建Kafka Connect工作节点pods.connect-cluster-name-connect-apiService公开用于管理Kafka Connect集群的REST接口.connect-cluster-name-configConfigMap包含Kafka Connect辅助配置，并通过为Kafka Connect辅助节点配置的Kafka代理pods.connect-cluster-name-connectPod中断预算作为卷安装。

2.3。具有Source2Image支持的Kafka Connect集群配置

模式参考中描述了资源的完整模式。应用于所需资源的所有标签也将应用于构成具有Source2Image支持的Kafka Connect集群的Kubernetes资源。这提供了一种方便的机制，可以根据需要标记资源。 `KafkaConnectS2I` `KafkaConnectS2I` `KafkaConnectS2I`

2.3.1。复制品

Kafka Connect群集可以包含一个或多个节点。节点的数量在和资源中定义。运行具有多个节点的Kafka Connect群集可以提供更好的可用性和可伸缩性。但是，在Kubernetes上运行Kafka Connect时，不必为了高可用性而运行Kafka Connect的多个节点。如果将部署Kafka Connect的节点崩溃，Kubernetes将自动将Kafka Connect pod的时间表重新安排到其他节点。但是，与多个节点一起运行Kafka Connect可以提供更快的故障转移时间，因为其他节点已经启动并正在运行。 `KafkaConnect` `KafkaConnectS2I`

配置节点数

Kafka连接的节点的数目是使用配置在属性和。先决条件 `replicas` `KafkaConnect.spec` `KafkaConnectS2I.spec`

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 在或资源中编辑属性。例如： `replicas` `KafkaConnect` `KafkaConnectS2I`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnectS2I
metadata:
  name: my-cluster
spec:
  # ...
  replicas: 3
  # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

```
kubectl apply -f your-file
```

2.3.2。引导服务器

Kafka Connect群集始终与Kafka群集结合使用。Kafka群集被指定为引导服务器列表。在Kubernetes上，理想情况下，该列表必须包含名为的Kafka集群引导服务以及端口9092（用于普通流量）或9093（用于加密流量）。 *`cluster-name`*`-kafka-bootstrap`

引导服务器列表在和中的属性中配置。必须将服务器定义为指定一个或多个Kafka代理的逗号分隔列表，或者将服务指定为成对指向Kafka代理的服务。 `bootstrapServers` `KafkaConnect.spec` `KafkaConnectS2I.spec` *`hostname`*`:_port_`

将Kafka Connect与不受Strimzi管理的Kafka群集一起使用时，可以根据群集的配置指定引导服务器列表。

配置引导服务器

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 在或资源中编辑属性。例如： `bootstrapServers` `KafkaConnect` `KafkaConnectS2I`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-cluster
spec:
  # ...
  bootstrapServers: my-cluster-kafka-bootstrap:9092
  # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

```
kubectl apply -f your-file
```

2.3.3。使用TLS连接到Kafka经纪人

默认情况下，Kafka Connect尝试使用纯文本连接连接到Kafka代理。如果您更喜欢使用TLS，则需要其他配置。

Kafka Connect中的TLS支持

TLS支持在和中的属性中配置。该属性包含密钥名称的机密列表，用于存储证书。证书必须以X509格式存储。显示带有多个证书的TLS配置的示例 `tls` `KafkaConnect.spec` `KafkaConnectS2I.spec` `tls`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-cluster
spec:
  # ...
  tls:
    trustedCertificates:
      - secretName: my-secret
        certificate: ca.crt
      - secretName: my-other-secret
        certificate: certificate.crt
  # ...
```

当多个证书存储在同一秘密中时，可以多次列出。
显示带有来自同一机密的多个证书的TLS配置的示例

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnectS2I
metadata:
  name: my-cluster
spec:
  # ...
  tls:
    trustedCertificates:
      - secretName: my-secret
        certificate: ca.crt
      - secretName: my-secret
        certificate: ca2.crt
  # ...
```

在Kafka Connect中配置TLS

先决条件

- Kubernetes集群
- 正在运行的集群 Operator
- 如果存在，则用于TLS服务器身份验证的证书的名称，以及用于将证书存储在其中的密钥。 `Secret` `Secret`

程序

1. （可选）如果尚不存在它们，请在文件中准备用于身份验证的TLS证书，并创建一个。 `Secret`

注意	由集群 Operator 为Kafka集群创建的机密可以直接使用。

可以使用： `kubectl create`

```
kubectl create secret generic my-secret --from-file=my-file.crt
```

2. 在或资源中编辑属性。例如： `tls` `KafkaConnect` `KafkaConnectS2I`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  tls:
    trustedCertificates:
      - secretName: my-cluster-cluster-cert
        certificate: ca.crt
  # ...
```

3. 创建或更新资源。

可以使用： `kubectl apply`

```
kubectl apply -f your-file
```

2.3.4。通过身份验证连接到Kafka经纪人

默认情况下，Kafka Connect将尝试在不进行身份验证的情况下连接到Kafka代理。通过和资源启用身份验证。 `KafkaConnect` `KafkaConnectS2I`

Kafka Connect中的身份验证支持

认证是通过配置在属性和。该属性指定应使用的身份验证机制的类型以及取决于该机制的其他配置详细信息。支持的身份验证类型为： `authentication` `KafkaConnect.spec` `KafkaConnectS2I.spec` `authentication`

- TLS客户端身份验证
- 使用SCRAM-SHA-512机制的基于SASL的身份验证
- 使用PLAIN机制的基于SASL的身份验证
- 基于OAuth 2.0令牌的身份验证

TLS客户端身份验证

要使用TLS客户端身份验证，请将属性设置为值。TLS客户端身份验证使用TLS证书进行身份验证。该证书在属性中指定，并且始终从Kubernetes机密中加载。秘密中，证书必须以X509格式存储在两个不同的密钥下：公用密钥和专用密钥。 `type` `tls` `certificateAndKey`

注意	TLS客户端身份验证只能与TLS连接一起使用。有关Kafka Connect中TLS配置的更多详细信息，请参阅使用TLS连接到Kafka代理。

TLS客户端身份验证配置示例

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-cluster
spec:
  # ...
  authentication:
    type: tls
    certificateAndKey:
      secretName: my-secret
      certificate: public.crt
      key: private.key
  # ...
```

基于SASL的SCRAM-SHA-512身份验证

要将Kafka Connect配置为使用基于SASL的SCRAM-SHA-512身份验证，请将属性设置为。此身份验证机制需要用户名和密码。 `type` `scram-sha-512`

- 在属性中指定用户名。 `username`
- 在属性中，指定指向包含密码的链接。该属性包含的名称和属性包含其下的密码存储在里面的键的名称。 `passwordSecret` `Secret` `secretName` `Secret` `password` `Secret`

重要	不要在字段中指定实际密码。 `password`

基于SASL的示例SCRAM-SHA-512客户端身份验证配置

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-cluster
spec:
  # ...
  authentication:
    type: scram-sha-512
    username: my-connect-user
    passwordSecret:
      secretName: my-connect-user
      password: my-connect-password-key
  # ...
```

基于SASL的PLAIN身份验证

要将Kafka Connect配置为使用基于SASL的PLAIN身份验证，请将属性设置为。此身份验证机制需要用户名和密码。 `type` `plain`

警告	SASL PLAIN机制将以明文形式在网络上传输用户名和密码。如果启用了TLS加密，则仅使用SASL PLAIN身份验证。

- 在属性中指定用户名。 `username`
- 在属性中，指定指向包含密码的链接。该属性包含这样的名称，该属性包含密钥的名称，在该密钥下密码存储在内。 `passwordSecret` `Secret` `secretName` `Secret` `password` `Secret`

重要	不要在字段中指定实际密码。 `password`

显示基于SASL的PLAIN客户端身份验证配置的示例

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-cluster
spec:
  # ...
  authentication:
    type: plain
    username: my-connect-user
    passwordSecret:
      secretName: my-connect-user
      password: my-connect-password-key
  # ...
```

在Kafka Connect中配置TLS客户端身份验证

先决条件

- Kubernetes集群
- 正在运行的集群 Operator
- 如果存在，则带有用于TLS客户端身份验证的公钥和私钥的的名称，以及将其存储在 `Secret` `Secret`

程序

1. （可选）如果尚不存在，请在文件中准备用于身份验证的密钥，然后创建。 `Secret`

注意	可以使用由用户 Operator 创建的机密。

   可以使用： `kubectl create`

   ```
   kubectl create secret generic my-secret --from-file=my-public.crt --from-file=my-private.key
   ```

2. 在或资源中编辑属性。例如： `authentication` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: KafkaConnect
   metadata:
     name: my-connect
   spec:
     # ...
     authentication:
       type: tls
       certificateAndKey:
         secretName: my-secret
         certificate: my-public.crt
         key: my-private.key
     # ...
   ```

3. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

在Kafka Connect中配置SCRAM-SHA-512身份验证

先决条件

- Kubernetes集群
- 正在运行的集群 Operator
- 用于验证的用户的用户名
- 如果存在，请输入的名称以及用于身份验证的密码，以及用于存储密码的密钥 `Secret` `Secret`

程序

1. （可选）如果尚不存在，请准备一个文件，其中包含用于身份验证的密码并创建。 `Secret`

注意	可以使用由用户 Operator 创建的机密。

   可以使用： `kubectl create`

   ```
   echo -n 'PASSWORD' > MY-PASSWORD.txt
   kubectl create secret generic MY-SECRET --from-file=MY-PASSWORD.txt
   ```

2. 在或资源中编辑属性。例如： `authentication` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: KafkaConnect
   metadata:
     name: my-connect
   spec:
     # ...
     authentication:
       type: scram-sha-512
       username: MY-USERNAME
       passwordSecret:
         secretName: MY-SECRET
         password: MY-PASSWORD.txt
     # ...
   ```

3. 创建或更新资源。

   在Kubernetes上可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.3.5。Kafka Connect配置

Strimzi允许您通过编辑Apache Kafka文档中列出的某些选项来自定义Apache Kafka Connect节点的配置。

无法配置的配置选项涉及：

- Kafka集群引导地址
- 安全性（加密，认证和授权）
- 侦听器/ REST接口配置
- 插件路径配置

这些选项由Strimzi自动配置。

Kafka Connect配置

Kafka连接是使用配置在属性和。该属性包含Kafka Connect配置选项作为键。值可以是以下JSON类型之一： `config` `KafkaConnect.spec` `KafkaConnectS2I.spec`

- 串
- 数
- 布尔型

您可以指定和配置Apache Kafka文档中列出的选项，但由Strimzi直接管理的选项除外。特别是，禁止使用键等于或以下列字符串之一开头的配置选项：

- `ssl.`
- `sasl.`
- `security.`
- `listeners`
- `plugin.path`
- `rest.`
- `bootstrap.servers`

当属性中存在禁止选项时，它将被忽略，并且警告消息将打印到Cluster Operator日志文件中。所有其他选项都传递给Kafka Connect。 config

重要	群集 operator 不验证提供的对象中的键或值。提供无效的配置时，Kafka Connect群集可能无法启动或变得不稳定。在这种情况下，请在或对象中修复配置，然后集群 Operator 可以将新配置推广到所有Kafka Connect节点。 config KafkaConnect.spec.config KafkaConnectS2I .spec.config

某些选项具有默认值：

- group.id 具有默认值 connect-cluster
- offset.storage.topic 具有默认值 connect-cluster-offsets
- config.storage.topic 具有默认值 connect-cluster-configs
- status.storage.topic 具有默认值 connect-cluster-status
- key.converter 具有默认值 org.apache.kafka.connect.json.JsonConverter
- value.converter 具有默认值 org.apache.kafka.connect.json.JsonConverter

如果或属性中不存在这些选项，则会自动进行配置。 KafkaConnect.spec.config KafkaConnectS2I.spec.config

禁止选项也有例外。对于特定于TLS版本的*密码套件*，可以使用三个允许的配置选项进行客户端连接。密码套件结合了用于安全连接和数据传输的算法。您还可以配置该属性以启用或禁用主机名验证。Kafka Connect示例配置 ssl ssl.endpoint.identification.algorithm

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    group.id: my-connect-cluster
    offset.storage.topic: my-connect-cluster-offsets
    config.storage.topic: my-connect-cluster-configs
    status.storage.topic: my-connect-cluster-status
    key.converter: org.apache.kafka.connect.json.JsonConverter
    value.converter: org.apache.kafka.connect.json.JsonConverter
    key.converter.schemas.enable: true
    value.converter.schemas.enable: true
    config.storage.replication.factor: 3
    offset.storage.replication.factor: 3
    status.storage.replication.factor: 3
    ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" (1)
    ssl.enabled.protocols: "TLSv1.2" (2)
    ssl.protocol: "TLSv1.2" (3)
    ssl.endpoint.identification.algorithm: HTTPS (4)
  # ...
```

1. TLS的密码套件，使用密钥交换机制，认证算法，批量加密算法和MAC算法的组合。 ECDHE RSA AES SHA384
2. 启用SSl协议。 TLSv1.2
3. 指定生成SSL上下文的协议。允许的值为和。 TLSv1.2 TLSv1.1 TLSv1.2
4. 通过设置为启用主机名验证。空字符串将禁用验证。 HTTPS

多个实例的Kafka Connect配置

如果您正在运行Kafka Connect的多个实例，则必须更改以下属性的默认配置： config

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    group.id: connect-cluster (1)
    offset.storage.topic: connect-cluster-offsets (2)
    config.storage.topic: connect-cluster-configs (3)
    status.storage.topic: connect-cluster-status  (4)
    # ...
# ...
```

1. 实例所属的Kafka Connect群集组。
2. 存储连接器偏移量的Kafka Topic 。
3. 存储连接器和任务状态配置的Kafka Topic 。
4. 存储连接器和任务状态更新的Kafka Topic 。

注意	对于所有具有相同的Kafka Connect实例，这三个 Topic 的值必须相同。 group.id

除非您更改默认设置，否则连接到同一Kafka群集的每个Kafka Connect实例将使用相同的值进行部署。实际上，发生的事情是所有实例都耦合在一起在集群中运行并使用相同的 Topic 。

如果多个Kafka Connect群集尝试使用相同的 Topic ，则Kafka Connect将无法按预期工作并产生错误。

如果您希望运行多个Kafka Connect实例，请为每个实例更改这些属性的值。

配置Kafka Connect

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 在或资源中编辑属性。例如： `config` `KafkaConnect` `KafkaConnectS2I`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    group.id: my-connect-cluster
    offset.storage.topic: my-connect-cluster-offsets
    config.storage.topic: my-connect-cluster-configs
    status.storage.topic: my-connect-cluster-status
    key.converter: org.apache.kafka.connect.json.JsonConverter
    value.converter: org.apache.kafka.connect.json.JsonConverter
    key.converter.schemas.enable: true
    value.converter.schemas.enable: true
    config.storage.replication.factor: 3
    offset.storage.replication.factor: 3
    status.storage.replication.factor: 3
  # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

3. 如果为Kafka Connect启用了授权，请配置Kafka Connect用户以启用对Kafka Connect使用者组和 Topic 的访问。

2.3.6。Kafka Connect用户授权

如果为Kafka Connect启用了授权，则必须配置Kafka Connect用户以提供对Kafka Connect使用者组和内部 Topic 的读/写访问权限。

消费者组和内部 Topic 的属性由Strimzi自动配置，或者可以在for 或配置中显式指定。 `spec` `KafkaConnect` `KafkaConnectS2I`

以下示例显示了资源中属性的配置，需要在Kafka Connect用户配置中表示。 `KafkaConnect`

```
apiVersion: kafka.strimzi.io/v1beta1
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. 实例所属的Kafka Connect群集组。
2. 存储连接器偏移量的Kafka Topic 。
3. 存储连接器和任务状态配置的Kafka Topic 。
4. 存储连接器和任务状态更新的Kafka Topic 。

配置Kafka Connect用户授权

此过程描述了如何授权用户访问Kafka Connect。

在Kafka中使用任何类型的授权时，Kafka Connect用户都需要对使用者组的访问权限以及Kafka Connect的内部 Topic 。

此过程显示使用授权时如何提供访问。 `simple`

简单授权使用由Kafka 插件处理的ACL规则来提供正确的访问级别。有关配置资源以使用简单授权的更多信息，请参见模式参考。 `AclAuthorizer` `KafkaUser` `AclRule`

注意	运行多个实例时，使用者组和 Topic 的默认值将有所不同。

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性以向用户提供访问权限。 `authorization` `KafkaUser`

   在以下示例中，使用名称值为Kafka Connect Topic 和使用者组配置了访问权限： `literal`

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

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: KafkaUser
   metadata:
     name: my-user
     labels:
       strimzi.io/cluster: my-cluster
   spec:
     # ...
     authorization:
       type: simple
       acls:
         # access to offset.storage.topic
         - resource:
             type: topic
             name: connect-cluster-offsets
             patternType: literal
           operation: Write
           host: "*"
         - resource:
             type: topic
             name: connect-cluster-offsets
             patternType: literal
           operation: Create
           host: "*"
         - resource:
             type: topic
             name: connect-cluster-offsets
             patternType: literal
           operation: Describe
           host: "*"
         - resource:
             type: topic
             name: connect-cluster-offsets
             patternType: literal
           operation: Read
           host: "*"
         # access to status.storage.topic
         - resource:
             type: topic
             name: connect-cluster-status
             patternType: literal
           operation: Write
           host: "*"
         - resource:
             type: topic
             name: connect-cluster-status
             patternType: literal
           operation: Create
           host: "*"
         - resource:
             type: topic
             name: connect-cluster-status
             patternType: literal
           operation: Describe
           host: "*"
         - resource:
   ```

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

2. 创建或更新资源。

```
kubectl apply -f your-file
```

2.3.7。CPU和内存资源

对于每个部署的容器，Strimzi允许您请求特定资源并定义这些资源的最大消耗量。

Strimzi支持两种类型的资源：

- 中央处理器
- 记忆

Strimzi使用Kubernetes语法指定CPU和内存资源。

资源限制和要求

使用以下资源中的属性配置资源限制和请求： `resources`

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `Kafka.spec.entityOperator.topicOperator`
- `Kafka.spec.entityOperator.userOperator`
- `Kafka.spec.entityOperator.tlsSidecar`
- `Kafka.spec.KafkaExporter`
- `KafkaConnect.spec`
- `KafkaConnectS2I.spec`
- `KafkaBridge.spec`

额外资源

- 有关在Kubernetes上管理计算资源的更多信息，请参阅管理容器的计算资源。

资源要求

请求指定要为给定容器保留的资源。保留资源可确保它们始终可用。

重要	如果资源请求所需要的资源超出了Kubernetes集群中的可用空闲资源，则未安排Pod。

资源请求在属性中指定。Strimzi当前支持的资源请求： `requests`

- `cpu`

- memory

可以为一个或多个支持的资源配置请求。
所有资源的示例资源请求配置

```
# ...
resources:
  requests:
    cpu: 12
    memory: 64Gi
# ...
```

资源限制

限制指定给定容器可以消耗的最大资源。该限制不是保留的，可能并不总是可用。容器只有在可用时才能使用资源。资源限制应始终高于资源请求。

资源限制在属性中指定。Strimzi当前支持的资源限制： `limits`

- cpu
- memory

可以为一个或多个受支持的限制配置资源。
示例资源限制配置

```
# ...
resources:
  limits:
    cpu: 12
    memory: 64Gi
# ...
```

支持的CPU格式

支持以下格式的CPU请求和限制：

- CPU内核数，可以是整数（`5` CPU内核）或十进制（`2.5` CPU内核）。
- 数字或*毫微克* / *毫厘*（），其中1000 *毫欧*是同一CPU核心。 `100m` `1`

示例CPU单元

```
# ...
resources:
  requests:
    cpu: 500m
  limits:
    cpu: 2.5
# ...
```

注意	1个CPU内核的计算能力可能会因所部署的Kubernetes平台而异。

额外资源

- 有关CPU规格的更多信息，请参见CPU的含义。

支持的内存格式

内存请求和限制以兆字节，千兆字节，兆字节和千兆字节指定。

- 要指定兆字节的内存，请使用后缀。例如。 `M` `1000M`
- 要以GB为单位指定内存，请使用后缀。例如。 `G` `1G`
- 要以兆字节为单位指定内存，请使用后缀。例如。 `Mi` `1000Mi`
- 要以千兆字节指定内存，请使用后缀。例如。 `Gi` `1Gi`

使用不同存储单元的示例

```
# ...
resources:
  requests:
    memory: 512Mi
  limits:
    memory: 2Gi
# ...
```

额外资源

- 有关内存规格和其他受支持单位的更多详细信息，请参阅内存的含义。

配置资源请求和限制

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性以指定集群部署。例如： `resources`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  kafka:
    # ...
    resources:
      requests:
        cpu: "8"
        memory: 64Gi
      limits:
        cpu: "12"
        memory: 128Gi
    # ...
  zookeeper:
    # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

额外资源

- 有关架构的更多信息，请参见ResourceRequirements API参考。

2.3.8。Kafka Connect与S2I记录器

Kafka Connect with Source2Image支持具有其自己的可配置记录器：

- `log4j.rootLogger`
- `log4j.logger.org.reflections`

Kafka Connect使用Apache 记录器实现。 `log4j`

使用该属性来配置记录器和记录器级别。 `logging`

您可以通过指定记录器和直接（内联）级别或使用自定义（外部）ConfigMap来设置日志级别。如果使用ConfigMap，则将属性设置为包含外部日志记录配置的ConfigMap的名称。在ConfigMap内部，使用来描述日志记录配置。 `logging.name` `log4j.properties`

在这里，我们看到和的示例。内联记录 `inline` `external`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnectS2I
spec:
  # ...
  logging:
    type: inline
    loggers:
      log4j.rootLogger: "INFO"
  # ...
```

外部记录

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnectS2I
spec:
  # ...
  logging:
    type: external
    name: customConfigMap
  # ...
```

额外资源

- 垃圾收集器（GC）日志记录也可以启用（或禁用）。有关GC日志记录的更多信息，请参阅JVM配置。
- 有关日志级别的更多信息，请参阅Apache日志服务。

2.3.9。健康检查

运行状况检查是定期测试，可验证应用程序的运行状况。当运行状况检查探针失败时，Kubernetes会认为该应用程序运行状况不佳，并尝试对其进行修复。

Kubernetes支持两种类型的Healthcheck探针：

- 活力探针
- 准备就绪探针

有关探针的更多详细信息，请参阅“ 配置活动性和就绪性探针”。两种类型的探针都用于Strimzi组件中。

用户可以为活动和就绪探针配置选定的选项。

健康检查配置

可以使用以下资源中的和属性来配置活动性和就绪性探针： `livenessProbe` `readinessProbe`

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `Kafka.spec.entityOperator.tlsSidecar`
- `Kafka.spec.entityOperator.topicOperator`
- `Kafka.spec.entityOperator.userOperator`
- `Kafka.spec.KafkaExporter`
- `KafkaConnect.spec`
- `KafkaConnectS2I.spec`
- `KafkaMirrorMaker.spec`
- `KafkaBridge.spec`

双方并支持以下选项： `livenessProbe` `readinessProbe`

- `initialDelaySeconds`
- `timeoutSeconds`
- `periodSeconds`
- `successThreshold`
- `failureThreshold`

有关和选项的更多信息，请参见模式参考。活动和就绪探针配置示例 `livenessProbe` `readinessProbe` `Probe`

```
# ...
readinessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
# ...
```

配置健康检查

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑或财产的，或资源。例如： `livenessProbe` `readinessProbe` `Kafka` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
       readinessProbe:
         initialDelaySeconds: 15
         timeoutSeconds: 5
       livenessProbe:
         initialDelaySeconds: 15
         timeoutSeconds: 5
       # ...
     zookeeper:
       # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.3.10。普罗米修斯指标

Strimzi使用Prometheus JMX导出器支持Prometheus指标，以将Apache Kafka和ZooKeeper支持的JMX指标转换为Prometheus指标。启用度量后，它们将在端口9404上公开。

有关设置和部署Prometheus和Grafana的更多信息，请参阅*Deploying Strimzi*指南中的向Kafka引入度量。

指标配置

通过在以下资源中配置属性来启用Prometheus指标： `metrics`

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `KafkaConnect.spec`
- `KafkaConnectS2I.spec`

当资源中未定义属性时，将禁用Prometheus指标。要在不进行任何其他配置的情况下启用Prometheus指标导出，您可以将其设置为空对象（）。无需进一步配置即可启用指标的示例 `metrics` `{}`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    metrics: {}
    # ...
  zookeeper:
    # ...
```

该属性可能包含Prometheus JMX导出器的其他配置。使用其他Prometheus JMX Exporter配置启用指标的示例 `metrics`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    metrics:
      lowercaseOutputName: true
      rules:
        - pattern: "kafka.server<type=(.+), name=(.+)PerSec\\w*><>Count"
          name: "kafka_server_$1_$2_total"
        - pattern: "kafka.server<type=(.+), name=(.+)PerSec\\w*, topic=(.+)><>Count"
          name: "kafka_server_$1_$2_total"
          labels:
            topic: "$3"
    # ...
  zookeeper:
    # ...
```

配置Prometheus指标

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑在属性，或资源。例如： `metrics` `Kafka` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
     zookeeper:
       # ...
       metrics:
         lowercaseOutputName: true
       # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

```
kubectl apply -f your-file
```

2.3.11。JVM选项

Strimzi的以下组件在虚拟机（VM）中运行：

- 阿帕奇•Kafka
- Apache ZooKeeper
- Apache Kafka连接
- Apache Kafka MirrorMaker
- Strimzi Kafka桥

JVM配置选项可优化不同平台和体系结构的性能。Strimzi允许您配置其中一些选项。

JVM配置

使用该属性可以为运行组件的JVM配置受支持的选项。 `jvmOptions`

支持的JVM选项有助于优化不同平台和体系结构的性能。

配置JVM选项

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑在属性，，，，或资源。例如： `jvmOptions` `Kafka` `KafkaConnect` `KafkaConnectS2I` `KafkaMirrorMaker` `KafkaBridge`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
       jvmOptions:
         "-Xmx": "8g"
         "-Xms": "8g"
       # ...
     zookeeper:
       # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.3.12。容器图片

Strimzi允许您配置将用于其组件的容器镜像。仅在需要使用其他容器注册表的特殊情况下才建议覆盖容器镜像。例如，由于您的网络不允许访问Strimzi使用的容器存储库。在这种情况下，您应该复制Strimzi图像或从源代码构建它们。如果配置的图像与Strimzi图像不兼容，则可能无法正常工作。

容器镜像配置

使用该属性指定要使用的容器图像。 `image`

警告	仅在特殊情况下才建议覆盖容器镜像。

配置容器镜像

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑在属性，或资源。例如： `image` `Kafka` `KafkaConnect` `KafkaConnectS2I`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    image: my-org/my-image:latest
    # ...
  zookeeper:
    # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.3.13。配置窗格调度

重要	当将两个应用程序安排到同一Kubernetes节点时，这两个应用程序可能会使用相同的资源（如磁盘I / O）并影响性能。这会导致性能下降。避免与其他关键工作负载共享节点，使用正确的节点或仅将一组节点专用于Kafka的方式来调度Kafka Pod是避免此类问题的最佳方法。

根据其他应用程序调度Pod

避免关键应用程序共享节点

可以使用Pod抗关联性来确保关键应用程序永远不会安排在同一磁盘上。在运行Kafka群集时，建议使用pod anti-affinity以确保Kafka代理不与其他工作负载（如数据库）共享节点。

亲和力

可以使用以下资源中的属性来配置亲和力： `affinity`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

相似性配置可以包括不同类型的相似性：

- Pod亲和力和反亲和力
- 节点亲和力

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes节点和pod关联文档。 `affinity`

在Kafka组件中配置Pod抗亲和力

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性以指定集群部署。使用标签指定不应在同一节点上安排的Pod。该应设置为指定所选荚不应该使用相同的主机节点进行调度。例如： `affinity` `topologyKey` `kubernetes.io/hostname`

```
apiVersion: kafka.strimzi.io/v1beta1
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。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

将Pod调度到特定节点

节点调度

Kubernetes集群通常由许多不同类型的工作程序节点组成。有些针对CPU繁重的工作负载进行了优化，有些针对内存进行了优化，而其他针对存储（快速本地SSD）或网络进行了优化。使用不同的节点有助于优化成本和性能。为了获得最佳性能，允许安排Strimzi组件使用正确的节点很重要。

Kubernetes使用节点关联性将工作负载调度到特定节点上。节点亲缘关系允许您为将在其上调度容器的节点创建调度约束。该约束被指定为标签选择器。您可以使用内置节点标签（例如）或自定义标签来指定标签，以选择正确的节点。 `beta.kubernetes.io/instance-type`

亲和力

可以使用以下资源中的属性来配置亲和力： `affinity`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

相似性配置可以包括不同类型的相似性：

- Pod亲和力和反亲和力
- 节点亲和力

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes节点和pod关联文档。 `affinity`

在Kafka组件中配置节点关联

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 标记应安排Strimzi组件的节点。

   可以使用： `kubectl label`

   ```
   kubectl label node your-node node-type=fast-network
   ```

   或者，某些现有标签可能会被重用。
2. 编辑资源中的属性以指定集群部署。例如： `affinity`

```
apiVersion: kafka.strimzi.io/v1beta1
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。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

使用专用节点

专用节点

群集管理员可以将选定的Kubernetes节点标记为已污染。带有污点的节点不包括在常规调度中，普通的吊舱也不会安排在它们上运行。只能在节点上安排可以容忍在节点上设置污点的服务。在此类节点上运行的唯一其他服务将是系统服务，例如日志收集器或软件定义的网络。

污渍可用于创建专用节点。在专用节点上运行Kafka及其组件可具有许多优势。在同一节点上不会运行其他任何可能引起干扰或消耗Kafka所需资源的应用程序。这可以提高性能和稳定性。

要在专用节点上安排Kafka Pod，请配置节点亲和力和容差。

亲和力

可以使用以下资源中的属性来配置亲和力： `affinity`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

相似性配置可以包括不同类型的相似性：

- Pod亲和力和反亲和力
- 节点亲和力

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes节点和pod关联文档。 `affinity`

公差范围

可以使用以下资源中的属性来配置公差： `tolerations`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes污点和公差。 `tolerations`

设置专用节点并在其上调度Pod

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 选择应用作专用节点。

2. 确保在这些节点上没有安排任何工作负载。
3. 在所选节点上设置污点：

   可以使用： `kubectl taint`

   ```
   kubectl taint node your-node dedicated=Kafka:NoSchedule
   ```

4. 此外，还向所选节点添加标签。

   可以使用： `kubectl label`

   ```
   kubectl label node your-node dedicated=Kafka
   ```

5. 编辑资源中的和属性以指定集群部署。例如： `affinity` `tolerations`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   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。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.3.14。使用外部配置和机密

使用Kafka Connect HTTP REST接口或使用创建，重新配置和删除连接器。有关这些方法的更多信息，请参阅*Deploying Strimzi*指南中的创建和管理连接器。连接器配置作为HTTP请求的一部分传递到Kafka Connect，并存储在Kafka本身中。 `KafkaConnectors`

ConfigMap和Secrets是用于存储配置和机密数据的标准Kubernetes资源。无论使用哪种方法来管理连接器，都可以使用ConfigMaps和Secrets来配置连接器的某些元素。然后，您可以在HTTP REST命令中引用配置值（如果需要，这可以使配置独立且更安全）。此方法尤其适用于机密数据，例如用户名，密码或证书。

从外部存储连接器配置

您可以将ConfigMap或Secrets作为卷或环境变量安装到Kafka Connect窗格中。在和中的属性中配置卷和环境变量。 `externalConfiguration` `KafkaConnect.spec` `KafkaConnectS2I.spec`

外部配置作为环境变量

该属性用于指定一个或多个环境变量。这些变量可以包含ConfigMap或Secret中的值。 `env`

注意	用户定义的环境变量的名称不能以或开头。 `KAFKA_` `STRIMZI_`

要将值从Secret装入到环境变量，请使用属性和下例所示。将环境变量设置为Secret中的值的示例 `valueFrom` `secretKeyRef`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  externalConfiguration:
    env:
      - name: MY_ENVIRONMENT_VARIABLE
        valueFrom:
          secretKeyRef:
            name: my-secret
            key: my-key
```

将Secrets装入环境变量的常见用例是连接器需要与Amazon AWS通信并需要使用凭证读取和环境变量时。 `AWS_ACCESS_KEY_ID` `AWS_SECRET_ACCESS_KEY`

要将值从ConfigMap挂载到环境变量，请在属性中使用，如以下示例所示。将环境变量设置为ConfigMap中的值的示例 `configMapKeyRef` `valueFrom`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  externalConfiguration:
    env:
      - name: MY_ENVIRONMENT_VARIABLE
        valueFrom:
          configMapKeyRef:
            name: my-config-map
            key: my-key
```

外部配置为卷

您还可以将ConfigMap或Secrets作为卷安装到Kafka Connect吊舱。在以下情况下，使用卷代替环境变量非常有用：

- 使用TLS证书挂载信任库或密钥库
- 挂载用于配置Kafka Connect连接器的属性文件

在资源的属性中，列出将作为卷挂载的ConfigMaps或Secrets。每个卷都必须在属性中指定一个名称以及对ConfigMap或Secret的引用。具有外部配置的卷示例 `volumes` `externalConfiguration` `name`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  externalConfiguration:
    volumes:
      - name: connector1
        configMap:
          name: connector1-configuration
      - name: connector1-certificates
        secret:
          secretName: connector1-certificates
```

这些卷将安装在该路径中的Kafka Connect容器内。例如，名为的卷中的文件将出现在directory中。 `/opt/kafka/external-configuration/<volume-name>` `connector1` `/opt/kafka/external-configuration/connector1`

在必须被用于读取在连接器结构从所安装的属性文件中的值。 `FileConfigProvider`

将Secrets挂载为环境变量

您可以创建一个Kubernetes Secret并将其作为环境变量安装到Kafka Connect。
先决条件

- 正在运行的集群 Operator 。

程序

1. 创建一个包含将作为环境变量装入的信息的机密。例如：

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: aws-creds
   type: Opaque
   data:
     awsAccessKey: QUtJQVhYWFhYWFhYWFhYWFg=
     awsSecretAccessKey: Ylhsd1lYTnpkMjl5WkE=
   ```

2. 创建或编辑Kafka Connect资源。配置资源或自定义资源的部分以引用机密。例如： `externalConfiguration` `KafkaConnect` `KafkaConnectS2I`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  externalConfiguration:
    env:
      - name: AWS_ACCESS_KEY_ID
        valueFrom:
          secretKeyRef:
            name: aws-creds
            key: awsAccessKey
      - name: AWS_SECRET_ACCESS_KEY
        valueFrom:
          secretKeyRef:
            name: aws-creds
            key: awsSecretAccessKey
```

3. 将更改应用于您的Kafka Connect部署。

   用途： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

现在，开发连接器时可以使用环境变量。
额外资源

- 有关Kafka Connect中外部配置的更多信息，请参阅模式参考。 `ExternalConfiguration`

将Secrets作为卷进行连接器配置

您可以创建一个Kubernetes Secret，将其作为卷安装到Kafka Connect，然后使用它来配置Kafka Connect连接器。

配置提供程序允许您从外部源加载配置值。

在此过程中，将名为的Secret 安装到名为的卷上。Kafka 的别名为，用于从文件读取和提取数据库*用户名*和*密码*属性值，以用于连接器配置。先决条件
`mysecret` `connector-config` `FileConfigProvider` `file`

- 正在运行的集群 Operator 。

程序

1. 创建一个包含用于定义连接器配置的配置选项的属性的机密。

   例如：

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: mysecret
   type: Opaque
   stringData:
     connector.properties: |- (1)
       dbUsername: my-user (2)
       dbPassword: my-password
   ```

   a. 属性文件格式的连接器配置。
   b. 配置中使用的数据库用户名和密码属性。
2. 创建或编辑Kafka Connect资源。

   在或自定义资源的部分和部分中，配置配置提供程序以引用机密。 `config` `externalConfiguration` `KafkaConnect` `KafkaConnectS2I`

   例如：

```
apiVersion: kafka.strimzi.io/v1beta1
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)
```

a. 配置提供程序的别名，用于定义其他配置参数。如果要添加多个提供程序，请使用逗号分隔的列表。
b. 的是配置提供商，其提供从属性文件的值。参数使用from的别名，格式为。 `FileConfigProvider` `config.providers` `config.providers.${alias}.class`
c. 包含机密的卷的名称。
d. 秘密的名称。

3. 将更改应用于您的Kafka Connect部署。

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

4. 配置您的连接器

- 如果您使用的是Kafka Connect HTTP REST接口，请使用带有连接器配置的JSON有效负载中的已安装属性文件中的值。例如：

```
{
   "name":"my-connector",
   "config":{
      "connector.class":"MyDbConnector",
      "tasks.max":"3",
      "database": "my-postgresql:5432",
      "username":"${file:/opt/kafka/external-configuration/connector-config/connector.properties:dbUsername}", (1)
      "password":"${file:/opt/kafka/external-configuration/connector-config/connector.properties:dbPassword}", (2)
      # ...
   }
}
```

a. 用户名属性值的路径。路径采用形式。 `/opt/kafka/external-configuration/SECRET-VOLUME-NAME/PROPERTIES-FILENAME:USERNAME-PROPERTY`
b. 密码属性值的路径。路径采用形式。 `/opt/kafka/external-configuration/SECRET-VOLUME-NAME/PROPERTIES-FILENAME:PASSWORD-PROPERTY`

- 如果使用的是资源，请使用自定义资源部分中已装入的属性文件中的值。 `KafkaConnector` `spec.config`

例如：

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnector
metadata:
  name: my-connector
  # ...
spec:
  class: "MyDbConnector"
  tasksMax: 3
  config:
    database: "my-postgresql:5432"
    username: "${file:/opt/kafka/external-configuration/connector-config/connector.properties:dbUsername}"
    password: "${file:/opt/kafka/external-configuration/connector-config/connector.properties:dbPassword}"
```

额外资源

- 有关Kafka Connect中外部配置的更多信息，请参阅模式参考。 `ExternalConfiguration`

2.3.15。启用资源 KafkaConnector

要启用Kafka Connect集群，请将注释添加到或自定义资源。先决条件 `KafkaConnectors` `strimzi.io/use-connector-resources` `KafkaConnect` `KafkaConnectS2I`

- 正在运行的集群 Operator

程序

1. 编辑或资源。添加注释。例如： `KafkaConnect` `KafkaConnectS2I` `strimzi.io/use-connector-resources`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  # ...
```

2. 使用以下命令创建或更新资源： `kubectl apply`

```
kubectl apply -f kafka-connect.yaml
```

额外资源

- 创建和管理连接器
- 将资源部署到Kafka Connect `KafkaConnector`
- `KafkaConnect` 模式参考
- `KafkaConnectS2I` 模式参考

2.3.16。作为具有Source2Image支持的Kafka Connect集群的一部分创建的资源列表

Kubernetes集群中的集群 Operator 将创建以下资源：

connect-cluster-name-connect-sourceImageStream用作新建Docker镜像的基础镜像。connect-cluster-name-connectBuildConfig负责构建新的Kafka Connect Docker镜像.connect-cluster-name-connectImageStream连接新创建的Docker镜像的位置.connect-cluster-name-connectDeploymentConfig，负责创建Kafka Connect工作节点pods.connect-cluster-name-connect-apiService，它公开用于管理Kafka Connect集群的REST接口.connect-cluster-name-configConfigMap，其中包含Kafka Connect辅助配置，并通过为Kafka Connect辅助节点配置的Kafka代理pods.connect-cluster-name-connectPod中断预算作为卷安装。

2.4。Kafka MirrorMaker配置

本章介绍如何在Strimzi群集中配置Kafka MirrorMaker部署以在Kafka群集之间复制数据。

您可以将Strimzi与MirrorMaker或MirrorMaker 2.0一起使用。MirrorMaker 2.0是最新版本，并提供了一种更有效的方式来在Kafka群集之间镜像数据。

如果使用的是MirrorMaker，请配置资源。 `KafkaMirrorMaker`

以下过程显示如何配置资源：

- 配置Kafka MirrorMaker

该资源的完整架构在KafkaMirrorMaker架构参考中进行了描述。 `KafkaMirrorMaker`

注意	应用于资源的标签也将应用于包括Kafka MirrorMaker的Kubernetes资源。这提供了一种方便的机制，可以根据需要标记资源。 `KafkaMirrorMaker`

2.4.1。配置Kafka MirrorMaker

使用资源的属性来配置您的Kafka MirrorMaker部署。 `KafkaMirrorMaker`

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。此过程显示在消费方和生产方使用TLS加密和身份验证的配置。
先决条件

- 有关运行以下命令的说明，请参阅《*部署Strimzi*》指南：
  - 集群 Operator
  - Kafka集群
- 源和目标Kafka群集必须可用

程序

1. 编辑资源的属性。 `spec` `KafkaMirrorMaker`

   此示例配置中显示了可以配置的属性：

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaMirrorMaker
metadata:
  name: my-mirror-maker
spec:
  replicas: 3 (1)
  consumer:
    bootstrapServers: my-source-cluster-kafka-bootstrap:9092 (2)
    groupId: "my-group" (3)
    numStreams: 2 (4)
    offsetCommitInterval: 120000 (5)
```

```
  tls: (6)
    trustedCertificates:
    - secretName: my-source-cluster-ca-cert
      certificate: ca.crt
  authentication: (7)
    type: tls
    certificateAndKey:
      secretName: my-source-secret
      certificate: public.crt
      key: private.key
  config: (8)
    max.poll.records: 100
    receive.buffer.bytes: 32768
    ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" (9)
    ssl.enabled.protocols: "TLSv1.2"
    ssl.protocol: "TLSv1.2"
    ssl.endpoint.identification.algorithm: HTTPS (10)
producer:
  bootstrapServers: my-target-cluster-kafka-bootstrap:9092
  abortOnSendFailure: false (11)
  tls:
    trustedCertificates:
    - secretName: my-target-cluster-ca-cert
      certificate: ca.crt
  authentication:
    type: tls
    certificateAndKey:
      secretName: my-target-secret
      certificate: public.crt
      key: private.key
  config:
    compression.type: gzip
    batch.size: 8192
    ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" (12)
    ssl.enabled.protocols: "TLSv1.2"
    ssl.protocol: "TLSv1.2"
    ssl.endpoint.identification.algorithm: HTTPS (13)
whitelist: "my-topic|other-topic" (14)
resources: (15)
  requests:
    cpu: "1"
    memory: 2Gi
  limits:
    cpu: "2"
    memory: 2Gi
logging: (16)
  type: inline
  loggers:
    mirrormaker.root.logger: "INFO"
readinessProbe: (17)
  initialDelaySeconds: 15
  timeoutSeconds: 5
livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
metrics: (18)
  lowercaseOutputName: true
  rules:
    - pattern: "kafka.server<type=(.+), name=(.+)PerSec\\w*><>Count"
      name: "kafka_server_$1_$2_total"
    - pattern: "kafka.server<type=(.+), name=(.+)PerSec\\w*,
      topic=(.+)><>Count"
      name: "kafka_server_$1_$2_total"
      labels:
        topic: "$3"
jvmOptions: (19)
  "-Xmx": "1g"
  "-Xms": "1g"
image: my-org/my-image:latest (20)
template: (21)
  pod:
    affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
                - key: application
                  operator: In
                  values:
                    - postgresql
                    - mongodb
            topologyKey: "kubernetes.io/hostname"
  connectContainer: (22)
    env:
```

```
        - name: JAEGER_SERVICE_NAME
          value: my-jaeger-service
        - name: JAEGER_AGENT_HOST
          value: jaeger-agent-name
        - name: JAEGER_AGENT_PORT
          value: "6831"
tracing: (23)
  type: jaeger
```

a. 复制节点的数量。
b. 消费者和生产者的引导服务器。
c. 消费者的组ID。
d. 消费者流的数量。
e. 偏移量自动提交间隔（以毫秒为单位）。
f. 使用密钥名称的TLS加密，在该密钥名称下，TLS证书以X.509格式存储给使用者或生产者。
g. 使用TLS机制（如此处所示），使用OAuth承载令牌或基于SASL的SCRAM-SHA-512或PLAIN机制对消费者或生产者进行身份验证。
h. 消费者和生产者的 Kafka配置选项。
i. 外部侦听器的SSL属性可与TLS版本的特定*密码套件*一起运行。
j. 通过设置为启用主机名验证。空字符串将禁用验证。 HTTPS
k. 如果该属性设置为，则Kafka MirrorMaker将退出，并且在消息发送失败后容器将重新启动。 abortOnSendFailure true
l. 外部侦听器的SSL属性可与TLS版本的特定*密码套件*一起运行。
m. 通过设置为启用主机名验证。空字符串将禁用验证。 HTTPS
n. 甲*白名单的* Topic 从源镜像到目标Kafka群集。
o. 请求保留支持的资源，当前和，以及用于指定可以消耗的最大资源的限制。 cpu memory
p. 通过ConfigMap 直接添加（）或间接添加（）的指定记录器和日志级别。必须将自定义ConfigMap放在或键下。MirrorMaker有一个名为的记录器。您可以将日志级别设置为INFO，ERROR，WARN，TRACE，DEBUG，FATAL或OFF。 inlineexternal log4j.properties log4j2.properties mirrormaker.root.logger
q. 运行状况检查 以了解何时重新启动容器（活动性）以及何时容器可以接受流量（就绪）。
r. Prometheus指标，在此示例中，通过配置Prometheus JMX导出器启用了该指标。您可以使用启用指标，而无需进一步配置。 metrics: {}
s. JVM配置选项 可优化运行Kafka MirrorMaker的虚拟机（VM）的性能。
t. 进阶选项：容器镜像配置，仅在特殊情况下才建议使用。
u. 模板定制。在这里，以非亲和性调度了一个Pod，因此没有在具有相同主机名的节点上调度该Pod。
v. 还使用Jaeger设置了环境变量以进行分布式跟踪。
w. 为Jaeger启用了分布式跟踪。

警告	将属性设置为，生产者将尝试发送 Topic 中的下一条消息。由于没有尝试重新发送失败的消息，因此原始消息可能会丢失。 abortOnSendFailure false

2. 创建或更新资源：

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

2.4.2。作为Kafka MirrorMaker的一部分创建的资源列表

Kubernetes集群中的集群 Operator 创建了以下资源：

<mirror-maker-name> -mirror-makerDeployment负责创建Kafka MirrorMaker容器。<mirror-maker-name> -configConfigMap包含Kafka MirrorMaker的辅助配置，并由Kafka经纪人容器作为卷安装为Kafka MirrorMaker工作程序节点配置的。<mirror-maker-name> -mirror-makerPod中断预算。

2.5。Kafka MirrorMaker 2.0配置

本节介绍如何在Strimzi群集中配置Kafka MirrorMaker 2.0部署。

MirrorMaker 2.0用于在数据中心内或跨数据中心的两个或多个活动Kafka群集之间复制数据。

跨集群的数据复制支持需要以下场景：

- 系统发生故障时恢复数据
- 汇总数据以进行分析
- 限制对特定集群的数据访问
- 在特定位置提供数据以改善延迟

如果使用的是MirrorMaker 2.0，请配置资源。 KafkaMirrorMaker2

MirrorMaker 2.0引入了一种在群集之间复制数据的全新方法。

因此，资源配置与MirrorMaker的早期版本不同。如果选择使用MirrorMaker 2.0，则当前不支持旧版，因此必须将任何资源手动转换为新格式。

MirrorMaker 2.0如何复制数据的说明如下：

- MirrorMaker 2.0数据复制

以下过程显示了如何为MirrorMaker 2.0配置资源：

- 在Kafka集群之间同步数据

该资源的完整架构在KafkaMirrorMaker2架构参考中进行了描述。 KafkaMirrorMaker2

2.5.1。MirrorMaker 2.0数据复制

MirrorMaker 2.0使用来自源Kafka群集的消息，并将消息写入目标Kafka群集。

MirrorMaker 2.0使用：

- 源集群配置以使用源集群中的数据
- 目标集群配置以将数据输出到目标集群

MirrorMaker 2.0基于Kafka Connect框架，该*连接器*管理群集之间的数据传输。MirrorMaker 2.0将 Topic 从源群集复制到目标群集。 `MirrorSourceConnector`

将数据从一个群集*镜像*到另一群集的过程是异步的。推荐的模式是让消息与源Kafka群集一起在本地生成，然后在目标Kafka群集附近远程使用。

MirrorMaker 2.0可以与多个源群集一起使用。

图1.跨两个集群的复制

2.5.2。集群配置

您可以在*主动/被动*或*主动/主动*群集配置中使用MirrorMaker 2.0 。

- 在*主动/被动*配置中，来自主动群集的数据被复制到被动群集中，该被动群集保持备用状态，例如，在系统故障时用于数据恢复。
- 在*主动/主动*配置中，两个群集均处于活动状态，并且同时提供相同的数据，如果要在不同地理位置的本地提供相同的数据，这很有用。

期望生产者和消费者仅连接到活动集群。

双向复制

MirrorMaker 2.0体系结构在*主动/主动*群集配置中支持双向复制。每个目标目的地都需要MirrorMaker 2.0群集。

每个群集都使用*源* Topic 和*远程* Topic 的概念复制另一个群集的数据。由于每个集群中存储了相同的 Topic ，因此MirrorMaker 2.0会自动重命名远程 Topic 以表示源集群。

图2. Topic 重命名

通过标记原始群集，不会将 Topic 复制回该群集。

通过*远程* Topic 进行复制的概念在配置需要数据聚合的体系结构时非常有用。消费者可以在同一群集中订阅源和远程 Topic ，而无需单独的聚合群集。

Topic 配置同步

Topic 配置在源群集和目标群集之间自动同步。通过同步配置属性，减少了重新平衡的需求。

数据的完整性

MirrorMaker 2.0监视源 Topic ，并将所有配置更改传播到远程 Topic ，检查并创建缺少的分区。仅MirrorMaker 2.0可以写入远程 Topic 。

偏移量跟踪

MirrorMaker 2.0使用*内部 Topic* 跟踪消费者组的偏移量。

- 的*偏移量的同步* Topic 映射源和目标偏移从记录元数据复制的 Topic 分区
- 该*检查站* Topic 地图最后为每个消费者组复制 Topic 分区中的源和目标集群中犯偏移

通过配置以预定间隔跟踪*检查点* Topic 的偏移量。通过这两个 Topic ，都可以在故障转移时从正确的偏移位置完全恢复复制。

MirrorMaker 2.0使用其发出*检查点*以进行偏移量跟踪。 `MirrorCheckpointConnector`

连接检查

甲*心跳*簇之间内部 Topic 检查连通性。

该*心跳*的话题从源群集复制。

目标集群使用该 Topic 检查：

- 连接器正在管理群集之间的连接
- 源集群可用

MirrorMaker 2.0使用其发出执行这些检查的*心跳*。 `MirrorHeartbeatConnector`

2.5.3。ACL规则同步

如果您**不**使用用户 Operator ，则可以通过ACL访问远程 Topic 。

如果正在使用（没有用户 Operator ），则管理代理访问的ACL规则也适用于远程 Topic 。可以阅读源 Topic 的用户可以阅读其远程等效项。 `AclAuthorizer`

注意	OAuth 2.0授权不支持以这种方式访问远程 Topic 。

2.5.4。使用MirrorMaker 2.0在Kafka集群之间同步数据

使用MirrorMaker 2.0通过配置在Kafka群集之间同步数据。

配置必须指定：

- 每个Kafka集群
- 每个群集的连接信息，包括TLS身份验证
- 复制流程和方向
    - 集群到集群
    - 话题对话题

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

注意	继续支持MirrorMaker的早期版本。如果希望使用为先前版本配置的资源，则必须将它们更新为MirrorMaker 2.0支持的格式。

MirrorMaker 2.0为属性（例如复制因子）提供默认配置值。最小配置，默认值保持不变，将类似于以下示例：

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  version: 2.6.0
  connectCluster: "my-cluster-target"
  clusters:
  - alias: "my-cluster-source"
    bootstrapServers: my-cluster-source-kafka-bootstrap:9092
  - alias: "my-cluster-target"
    bootstrapServers: my-cluster-target-kafka-bootstrap:9092
  mirrors:
  - sourceCluster: "my-cluster-source"
    targetCluster: "my-cluster-target"
    sourceConnector: {}
```

您可以使用TLS或SASL身份验证为源群集和目标群集配置访问控制。此过程显示了对源群集和目标群集使用TLS加密和身份验证的配置。
先决条件

- 有关运行以下命令的说明，请参阅《*部署Strimzi*》指南：
    - 集群 Operator
    - Kafka集群
- 源和目标Kafka群集必须可用

程序

1. 编辑资源的属性。 `spec` `KafkaMirrorMaker2`

   此示例配置中显示了可以配置的属性：

   ```
   apiVersion: kafka.strimzi.io/v1alpha1
   kind: KafkaMirrorMaker2
   metadata:
     name: my-mirror-maker2
   spec:
     version: 2.6.0 (1)
     replicas: 3 (2)
     connectCluster: "my-cluster-target" (3)
     clusters: (4)
     - alias: "my-cluster-source" (5)
       authentication: (6)
         certificateAndKey:
           certificate: source.crt
           key: source.key
           secretName: my-user-source
         type: tls
       bootstrapServers: my-cluster-source-kafka-bootstrap:9092 (7)
       tls: (8)
         trustedCertificates:
         - certificate: ca.crt
           secretName: my-cluster-source-cluster-ca-cert
     - alias: "my-cluster-target" (9)
       authentication: (10)
         certificateAndKey:
           certificate: target.crt
           key: target.key
           secretName: my-user-target
         type: tls
       bootstrapServers: my-cluster-target-kafka-bootstrap:9092 (11)
       config: (12)
   ```

```
    config.storage.replication.factor: 1
    offset.storage.replication.factor: 1
    status.storage.replication.factor: 1
    ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" (13)
    ssl.enabled.protocols: "TLSv1.2"
    ssl.protocol: "TLSv1.2"
    ssl.endpoint.identification.algorithm: HTTPS (14)
  tls: (15)
    trustedCertificates:
    - certificate: ca.crt
      secretName: my-cluster-target-cluster-ca-cert
mirrors: (16)
- sourceCluster: "my-cluster-source" (17)
  targetCluster: "my-cluster-target" (18)
  sourceConnector: (19)
    config:
      replication.factor: 1 (20)
      offset-syncs.topic.replication.factor: 1 (21)
      sync.topic.acls.enabled: "false" (22)
  heartbeatConnector: (23)
    config:
      heartbeats.topic.replication.factor: 1 (24)
  checkpointConnector: (25)
    config:
      checkpoints.topic.replication.factor: 1 (26)
  topicsPattern: ".*" (27)
  groupsPattern: "group1|group2|group3" (28)
resources: (29)
  requests:
    cpu: "1"
    memory: 2Gi
  limits:
    cpu: "2"
    memory: 2Gi
logging: (30)
  type: inline
  loggers:
    log4j.rootLogger: "INFO"
readinessProbe: (31)
  initialDelaySeconds: 15
  timeoutSeconds: 5
livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
jvmOptions: (32)
  "-Xmx": "1g"
  "-Xms": "1g"
image: my-org/my-image:latest (33)
template: (34)
  pod:
    affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
                - key: application
                  operator: In
                  values:
                    - postgresql
                    - mongodb
            topologyKey: "kubernetes.io/hostname"
  connectContainer: (35)
    env:
      - name: JAEGER_SERVICE_NAME
        value: my-jaeger-service
      - name: JAEGER_AGENT_HOST
        value: jaeger-agent-name
      - name: JAEGER_AGENT_PORT
        value: "6831"
tracing:
  type: jaeger (36)
externalConfiguration: (37)
  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
```

a. Kafka Connect 版本。
b. 复制节点的数量。
c. Kafka Connect的群集别名。
d. 同步Kafka集群的规范。
e. 源Kafka群集的群集别名。
f. 使用TLS机制（如此处所示），使用OAuth承载令牌或基于SASL的SCRAM-SHA-512或PLAIN机制对源群集进行身份验证。
g. 用于连接到源Kafka群集的Bootstrap服务器。
h. 使用密钥名称进行TLS加密，在该密钥名称下，源Kafka群集以X.509格式存储TLS证书。
i. 目标Kafka群集的群集别名。
j. 目标Kafka群集的身份验证与源Kafka群集的身份验证配置相同。
k. 引导服务器， 用于连接到目标Kafka集群。
l. Kafka Connect配置。可以提供标准的Apache Kafka配置，但仅限于Strimzi不直接管理的那些属性。
m. 外部侦听器的SSL属性可与TLS版本的特定*密码套件*一起运行。
n. 通过设置为启用主机名验证。空字符串将禁用验证。 `HTTPS`
o. 目标Kafka群集的TLS加密的配置方式与源Kafka群集的相同。
p. MirrorMaker 2.0连接器。
q. MirrorMaker 2.0连接器使用的源群集的群集别名。
r. MirrorMaker 2.0连接器使用的目标群集的群集别名。
s. 用于 `MirrorSourceConnector` 创建远程 Topic 的配置。将覆盖默认配置选项。 `config`
t. 在目标群集上创建的镜像 Topic 的复制因子。
u. 内部 Topic 的复制因子，用于映射源和目标群集的偏移量。 `MirrorSourceConnector` `offset-syncs`
v. 当ACL规则同步启用，ACL用于同步的 Topic 。默认值为。 `true`
w. 用于 `MirrorHeartbeatConnector` 执行连接检查的配置。将覆盖默认配置选项。 `config`
x. 在目标群集上创建的心跳 Topic 的复制因子。
y. `MirrorCheckpointConnector` 跟踪偏移量的配置。将覆盖默认配置选项。 `config`
z. 在目标群集上创建的检查点 Topic 的复制因子。
aa. 从源群集进行 Topic 复制，定义为正则表达式模式。在这里，我们要求所有 Topic 。
ab. 消费者组从源群集复制为正则表达式模式。在这里，我们按名称请求三个消费者组。您可以使用逗号分隔的列表。
ac. 请求保留支持的资源，当前和，以及用于指定可以消耗的最大资源的限制。 `cpu` `memory`
ad. 通过ConfigMap 直接添加（）或间接添加（）的指定Kafka Connect记录器和日志级别。必须将自定义ConfigMap放在或键下。Kafka Connect有一个名为的记录器。您可以将日志级别设置为INFO，ERROR，WARN，TRACE，DEBUG，FATAL或OFF。 `inlineexternal` `log4j.properties` `log4j2.properties` `log4j.rootLogger`
ae. 运行状况检查 以了解何时重新启动容器（活动性）以及何时容器可以接受流量（就绪）。
af. JVM配置选项 可优化运行Kafka MirrorMaker的虚拟机（VM）的性能。
ag. 进阶选项：容器镜像配置，仅在特殊情况下才建议使用。
ah. 模板定制。在这里，以非亲和性调度了一个Pod，因此没有在具有相同主机名的节点上调度该Pod。
ai. 还使用Jaeger设置了环境变量以进行分布式跟踪。
aj. 为Jaeger启用了分布式跟踪。
ak. 安装到Kafka MirrorMaker上的Kubernetes Secret作为环境变量的外部配置。

2. 创建或更新资源：

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

2.6。Kafka Bridge配置

模式参考中描述了资源的完整模式。所有应用于所需资源的标签也将应用于构成Kafka Bridge集群的Kubernetes资源。这提供了一种方便的机制，可以根据需要标记资源。 `KafkaBridge` `KafkaBridge` `KafkaBridge`

“ *使用Strimzi*”指南介绍了如何启用对网桥的监视。

2.6.1。复制品

Kafka Bridge可以运行多个节点。节点数在资源中定义。运行具有多个节点的Kafka Bridge可以提供更好的可用性和可伸缩性。但是，在Kubernetes上运行Kafka Bridge时，并非绝对必须运行Kafka Bridge的多个节点以实现高可用性。 `KafkaBridge`

重要	如果将部署Kafka Bridge的节点崩溃，Kubernetes将自动将Kafka Bridge pod的时间表重新安排到其他节点。为了防止在不同的Kafka Bridge实例处理客户端使用者请求时出现问题，必须使用基于寻址的路由来确保将请求路由到正确的Kafka Bridge实例。此外，每个独立的Kafka Bridge实例必须具有一个副本。Kafka Bridge实例具有自己的状态，不会与其他实例共享。

配置节点数

使用中的属性配置Kafka Bridge节点的数量。先决条件 `replicas` `KafkaBridge.spec`

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性。例如： `replicas` `KafkaBridge`

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  replicas: 3
  # ...
```

2. 创建或更新资源。

```
kubectl apply -f your-file
```

2.6.2。引导服务器

Kafka桥始终与Kafka集群结合使用。Kafka群集被指定为引导服务器列表。在Kubernetes上，理想情况下，该列表必须包含名为的Kafka集群引导服务以及端口9092（用于普通流量）或9093（用于加密流量）。 *cluster-name*-kafka-bootstrap

引导服务器列表在中的属性中配置。必须将服务器定义为指定一个或多个Kafka代理的逗号分隔列表，或者将服务指定为成对指向Kafka代理的服务。 bootstrapServers KafkaBridge.kafka.spec *hostname*:_port_

将Kafka Bridge与非Strimzi管理的Kafka群集一起使用时，可以根据群集的配置指定引导服务器列表。

配置引导服务器

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性。例如： bootstrapServers KafkaBridge

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  bootstrapServers: my-cluster-kafka-bootstrap:9092
  # ...
```

2. 创建或更新资源。

```
kubectl apply -f your-file
```

2.6.3。使用TLS连接到Kafka经纪人

默认情况下，Kafka Bridge尝试使用纯文本连接连接到Kafka代理。如果您更喜欢使用TLS，则需要其他配置。

TLS支持Kafka连接到Kafka桥

在中的属性中配置了对Kafka连接的TLS支持。该属性包含密钥名称的机密列表，用于存储证书。证书必须以X509格式存储。显示带有多个证书的TLS配置的示例 tls KafkaBridge.spec tls

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  tls:
    trustedCertificates:
    - secretName: my-secret
      certificate: ca.crt
    - secretName: my-other-secret
      certificate: certificate.crt
  # ...
```

当多个证书存储在同一秘密中时，可以多次列出。
显示带有来自同一机密的多个证书的TLS配置的示例

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  tls:
    trustedCertificates:
    - secretName: my-secret
      certificate: ca.crt
    - secretName: my-secret
      certificate: ca2.crt
  # ...
```

在Kafka Bridge中配置TLS

先决条件

- Kubernetes集群
- 正在运行的集群 Operator
- 如果存在，则用于TLS服务器身份验证的证书的名称，以及用于将证书存储在其中的密钥。 `Secret` `Secret`

程序

1. （可选）如果尚不存在它们，请在文件中准备用于身份验证的TLS证书，并创建一个。 `Secret`

注意	由集群 Operator 为Kafka集群创建的机密可以直接使用。

   ```
   kubectl create secret generic my-secret --from-file=my-file.crt
   ```

2. 编辑资源中的属性。例如： `tls` `KafkaBridge`

   ```
   apiVersion: kafka.strimzi.io/v1alpha1
   kind: KafkaBridge
   metadata:
     name: my-bridge
   spec:
     # ...
     tls:
           trustedCertificates:
           - secretName: my-cluster-cluster-cert
             certificate: ca.crt
     # ...
   ```

3. 创建或更新资源。

   ```
   kubectl apply -f your-file
   ```

2.6.4。通过身份验证连接到Kafka经纪人

默认情况下，Kafka Bridge将尝试在不进行身份验证的情况下连接到Kafka代理。通过资源启用身份验证。 `KafkaBridge`

Kafka Bridge中的身份验证支持

通过中的属性配置身份验证。该属性指定应使用的身份验证机制的类型以及取决于该机制的其他配置详细信息。当前支持的身份验证类型为： `authentication` `KafkaBridge.spec` `authentication`

- TLS客户端身份验证
- 使用SCRAM-SHA-512机制的基于SASL的身份验证
- 使用PLAIN机制的基于SASL的身份验证
- 基于OAuth 2.0令牌的身份验证

TLS客户端身份验证

要使用TLS客户端身份验证，请将属性设置为值。TLS客户端身份验证使用TLS证书进行身份验证。该证书在属性中指定，并且始终从Kubernetes机密中加载。秘密中，证书必须以X509格式存储在两个不同的密钥下：公用密钥和专用密钥。 `type` `tls` `certificateAndKey`

注意	TLS客户端身份验证只能与TLS连接一起使用。有关Kafka Bridge中TLS配置的更多详细信息，请参见使用TLS连接到Kafka代理。

TLS客户端身份验证配置示例

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  authentication:
    type: tls
    certificateAndKey:
      secretName: my-secret
      certificate: public.crt
      key: private.key
  # ...
```

SCRAM-SHA-512身份验证

要将Kafka Bridge配置为使用基于SASL的SCRAM-SHA-512身份验证，请将属性设置为。此身份验证机制需要用户名和密码。 `type` `scram-sha-512`

- 在属性中指定用户名。 `username`
- 在属性中，指定指向包含密码的链接。该属性包含的名称和属性包含其下的密码存储在里面的键的名称。 `passwordSecret` `Secret` `secretName` `Secret` `password` `Secret`

重要	不要在字段中指定实际密码。 `password`

基于SASL的示例SCRAM-SHA-512客户端身份验证配置

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  authentication:
    type: scram-sha-512
    username: my-bridge-user
    passwordSecret:
      secretName: my-bridge-user
      password: my-bridge-password-key
  # ...
```

基于SASL的PLAIN身份验证

要将Kafka Bridge配置为使用基于SASL的PLAIN身份验证，请将属性设置为。此身份验证机制需要用户名和密码。 `type` `plain`

警告	SASL PLAIN机制将以明文形式在网络上传输用户名和密码。如果启用了TLS加密，则仅使用SASL PLAIN身份验证。

- 在属性中指定用户名。 `username`
- 在属性中，指定指向包含密码的链接。该属性包含的名称，该属性包含密钥的名称，在密钥名称下存储密码。 `passwordSecret` `Secret` `secretName` `Secret` `password` `Secret`

重要	不要在字段中指定实际密码。 `password`

显示基于SASL的PLAIN客户端身份验证配置的示例

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  authentication:
    type: plain
    username: my-bridge-user
    passwordSecret:
      secretName: my-bridge-user
      password: my-bridge-password-key
  # ...
```

在Kafka Bridge中配置TLS客户端身份验证

先决条件

- Kubernetes集群
- 正在运行的集群 Operator
- 如果存在，则带有用于TLS客户端身份验证的公钥和私钥的的名称，以及将其存储在 `Secret` `Secret`

程序

1. （可选）如果尚不存在，请在文件中准备用于身份验证的密钥，然后创建。 `Secret`

注意	可以使用由用户 Operator 创建的机密。

   ```
   kubectl create secret generic my-secret --from-file=my-public.crt --from-file=my-private.key
   ```

2. 编辑资源中的属性。例如： `authentication` `KafkaBridge`

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  authentication:
  type: tls
  certificateAndKey:
    secretName: my-secret
    certificate: my-public.crt
    key: my-private.key
  # ...
```

3. 创建或更新资源。

```
kubectl apply -f your-file
```

在Kafka Bridge中配置SCRAM-SHA-512身份验证

先决条件

- Kubernetes集群
- 正在运行的集群 Operator
- 用于验证的用户的用户名
- 如果存在，请输入的名称以及用于身份验证的密码，以及用于存储密码的密钥 `Secret` `Secret`

程序

1. （可选）如果尚不存在，请准备一个文件，其中包含用于身份验证的密码并创建。 `Secret`

注意	可以使用由用户 Operator 创建的机密。

```
echo -n 'PASSWORD' > MY-PASSWORD.txt
kubectl create secret generic MY-SECRET --from-file=MY-PASSWORD.txt
```

2. 编辑资源中的属性。例如： `authentication` `KafkaBridge`

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  authentication:
    type: scram-sha-512
    username: MY-USERNAME
    passwordSecret:
      secretName: MY-SECRET
      password: MY-PASSWORD.txt
  # ...
```

3. 创建或更新资源。

```
kubectl apply -f your-file
```

2.6.5。Kafka Bridge配置

Strimzi允许您通过编辑列出的某些选项，自定义的ApacheKafka桥节点的配置Apache的Kafka配置文档，为消费者和生产者的ApacheKafka的配置文件。

可以配置的配置选项涉及：

- Kafka集群引导地址
- 安全性（加密，认证和授权）
- 消费者配置
- 生产者配置
- HTTP配置

Kafka Bridge Consumer配置

使用中的属性配置Kafka Bridge使用者。此属性包含Kafka Bridge使用者配置选项作为键。值可以是以下JSON类型之一： `KafkaBridge.spec.consumer`

- 串
- 数
- 布尔型

用户可以为用户指定和配置Apache Kafka配置文档中列出的选项，但由Strimzi直接管理的选项除外。具体来说，禁止所有键等于或以下列字符串之一开头的配置选项：

- ssl.
- sasl.
- security.
- bootstrap.servers
- group.id

如果属性中存在禁止选项之一，则它将被忽略，并且警告消息将被打印到Cluster Operator日志文件中。所有其他选项将传递给Kafka config

重要	群集 operator 不验证提供的对象中的键或值。提供无效的配置后，Kafka Bridge群集可能无法启动或变得不稳定。在这种情况下，请在对象中修复配置，然后集群 Operator 可以将新配置推广到所有Kafka Bridge节点。 config KafkaBridge.spec.consumer.config

禁止选项也有例外。对于特定于TLS版本的*密码套件*，可以使用三个允许的配置选项进行客户端连接。密码套件结合了用于安全连接和数据传输的算法。您还可以配置该属性以启用或禁用主机名验证。Kafka Bridge使用者配置示例 ssl ssl.endpoint.identification.algorithm

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  consumer:
    config:
      auto.offset.reset: earliest
      enable.auto.commit: true
      ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" (1)
      ssl.enabled.protocols: "TLSv1.2" (2)
      ssl.protocol: "TLSv1.2" (3)
      ssl.endpoint.identification.algorithm: HTTPS (4)
    # ...
```

1. TLS的密码套件，使用密钥交换机制，认证算法，批量加密算法和MAC算法的组合。 ECDHE RSA AES SHA384
2. 启用SSl协议。 TLSv1.2
3. 指定生成SSL上下文的协议。允许的值为和。 TLSv1.2 TLSv1.1 TLSv1.2
4. 通过设置为启用主机名验证。空字符串将禁用验证。 HTTPS

Kafka Bridge Producer配置

Kafka Bridge生产者使用中的属性进行配置。此属性包含Kafka Bridge生产者配置选项作为键。值可以是以下JSON类型之一： KafkaBridge.spec.producer

- 串
- 数
- 布尔型

用户可以为生产者指定和配置Apache Kafka配置文档中列出的选项，但由Strimzi直接管理的选项除外。具体来说，禁止所有键等于或以下列字符串之一开头的配置选项：

- ssl.
- sasl.
- security.
- bootstrap.servers

重要	群集 operator 不验证提供的对象中的键或值。提供无效的配置后，Kafka Bridge群集可能无法启动或变得不稳定。在这种情况下，请在对象中修复配置，然后集群 Operator 可以将新配置推广到所有Kafka Bridge节点。 config KafkaBridge.spec.producer.config

禁止选项也有例外。对于特定于TLS版本的*密码套件*，可以使用三个允许的配置选项进行客户端连接。密码套件结合了用于安全连接和数据传输的算法。您还可以配置该属性以启用或禁用主机名验证。Kafka Bridge生产者配置示例 ssl ssl.endpoint.identification.algorithm

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  producer:
    config:
      acks: 1
      delivery.timeout.ms: 300000
      ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" (1)
      ssl.enabled.protocols: "TLSv1.2" (2)
      ssl.protocol: "TLSv1.2" (3)
      ssl.endpoint.identification.algorithm: HTTPS (4)
    # ...
```

1. TLS的密码套件，使用密钥交换机制，认证算法，批量加密算法和MAC算法的组合。 ECDHE RSA AES SHA384
2. 启用SSl协议。 TLSv1.2

3. 指定生成SSL上下文的协议。允许的值为和。 `TLSv1.2` `TLSv1.1` `TLSv1.2`
4. 通过设置为启用主机名验证。空字符串将禁用验证。 `HTTPS`

Kafka Bridge HTTP配置

使用中的属性设置Kafka Bridge HTTP配置。 `KafkaBridge.spec.http`

除了支持对Kafka群集的HTTP访问外，HTTP属性还提供了通过跨域资源共享（CORS）启用和定义Kafka桥访问控制的功能。CORS是一种HTTP机制，它允许浏览器从多个来源访问选定的资源。要配置CORS，您需要定义允许的资源来源和访问它们的HTTP方法的列表。请求中的其他HTTP标头描述了允许访问Kafka群集的来源。Kafka Bridge HTTP配置示例

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  http:
    port: 8080 (1)
    cors:
      allowedOrigins: "https://strimzi.io" (2)
      allowedMethods: "GET,POST,PUT,DELETE,OPTIONS,PATCH" (3)
  # ...
```

1. Kafka网桥在端口8080上侦听的默认HTTP配置。
2. 以逗号分隔的允许的CORS来源列表。您可以使用URL或Java正则表达式。
3. 逗号分隔的CORS允许的HTTP方法列表。

配置Kafka桥

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑，，或在财产资源。例如： `kafka` `http` `consumer` `producer` `KafkaBridge`

   ```
   apiVersion: kafka.strimzi.io/v1alpha1
   kind: KafkaBridge
   metadata:
     name: my-bridge
   spec:
     # ...
     bootstrapServers: my-cluster-kafka:9092
     http:
       port: 8080
     consumer:
       config:
         auto.offset.reset: earliest
     producer:
       config:
         delivery.timeout.ms: 300000
     # ...
   ```

2. 创建或更新资源。

   ```
   kubectl apply -f your-file
   ```

2.6.6。CPU和内存资源

对于每个部署的容器，Strimzi允许您请求特定资源并定义这些资源的最大消耗量。

Strimzi支持两种类型的资源：

- 中央处理器
- 记忆

Strimzi使用Kubernetes语法指定CPU和内存资源。

资源限制和要求

使用以下资源中的属性配置资源限制和请求： `resources`

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `Kafka.spec.entityOperator.topicOperator`
- `Kafka.spec.entityOperator.userOperator`
- `Kafka.spec.entityOperator.tlsSidecar`

- `Kafka.spec.KafkaExporter`
- `KafkaConnect.spec`
- `KafkaConnectS2I.spec`
- `KafkaBridge.spec`

额外资源

- 有关在Kubernetes上管理计算资源的更多信息，请参阅管理容器的计算资源。

资源要求

请求指定要为给定容器保留的资源。保留资源可确保它们始终可用。

重要	如果资源请求所需要的资源超出了Kubernetes集群中的可用空闲资源，则未安排Pod。

资源请求在属性中指定。Strimzi当前支持的资源请求： `requests`

- `cpu`
- `memory`

可以为一个或多个支持的资源配置请求。
所有资源的示例资源请求配置

```
# ...
resources:
  requests:
    cpu: 12
    memory: 64Gi
# ...
```

资源限制

限制指定给定容器可以消耗的最大资源。该限制不是保留的，可能并不总是可用。容器只有在可用时才能使用资源。资源限制应始终高于资源请求。

资源限制在属性中指定。Strimzi当前支持的资源限制： `limits`

- `cpu`
- `memory`

可以为一个或多个受支持的限制配置资源。
示例资源限制配置

```
# ...
resources:
  limits:
    cpu: 12
    memory: 64Gi
# ...
```

支持的CPU格式

支持以下格式的CPU请求和限制：

- CPU内核数，可以是整数（`5` CPU内核）或十进制（`2.5` CPU内核）。
- 数字或*毫微克* / *毫厘*（），其中`1000` *毫欧*是同一CPU核心。 `100m` `1`

示例CPU单元

```
# ...
resources:
  requests:
    cpu: 500m
  limits:
    cpu: 2.5
# ...
```

注意	1个CPU内核的计算能力可能会因所部署的Kubernetes平台而异。

额外资源

- 有关CPU规格的更多信息，请参见CPU的含义。

支持的内存格式

内存请求和限制以兆字节，千兆字节，兆字节和千兆字节指定。

- 要指定兆字节的内存，请使用后缀。例如。 `M` `1000M`
- 要以GB为单位指定内存，请使用后缀。例如。 `G` `1G`

- 要以兆字节为单位指定内存，请使用后缀。例如。 `Mi` `1000Mi`
- 要以千兆字节指定内存，请使用后缀。例如。 `Gi` `1Gi`

使用不同存储单元的示例

```
# ...
resources:
  requests:
    memory: 512Mi
  limits:
    memory: 2Gi
# ...
```

额外资源

- 有关内存规格和其他受支持单位的更多详细信息，请参阅内存的含义。

配置资源请求和限制

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性以指定集群部署。例如： `resources`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   spec:
     kafka:
       # ...
       resources:
         requests:
           cpu: "8"
           memory: 64Gi
         limits:
           cpu: "12"
           memory: 128Gi
       # ...
     zookeeper:
       # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

额外资源

- 有关架构的更多信息，请参见ResourceRequirements API参考。

2.6.7。Kafka Bridge记录器

Kafka Bridge具有自己的可配置记录器：

- `logger.bridge`
- `logger.<operation-id>`

您可以在记录器中进行替换以设置特定操作的日志级别： `<operation-id>` `logger.<operation-id>`

- `createConsumer`
- `deleteConsumer`
- `subscribe`
- `unsubscribe`
- `poll`
- `assign`
- `commit`
- `send`
- `sendToPartition`
- `seekToBeginning`
- `seekToEnd`
- `seek`
- `healthy`
- `ready`
- `openapi`

每个操作均根据OpenAPI规范定义，并具有相应的API端点，网桥通过该API端点接收来自HTTP客户端的请求。您可以更改每个端点上的日志级别，以创建有关传入和传出HTTP请求的细粒度日志记录信息。

必须配置每个记录器，将其分配为。例如，为操作记录器配置记录级别意味着定义以下内容： `name` `http.openapi.operation.<operation-id>` `send`

```
logger.send.name = http.openapi.operation.send
logger.send.level = DEBUG
```

Kafka Bridge使用Apache 记录器实现。记录器在文件中定义，该文件对和端点具有以下默认配置： `log4j2` `log4j2.properties` `healthy` `ready`

```
logger.healthy.name = http.openapi.operation.healthy
logger.healthy.level = WARN
logger.ready.name = http.openapi.operation.ready
logger.ready.level = WARN
```

默认情况下，所有其他操作的日志级别都设置为。 `INFO`

使用该属性来配置记录器和记录器级别。 `logging`

您可以通过指定记录器和直接（内联）级别或使用自定义（外部）ConfigMap来设置日志级别。如果使用ConfigMap，则将属性设置为包含外部日志记录配置的ConfigMap的名称。在ConfigMap内部，使用来描述日志记录配置。 `logging.name` `log4j.properties`

在这里，我们看到和的示例。内联记录 `inline` `external`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaBridge
spec:
  # ...
  logging:
    type: inline
    loggers:
      logger.bridge.level: "INFO"
      # enabling DEBUG just for send operation
      logger.send.name: "http.openapi.operation.send"
      logger.send.level: "DEBUG"
  # ...
```

外部记录

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaBridge
spec:
  # ...
  logging:
    type: external
    name: customConfigMap
  # ...
```

额外资源

- 垃圾收集器（GC）日志记录也可以启用（或禁用）。有关GC日志记录的更多信息，请参阅JVM配置。
- 有关日志级别的更多信息，请参阅Apache日志服务。

2.6.8。JVM选项

Strimzi的以下组件在虚拟机（VM）中运行：

- 阿帕奇·Kafka
- Apache ZooKeeper
- Apache Kafka连接
- Apache Kafka MirrorMaker
- Strimzi Kafka桥

JVM配置选项可优化不同平台和体系结构的性能。Strimzi允许您配置其中一些选项。

JVM配置

使用该属性可以为运行组件的JVM配置受支持的选项。 `jvmOptions`

支持的JVM选项有助于优化不同平台和体系结构的性能。

配置JVM选项

先决条件

- Kubernetes集群

- 正在运行的集群 Operator

程序

1. 编辑在属性，，，，或资源。例如： `jvmOptions` `Kafka` `KafkaConnect` `KafkaConnectS2I` `KafkaMirrorMaker` `KafkaBridge`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    jvmOptions:
      "-Xmx": "8g"
      "-Xms": "8g"
    # ...
  zookeeper:
    # ...
```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.6.9。健康检查

运行状况检查是定期测试，可验证应用程序的运行状况。当运行状况检查探针失败时，Kubernetes会认为该应用程序运行状况不佳，并尝试对其进行修复。

Kubernetes支持两种类型的Healthcheck探针：

- 活力探针
- 准备就绪探针

有关探针的更多详细信息，请参阅“ 配置活动性和就绪性探针”。两种类型的探针都用于Strimzi组件中。

用户可以为活动和就绪探针配置选定的选项。

健康检查配置

可以使用以下资源中的和属性来配置活动性和就绪性探针： `livenessProbe` `readinessProbe`

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `Kafka.spec.entityOperator.tlsSidecar`
- `Kafka.spec.entityOperator.topicOperator`
- `Kafka.spec.entityOperator.userOperator`
- `Kafka.spec.KafkaExporter`
- `KafkaConnect.spec`
- `KafkaConnectS2I.spec`
- `KafkaMirrorMaker.spec`
- `KafkaBridge.spec`

双方并支持以下选项： `livenessProbe` `readinessProbe`

- `initialDelaySeconds`
- `timeoutSeconds`
- `periodSeconds`
- `successThreshold`
- `failureThreshold`

有关和选项的更多信息，请参见模式参考。活动和就绪探针配置示例 `livenessProbe` `readinessProbe` `Probe`

```
# ...
readinessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
# ...
```

配置健康检查

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑或财产的，或资源。例如： `livenessProbe` `readinessProbe` `Kafka` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
       readinessProbe:
         initialDelaySeconds: 15
         timeoutSeconds: 5
       livenessProbe:
         initialDelaySeconds: 15
         timeoutSeconds: 5
       # ...
     zookeeper:
       # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.6.10。容器图片

Strimzi允许您配置将用于其组件的容器镜像。仅在需要使用其他容器注册表的特殊情况下才建议覆盖容器镜像。例如，由于您的网络不允许访问Strimzi使用的容器存储库。在这种情况下，您应该复制Strimzi图像或从源代码构建它们。如果配置的图像与Strimzi图像不兼容，则可能无法正常工作。

容器镜像配置

使用该属性指定要使用的容器图像。 `image`

警告	仅在特殊情况下才建议覆盖容器镜像。

配置容器镜像

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑在属性，或资源。例如： `image` `Kafka` `KafkaConnect` `KafkaConnectS2I`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
       image: my-org/my-image:latest
       # ...
     zookeeper:
       # ...
   ```

2. 创建或更新资源。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.6.11。配置窗格调度

重要	当将两个应用程序安排到同一Kubernetes节点时，这两个应用程序可能会使用相同的资源（如磁盘I / O）并影响性能。这会导致性能下降。避免与其他关键工作负载共享节点，使用正确的节点或仅将一组节点专用于Kafka的方式来调度Kafka Pod是避免此类问题的最佳方法。

根据其他应用程序调度Pod

避免关键应用程序共享节点

可以使用Pod抗关联性来确保关键应用程序永远不会安排在同一磁盘上。在运行Kafka群集时，建议使用pod anti-affinity以确保Kafka代理不与其他工作负载（如数据库）共享节点。

亲和力

可以使用以下资源中的属性来配置亲和力： `affinity`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

相似性配置可以包括不同类型的相似性：

- Pod亲和力和反亲和力
- 节点亲和力

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes节点和pod关联文档。 `affinity`

在Kafka组件中配置Pod抗亲和力

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源中的属性以指定集群部署。使用标签指定不应在同一节点上安排的Pod。该应设置为指定所选荚不应该使用相同的主机节点进行调度。例如： `affinity` `topologyKey` `kubernetes.io/hostname`

```
apiVersion: kafka.strimzi.io/v1beta1
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。

   可以使用： `kubectl apply`

```
kubectl apply -f your-file
```

将Pod调度到特定节点

节点调度

Kubernetes集群通常由许多不同类型的工作程序节点组成。有些针对CPU繁重的工作负载进行了优化，有些针对内存进行了优化，而其他针对存储（快速本地SSD）或网络进行了优化。使用不同的节点有助于优化成本和性能。为了获得最佳性能，允许安排Strimzi组件使用正确的节点很重要。

Kubernetes使用节点关联性将工作负载调度到特定节点上。节点亲缘关系允许您为将在其上调度容器的节点创建调度约束。该约束被指定为标签选择器。您可以使用内置节点标签（例如）或自定义标签来指定标签，以选择正确的节点。 `beta.kubernetes.io/instance-type`

亲和力

可以使用以下资源中的属性来配置亲和力： `affinity`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`

- `KafkaBridge.spec.template.pod`

相似性配置可以包括不同类型的相似性：

- Pod亲和力和反亲和力
- 节点亲和力

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes节点和pod关联文档。 `affinity`

在Kafka组件中配置节点关联

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 标记应安排Strimzi组件的节点。

   可以使用： `kubectl label`

   ```
   kubectl label node your-node node-type=fast-network
   ```

   或者，某些现有标签可能会被重用。
2. 编辑资源中的属性以指定集群部署。例如： `affinity`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   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。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

使用专用节点

专用节点

群集管理员可以将选定的Kubernetes节点标记为已污染。带有污点的节点不包括在常规调度中，普通的吊舱也不会安排在它们上运行。只能在节点上安排可以容忍在节点上设置污点的服务。在此类节点上运行的唯一其他服务将是系统服务，例如日志收集器或软件定义的网络。

污渍可用于创建专用节点。在专用节点上运行Kafka及其组件可具有许多优势。在同一节点上不会运行其他任何可能引起干扰或消耗Kafka所需资源的应用程序。这可以提高性能和稳定性。

要在专用节点上安排Kafka Pod，请配置节点亲和力和容差。

亲和力

可以使用以下资源中的属性来配置亲和力： `affinity`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

相似性配置可以包括不同类型的相似性：

- Pod亲和力和反亲和力
- 节点亲和力

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes节点和pod关联文档。 `affinity`

公差范围

可以使用以下资源中的属性来配置公差： `tolerations`

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaConnectS2I.spec.template.pod`
- `KafkaBridge.spec.template.pod`

该属性的格式遵循Kubernetes规范。有关更多详细信息，请参见Kubernetes污点和公差。 `tolerations`

设置专用节点并在其上调度Pod

先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 选择应用作专用节点。
2. 确保在这些节点上没有安排任何工作负载。
3. 在所选节点上设置污点：

   可以使用： `kubectl taint`

   ```
   kubectl taint node your-node dedicated=Kafka:NoSchedule
   ```

4. 此外，还向所选节点添加标签。

   可以使用： `kubectl label`

   ```
   kubectl label node your-node dedicated=Kafka
   ```

5. 编辑资源中的和属性以指定集群部署。例如： `affinity` `tolerations`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   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。

   可以使用： `kubectl apply`

   ```
   kubectl apply -f your-file
   ```

2.6.12。作为Kafka Bridge集群的一部分创建的资源列表

Kubernetes集群中的集群 Operator 创建了以下资源：

bridge-cluster-name-bridgeDeployment负责创建Kafka Bridge工作节点pods.bridge-cluster-name-bridge-serviceService公开Kafka Bridge集群的REST接口.bridge-cluster-name-bridge-configConfigMap其中包含Kafka Bridge辅助配置，并由为Kafka Bridge工作程序节点配置的Kafka代理pods.bridge-cluster-name-bridgePod中断预算作为卷安装。

2.7。自定义Kubernetes资源

Strimzi创建几个Kubernetes资源，例如，，，和，这是由 Operator Strimzi管理。只有负责管理特定Kubernetes资源的 Operator 才能更改该资源。如果您尝试手动更改由 Operator 管理的Kubernetes资源，则 Operator 将还原您的更改。 `Deployments` `StatefulSets` `Pods` `Services`

但是，如果要执行某些任务，例如，更改 Operator 管理的Kubernetes资源可能会很有用：

- 添加自定义标签或注释以控制Istio或其他服务的处理方式 `Pods`
- 管理集群如何创建类型服务 `Loadbalancer`

您可以使用Strimzi自定义资源中的属性进行此类更改。以下资源支持该属性。API参考提供了有关可定制字段的更多详细信息。 `template` `template`

Kafka.spec.kafkaSee KafkaClusterTemplate模式参考Kafka.spec.zookeeperSee ZookeeperClusterTemplate模式参考Kafka.spec.entityOperatorSee EntityOperatorTemplate模式参考Kafka.spec.kafkaExporterSee KafkaExporterTemplate模式参考Kafka.spec.cruiseControlSeeSeeControlControlTemplate Seek。 KafkaConnectTemplate模式参考KafkaMirrorMaker.spec请参阅KafkaMirrorMakerTemplate模式参考KafkaMirrorMaker2.spec请参阅KafkaConnectTemplate模式参考KafkaBridge.spec请参阅KafkaBridgeTemplate模式参考KafkaUser.spec请参阅KafkaUserTemplate模式参考

在以下示例中，该属性用于修改Kafka经纪人的标签： `template` `StatefulSet`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
  labels:
    app: my-cluster
spec:
  kafka:
    # ...
    template:
      statefulset:
        metadata:
          labels:
            mylabel: myvalue
    # ...
```

2.7.1。自定义图像拉出策略

Strimzi允许您为群集 Operator 部署的所有吊舱中的容器自定义镜像拉策略。使用Cluster Operator部署中的环境变量配置镜像拉策略。该环境变量可以设置为三个不同的值： `STRIMZI_IMAGE_PULL_POLICY` `STRIMZI_IMAGE_PULL_POLICY`

每次启动或重新启动Pod时，都会从注册表中拉出AlwaysContainer图像。仅在之前未拉过它们时，才从注册表中拉出IfNotPresentContainer图像。从不从注册表中拉出NeverContainer图像。

当前只能一次为所有Kafka，Kafka Connect和Kafka MirrorMaker群集自定义镜像提取策略。更改策略将导致所有Kafka，Kafka Connect和Kafka MirrorMaker群集的滚动更新。
额外资源

- 有关集群 Operator 配置的更多信息，请参阅《使用集群 Operator 》。
- 有关“镜像拉出策略”的更多信息，请参见中断。

2.8。外部记录

设置资源的日志记录级别时，可以直接在资源YAML 的属性中*内联*指定它们： `spec.logging`

```
spec:
  # ...
  logging:
    type: inline
    loggers:
      log4j.logger.kafka: "INFO"
```

或者，您可以指定*外部*日志记录：

```
spec:
  # ...
  logging:
    type: external
    name: customConfigMap
```

对于外部日志记录，日志记录属性在ConfigMap中定义。ConfigMap的名称在属性中引用。 `spec.logging.name`

使用ConfigMap的优点是，日志记录属性保存在一个地方，并且可以由多个资源访问。

2.8.1。创建一个ConfigMap进行日志记录

要使用ConfigMap定义日志记录属性，请创建ConfigMap，然后将其作为资源中日志记录定义的一部分进行引用。 `spec`

ConfigMap必须包含适当的日志记录配置。

- `log4j.properties` 用于Kafka组件，ZooKeeper和Kafka桥

- `log4j2.properties` 针对 Topic operator 和用户 operator

必须将配置放在这些属性下。

在这里，我们演示ConfigMap如何为Kafka资源定义根记录器。
程序

1. 创建ConfigMap。

   您可以在命令行中使用YAML文件或属性文件创建ConfigMap 。 `kubectl`

   带有Kafka的根记录器定义的ConfigMap示例：

   ```
   kind: ConfigMap
   apiVersion: kafka.strimzi.io/v1beta1
   metadata:
     name: logging-configmap
   data:
     log4j.properties:
       log4j.logger.kafka="INFO"
   ```

   从命令行使用属性文件：

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

   属性文件定义了日志记录配置：

   ```
   # Define the logger
   log4j.logger.kafka="INFO"
   # ...
   ```

2. 在资源的中定义*外部*日志记录，将设置为ConfigMap的名称。 `spec` `logging.name`

   ```
   spec:
     # ...
     logging:
       type: external
       name: logging-configmap
   ```

3. 创建或更新资源。

   ```
   kubectl apply -f kafka.yaml
   ```

3. 配置外部侦听器

使用外部侦听器将您的Strimzi Kafka集群向Kubernetes环境之外的客户端公开。

指定连接以在外部侦听器配置中公开Kafka。 `type`

- `nodeport` 使用类型 `NodePort` `Services`
- `loadbalancer` 使用类型 `Loadbalancer` `Services`
- `ingress` 使用Kubernetes 和NGINX Ingress Controller进行Kubernetes `Ingress`
- `route` 使用OpenShift 和HAProxy路由器 `Routes`

有关侦听器配置的更多信息，请参见模式参考。 `KafkaListeners`

注意	`route` 仅在OpenShift上受支持

额外资源

- 在Strimzi中访问Apache Kafka

3.1。使用节点端口访问Kafka

此过程描述了如何使用节点端口从外部客户端访问Strimzi Kafka群集。

要连接到代理，您需要Kafka *引导程序地址*的主机名和端口号，以及用于身份验证的证书。先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 使用设置为类型的外部侦听器配置资源。 `Kafka` `nodeport`

   例如：

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  kafka:
    # ...
    listeners:
      external:
        type: nodeport
        tls: true
        authentication:
          type: tls
        # ...
    # ...
  zookeeper:
    # ...
```

2. 创建或更新资源。

   ```
   kubectl apply -f KAFKA-CONFIG-FILE
   ```

   `NodePort` 将为每个Kafka经纪人创建一个类型服务，以及一个外部*引导服务*。引导服务将外部流量路由到Kafka代理。用于连接的节点地址将传播到Kafka自定义资源的。 `status`

   还将使用与资源相同的名称创建用于验证kafka代理身份的群集CA证书。 `Kafka`
3. 从资源状态检索可用于访问Kafka群集的引导地址。 `Kafka`

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

4. 如果启用了TLS加密，请提取代理证书颁发机构的公共证书。

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

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

3.2。使用负载均衡器访问Kafka

此过程描述了如何使用负载平衡器从外部客户端访问Strimzi Kafka群集。

要连接到代理，您需要*引导负载均衡器*的地址，以及用于TLS加密的证书。

对于使用访问负载均衡器，端口始终是9094
先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 使用设置为类型的外部侦听器配置资源。 `Kafka` `loadbalancer`

   例如：

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   spec:
     kafka:
       # ...
       listeners:
         external:
           type: loadbalancer
           authentication:
             type: tls
           # ...
       # ...
     zookeeper:
       # ...
   ```

2. 创建或更新资源。

   ```
   kubectl apply -f KAFKA-CONFIG-FILE
   ```

   `loadbalancer` 为每个Kafka代理创建一个类型服务和负载平衡器，以及一个外部*引导服务*。引导服务将外部流量路由到所有Kafka代理。用于连接的DNS名称和IP地址将传播到每个服务的。 `status`

   还将使用与资源相同的名称创建用于验证kafka代理身份的群集CA证书。 `Kafka`
3. 检索可用于从资源状态访问Kafka群集的引导服务的地址。 `Kafka`

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

4. 如果启用了TLS加密，请提取代理证书颁发机构的公共证书。

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

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

3.3。使用入口访问Kafka

此过程显示了如何使用Nginx Ingress从Kubernetes外部的外部客户端访问Strimzi Kafka群集。

要连接到代理，您需要输入主机名（播发的地址）作为Ingress *引导地址*以及用于身份验证的证书。

对于使用Ingress的访问，端口始终为443。TLS
直通

Kafka使用基于TCP的二进制协议，但用于Kubernetes的NGINX入口控制器旨在与HTTP协议一起使用。为了能够通过Ingress传递Kafka连接，Strimzi使用了NGINX Ingress Controller for Kubernetes的TLS传递功能。确保在您的NGINX入口控制器中启用了TLS 传递以进行Kubernetes部署。

由于它使用TLS直通功能，因此使用公开Kafka时无法禁用TLS加密。 `Ingress`

有关启用TLS传递的更多信息，请参阅TLS传递文档。先决条件

- Kubernetes集群
- 为Kubernetes部署了启用了TLS直通的NGINX Ingress Controller
- 正在运行的集群 Operator

程序

1. 使用设置为类型的外部侦听器配置资源。 `Kafka` `ingress`

   为引导服务和Kafka代理指定Ingress主机。

   例如：

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   spec:
     kafka:
       # ...
       listeners:
         external:
           type: ingress
           authentication:
             type: tls
           configuration: (1)
             bootstrap:
               host: bootstrap.myingress.com
             brokers:
             - broker: 0
               host: broker-0.myingress.com
             - broker: 1
               host: broker-1.myingress.com
             - broker: 2
               host: broker-2.myingress.com
       # ...
     zookeeper:
       # ...
   ```

   a. 引导服务和Kafka代理的入口主机。
2. 创建或更新资源。

   ```
   kubectl apply -f KAFKA-CONFIG-FILE
   ```

   `ClusterIP` 将为每个Kafka经纪人创建类型服务，以及一个附加的*引导服务*。这些服务由Ingress控制器用来将流量路由到Kafka代理。还为每个服务创建一个资源，以使用Ingress控制器公开它们。Ingress主机将传播到每个服务的。 `Ingress` `status`

   还将使用与资源相同的名称创建用于验证kafka代理身份的群集CA证书。 `Kafka`

   使用您在Kafka客户端的和端口443（*BOOTSTRAP-HOST：443*）中指定的引导主机的地址作为*引导地址*，以连接到Kafka集群。 `configuration`
3. 提取经纪人证书颁发机构的公共证书。

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

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

3.4。使用OpenShift路由访问Kafka

此过程描述了如何使用路由从OpenShift外部的外部客户端访问Strimzi Kafka群集。

要连接到代理，您需要一个主机名作为路由*引导程序地址*，以及用于TLS加密的证书。

对于使用接入线路，端口始终是443
的先决条件

- 一个OpenShift集群
- 正在运行的集群 Operator

程序

1. 使用设置为类型的外部侦听器配置资源。 `Kafka` `route`

   例如：

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   spec:
     kafka:
       # ...
       listeners:
         external:
           type: route
           # ...
       # ...
     zookeeper:
       # ...
   ```

2. 创建或更新资源。

   ```
   kubectl apply -f KAFKA-CONFIG-FILE
   ```

   `ClusterIP` 将为每个Kafka经纪人创建一个类型服务，以及一个外部*引导服务*。服务将流量从OpenShift路由路由到Kafka代理。还为每个服务创建一个OpenShift 资源，以使用HAProxy负载平衡器公开它们。用于连接的DNS地址将传播到每个服务的。 `Route` `status`

   还将使用与资源相同的名称创建用于验证kafka代理身份的群集CA证书。 `Kafka`
3. 检索可用于从资源状态访问Kafka群集的引导服务的地址。 `Kafka`

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

4. 提取经纪人证书颁发机构的公共证书。

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

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

4. 管理对Kafka的安全访问

您可以通过管理每个客户端对Kafka经纪人的访问权限来保护您的Kafka集群。

Kafka经纪人和客户之间的安全连接可以包括：

- 加密进行数据交换
- 验证以证明身份
- 授权允许或拒绝用户执行的操作

本章介绍如何在Kafka代理和客户端之间建立安全连接，并描述以下内容：

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

4.1。Kafka的安全性选项

使用该资源来配置用于Kafka身份验证和授权的机制。 `Kafka`

4.1.1。侦听器身份验证

对于Kubernetes集群中的客户端，您可以创建（不加密）或侦听器类型。 `plain` `tls`

对于Kubernetes集群外的客户，您创建的听众和指定的连接机制，它可以是，，或（在OpenShift）。有关用于连接外部客户端的配置选项的更多信息，请参阅配置外部侦听器。 `external` `nodeport` `loadbalancer` `ingress` `route`

支持的身份验证选项：

1. 相互TLS身份验证（仅在启用了TLS加密的侦听器上）
2. SCRAM-SHA-512身份验证
3. 基于OAuth 2.0令牌的身份验证

您选择的身份验证选项取决于您希望如何验证对Kafka代理的客户端访问权限。

图3. Kafka侦听器身份验证选项

listener 属性用于指定特定于该侦听器的身份验证机制。 authentication

侦听器身份验证机制由字段定义。如果未指定任何属性，则侦听器不会验证通过该侦听器连接的客户端。侦听器无需身份验证即可接受所有连接。 type authentication

使用用户 Operator 进行管理时，必须配置身份验证。 KafkaUsers

以下示例显示：

- 为SCRAM-SHA-512身份验证配置的侦听器 plain
- 一个具有相互TLS身份验证监听器 tls
- 具有双向TLS身份验证的侦听器 external

显示侦听器身份验证配置的示例

```
# ...
listeners:
  plain:
    authentication:
      type: scram-sha-512
  tls:
    authentication:
      type: tls
  external:
    type: loadbalancer
    tls: true
    authentication:
      type: tls
# ...
```

相互TLS身份验证

相互TLS身份验证始终用于Kafka代理与ZooKeeper吊舱之间的通信。

Strimzi可以将Kafka配置为使用TLS（传输层安全性）来提供Kafka代理与客户端之间的加密通信，无论有无相互认证。对于双向或双向身份验证，服务器和客户端均提供证书。配置相互身份验证时，代理对客户端进行身份验证（客户端身份验证），客户端对代理进行身份验证（服务器身份验证）。

注意	TLS身份验证通常是一种单向方式，其中一方认证另一方的身份。例如，当在Web浏览器和Web服务器之间使用HTTPS时，浏览器将获得Web服务器身份的证明。

SCRAM-SHA-512身份验证

SCRAM（盐分挑战响应认证机制）是一种认证协议，可以使用密码建立相互认证。Strimzi可以将Kafka配置为使用SASL（简单身份验证和安全层）SCRAM-SHA-512在未加密和加密的客户端连接上提供身份验证。

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

SCRAM的以下属性即使在未加密的连接上也可以安全地使用SCRAM-SHA-512：

- 密码不是通过通信通道明文发送的。取而代之的是，客户端和服务器各自受到对方的挑战，以提供证明他们知道身份验证用户密码的证据。
- 服务器和客户端各自为每次身份验证交换生成新的挑战。这意味着交换可以抵抗重放攻击。

当用户 Operator 配置了a时，它将生成一个随机的12个字符的密码，由大写和小写ASCII字母和数字组成。 KafkaUser.spec.authentication.type scram-sha-512

网络政策

Strimzi自动为在Kafka代理上启用的每个侦听器创建资源。默认情况下，a 授予对所有应用程序和名称空间的侦听器的访问。 NetworkPolicy NetworkPolicy

如果要将网络级别的侦听器访问限制为仅选定的应用程序或名称空间，请使用该属性。 networkPolicyPeers

将网络策略用作侦听器身份验证配置的一部分。每个侦听器可以具有不同的配置。 networkPolicyPeers

有关更多信息，请参考Listener网络策略部分和NetworkPolicyPeer API参考。

注意	您的Kubernetes配置必须支持入口才能在Strimzi中使用网络策略。 NetworkPolicies

4.1.2。Kafka授权

您可以使用资源中的属性配置Kafka代理的授权。如果缺少该属性，则不会启用任何授权，并且客户端没有任何限制。启用后，授权将应用于所有启用的侦听器。授权方法在该字段中定义。 authorization Kafka.spec.kafka authorization type

支持的授权选项：

- 简单授权
- OAuth 2.0授权 （如果您使用的是基于OAuth 2.0令牌的身份验证）
- 开放策略代理（OPA）授权

图4. Kafka集群授权选项

超级用户

超级用户可以访问Kafka群集中的所有资源，而不受任何访问限制，并且所有授权机制都支持这些超级用户。

要为Kafka集群指定超级用户，请在属性中添加用户主体列表。如果用户使用TLS客户端身份验证，则其用户名是其证书 Topic 中以开头的通用名称。超级用户的示例配置 `superUsers` `CN=`

```
authorization:
  type: simple
  superUsers:
    - CN=client_1
    - user_2
    - CN=client_3
```

4.2。Kafka客户端的安全选项

使用该资源为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，客户端代表用户。 `KafkaUser`

您可以验证和授权用户对Kafka经纪人的访问。身份验证允许访问，而授权将访问限制为允许的操作。

您还可以创建对Kafka代理具有不受限制的访问权限的*超级用户*。

身份验证和授权机制必须匹配用于访问Kafka代理的侦听器的规范。

4.2.1。确定用于用户处理的Kafka集群

甲资源包括标签定义Kafka簇（来自名称衍生的合适的名称它属于哪个资源）。 `KafkaUser` `Kafka`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
```

用户 Operator 使用该标签来标识资源并创建新用户，以及随后对用户的处理。 `KafkaUser`

如果标签与Kafka群集不匹配，则用户 Operator 无法识别，并且不会创建用户。 `KafkaUser`

如果资源的状态保持为空，请检查标签。 `KafkaUser`

4.2.2。用户认证

使用中的属性配置用户身份验证。使用该字段指定为用户启用的身份验证机制。 `authentication` `KafkaUser.spec` `type`

支持的身份验证机制：

- TLS客户端身份验证
- SCRAM-SHA-512身份验证

如果未指定身份验证机制，则用户 Operator 不会创建用户或其凭据。
额外资源

- 何时对客户端使用双向TLS身份验证或SCRAM-SHA身份验证

TLS客户端身份验证

要使用TLS客户端身份验证，请将字段设置为。启用TLS客户端身份验证的示例 `type` `tls` `KafkaUser`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: tls
  # ...
```

由用户 Operator 创建用户时，它将创建一个与资源同名的新密钥。机密包含用于TLS客户端身份验证的私钥和公钥。公钥包含在用户证书中，该证书由客户端证书颁发机构（CA）签名。 `KafkaUser`

所有密钥均为X.509格式。

机密提供PEM和PKCS＃12格式的私钥和证书。

有关通过“秘密”保护Kafka通信的更多信息，请参阅“ 安全性”。具有用户凭证的示例
Secret

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

SCRAM-SHA-512认证

要使用SCRAM-SHA-512身份验证机制，请将字段设置为。启用SCRAM-SHA-512身份验证的示例 type scram-sha-512
KafkaUser

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: scram-sha-512
  # ...
```

由用户 Operator 创建用户时，它将创建一个与资源同名的新机密。机密在密钥中包含生成的密码，该密码使用base64编码。为了使用密码，必须对其进行解码。具有用户凭证的示例 KafkaUser password
Secret

```
apiVersion: v1
kind: Secret
metadata:
  name: my-user
  labels:
    strimzi.io/kind: KafkaUser
    strimzi.io/cluster: my-cluster
type: Opaque
data:
  password: Z2VuZXJhdGVkcGFzc3dvcmQ= (1)
```

1. 生成的密码，以base64编码。

解码生成的密码：

```
“ Z2VuZXJhdGVkcGFzc3dvcmQ =” | base64-
```

4.2.3。用户授权

使用中的属性配置用户授权。使用该字段指定为用户启用的授权类型。 authorization KafkaUser.spec type

要使用简单授权，请将属性设置为in 。简单授权使用默认的Kafka授权插件。 type simple KafkaUser.spec.authorization AclAuthorizer

或者，您可以使用OPA授权，或者如果您已经在使用基于OAuth 2.0令牌的身份验证，则还可以使用OAuth 2.0授权。

如果未指定授权，则用户 Operator 不会为用户提供任何访问权限。这样的服务器是否仍然可以访问资源取决于所使用的授权者。例如，为此由其配置确定。 KafkaUser AclAuthorizer allow.everyone.if.no.acl.found

ACL规则

AclAuthorizer 使用ACL规则来管理对Kafka经纪人的访问。

ACL规则授予您在属性中指定的用户访问权限。 acls

有关对象的更多信息，请参见模式参考。 AclRule AclRule

超级用户对Kafka经纪人的访问

如果在Kafka代理配置中将用户添加到超级用户列表中，则该用户将不受限制地访问群集，而不管中的ACL中定义的任何授权约束。 KafkaUser

有关配置对代理的超级用户访问权限的更多信息，请参阅Kafka授权。

用户配额

您可以为资源配置，以强制执行配额，以使用户不会超过基于字节阈值或CPU利用率时间限制的对Kafka代理的访问权限。用户配额示例 spec KafkaUser
KafkaUser

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  # ...
  quotas:
    producerByteRate: 1048576 (1)
    consumerByteRate: 2097152 (2)
    requestPercentage: 55 (3)
```

1. 用户可以推送到Kafka代理的数据量的每秒字节配额
2. 用户可以从Kafka代理获取的数据量的每秒字节配额
3. 客户端组的CPU利用率限制（以时间百分比表示）

有关这些属性的更多信息，请参见模式参考。 KafkaUserQuotas

4.3。确保访问Kafka经纪人

要建立对Kafka代理的安全访问，请配置并应用：

- 一个资源： Kafka
  - 创建具有指定身份验证类型的侦听器
  - 配置整个Kafka集群的授权
- 一个通过监听器安全地访问Kafka经纪人资源 KafkaUser

配置要设置的资源： Kafka

- 侦听器身份验证
- 限制访问Kafka侦听器的网络策略
- Kafka授权
- 超级用户可不受限制地访问经纪人

身份验证是为每个侦听器独立配置的。始终为整个Kafka集群配置授权。

群集 Operator 创建侦听器并设置群集和客户端证书颁发机构（CA）证书，以在Kafka群集中启用身份验证。

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

使用启用身份验证和授权机制，一个特定的客户端用来访问Kafka。 KafkaUser

配置要设置的资源： KafkaUser

- 身份验证以匹配启用的侦听器身份验证
- 授权以匹配启用的Kafka授权
- 控制客户端资源使用的配额

用户 Operator 根据所选的身份验证类型，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。
额外资源

有关以下架构的更多信息：

- Kafka，请参阅架构参考。 Kafka
- KafkaUser，请参阅架构参考。 KafkaUser

4.3.1。保护Kafka经纪人

此过程显示了在运行Strimzi时保护Kafka代理安全的步骤。

为Kafka经纪人实施的安全性必须与为需要访问的客户端实施的安全性兼容。

- Kafka.spec.kafka.listeners.*.authentication 火柴 KafkaUser.spec.authentication
- Kafka.spec.kafka.authorization 火柴 KafkaUser.spec.authorization

这些步骤显示了使用TLS身份验证的简单授权和侦听器的配置。有关侦听器配置的更多信息，请参见模式参考。 KafkaListeners

或者，您可以将SCRAM-SHA或OAuth 2.0用于侦听器身份验证，将OAuth 2.0或OPA用于Kafka授权。程序

1. 配置资源。 `Kafka`

    a. 配置用于授权的属性。 `authorization`
    b. 配置属性以创建具有身份验证的侦听器。 `listeners`

    例如：

    ```
    apiVersion: kafka.strimzi.io/v1beta1
    kind: Kafka
    spec:
      kafka:
        # ...
        authorization: (1)
          type: simple
          superUsers: (2)
            - CN=client_1
            - user_2
            - CN=client_3
        listeners:
          tls:
            authentication:
              type: tls (3)
        # ...
      zookeeper:
        # ...
    ```

    i. 授权使用Kafka插件在Kafka代理上启用授权。 `simple` `AclAuthorizer`
    ii. 无限制访问Kafka的用户主体列表。使用TLS身份验证时，*CN*是客户端证书中的通用名称。
    iii. 可以为每个侦听器配置侦听器身份验证机制，并将其指定为相互TLS或SCRAM-SHA。

    如果要配置外部侦听器，则配置取决于所选的连接机制。
2. 创建或更新资源。 `Kafka`

    ```
    kubectl apply -f KAFKA-CONFIG-FILE
    ```

    Kafka群集使用TLS身份验证配置了Kafka代理侦听器。

    将为每个Kafka经纪人吊舱创建一个服务。

    创建一个服务作为连接到Kafka群集的*引导地址*。

    还将使用与资源相同的名称创建用于验证kafka代理身份的群集CA证书。 `Kafka`

4.3.2。确保用户对Kafka的访问

使用资源的属性来配置Kafka用户。 `KafkaUser`

您可以用来创建或修改用户，以及删除现有用户。 `kubectl apply` `kubectl delete`

例如：

- `kubectl apply -f USER-CONFIG-FILE`
- `kubectl delete KafkaUser USER-NAME`

配置身份验证和授权机制时，请确保它们与等效配置匹配： `KafkaUser` `Kafka`

- `KafkaUser.spec.authentication` 火柴 `Kafka.spec.kafka.listeners.*.authentication`
- `KafkaUser.spec.authorization` 火柴 `Kafka.spec.kafka.authorization`

此过程显示了如何使用TLS身份验证创建用户。您也可以使用SCRAM-SHA身份验证创建用户。

所需的身份验证取决于为Kafka代理侦听器配置的身份验证类型。

注意	Kafka用户和Kafka代理之间的身份验证取决于各自的身份验证设置。例如，如果还没有在Kafka配置中启用TLS，则无法使用TLS对用户进行身份验证。

先决条件

- 正在运行的Kafka群集，该群集配置了使用TLS身份验证和加密的Kafka代理侦听器。
- 正在运行的用户 Operator （通常与实体 Operator 一起部署）。

输入的身份验证类型应与代理中配置的身份验证相匹配。程序 `KafkaUser` `Kafka`

1. 配置资源。 `KafkaUser`

    例如：

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication: (1)
    type: tls
  authorization:
    type: simple (2)
    acls:
      - resource:
          type: topic
          name: my-topic
          patternType: literal
        operation: Read
      - resource:
          type: topic
          name: my-topic
          patternType: literal
        operation: Describe
      - resource:
          type: group
          name: my-group
          patternType: literal
        operation: Read
```

a. 用户身份验证机制，定义为或。 `tls` `scram-sha-512`
b. 简单授权，需要随附ACL规则列表。

2. 创建或更新资源。 `KafkaUser`

```
kubectl apply -f USER-CONFIG-FILE
```

将创建用户以及与资源同名的Secret 。机密包含用于TLS客户端身份验证的私钥和公钥。 `KafkaUser`

有关使用属性配置Kafka客户端以安全连接到Kafka代理的信息，请参阅*Deploying Strimzi Guide*中的设置Kubernetes以外的客户端的访问权限。

4.3.3。使用网络策略限制对Kafka侦听器的访问

您可以使用该字段将访问者的访问权限限制为仅选定的应用程序。先决条件 `networkPolicyPeers`

- 一个支持Ingress NetworkPolicies的Kubernetes集群
- 群集 Operator 正在运行。

程序

1. 打开资源。 `Kafka`
2. 在该字段中，定义将被允许访问Kafka集群的应用程序容器或名称空间。 `networkPolicyPeers`

   例如，将侦听器配置为仅允许来自标签设置为的应用程序Pod进行连接： `tls` `app` `kafka-client`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  kafka:
    # ...
    listeners:
      tls:
        networkPolicyPeers:
          - podSelector:
              matchLabels:
                app: kafka-client
    # ...
  zookeeper:
    # ...
```

3. 创建或更新资源。

   用途： `kubectl apply`

```
kubectl apply -f your-file
```

额外资源

- 有关模式的更多信息，请参见NetworkPolicyPeer API参考和模式参考。 `KafkaListeners`

4.4。使用基于OAuth 2.0令牌的身份验证

Strimzi支持通过*SASL OAUTHBEARER*机制使用OAuth 2.0身份验证。

OAuth 2.0使用中央授权服务器来发布授予对资源的有限访问权限的令牌，从而在应用程序之间实现基于令牌的标准化身份验证和授权。

您可以先配置OAuth 2.0身份验证，然后再配置OAuth 2.0授权。

OAuth 2.0身份验证也可以与基于OPA的Kafka授权结合使用。 `simple`

使用基于OAuth 2.0令牌的身份验证，应用程序客户端可以访问应用程序服务器（称为*资源服务器*）上的*资源*，而无需暴露帐户凭据。

应用程序客户端传递访问令牌作为身份验证的方法，应用程序服务器也可以使用该令牌来确定授予的访问级别。授权服务器处理访问的授予和有关访问的查询。

在Strimzi中：

- Kafka代理充当OAuth 2.0资源服务器
- Kafka客户端充当OAuth 2.0应用程序客户端

Kafka客户向Kafka经纪人进行身份验证。代理和客户端根据需要与OAuth 2.0授权服务器进行通信，以获取或验证访问令牌。

对于Strimzi的部署，OAuth 2.0集成提供了：

- 服务器端OAuth 2.0对Kafka代理的支持
- 对Kafka MirrorMaker，Kafka Connect和Kafka Bridge的客户端OAuth 2.0支持

额外资源

- OAuth 2.0网站

4.4.1。OAuth 2.0身份验证机制

Kafka *SASL OAUTHBEARER*机制用于与Kafka代理建立经过身份验证的会话。

Kafka客户端使用*SASL OAUTHBEARER*机制启动与Kafka代理的会话以进行凭据交换，其中凭据采用访问令牌的形式。

Kafka代理和客户端需要配置为使用OAuth 2.0。

4.4.2。OAuth 2.0 Kafka代理配置

OAuth 2.0的Kafka代理配置涉及：

- 在授权服务器中创建OAuth 2.0客户端
- 在Kafka自定义资源中配置OAuth 2.0身份验证

注意	关于授权服务器，Kafka代理和Kafka客户端均被视为OAuth 2.0客户端。

授权服务器上的OAuth 2.0客户端配置

要配置Kafka代理以验证在会话发起期间接收到的令牌，建议的方法是在授权服务器中创建OAuth 2.0 *客户端*定义，该服务器配置为*机密*，并启用以下客户端凭据：

- 的客户ID （例如） `kafka`
- 客户端ID和密钥作为身份验证机制

注意	使用授权服务器的非公共自省端点时，仅需要使用客户端ID和密码。与快速本地JWT令牌验证一样，使用公共授权服务器端点时通常不需要凭据。

Kafka群集中的OAuth 2.0身份验证配置

要在Kafka群集中使用OAuth 2.0身份验证，您可以使用以下身份验证方法为Kafka群集自定义资源指定TLS侦听器配置：关联OAuth 2.0的身份验证方法类型 `oauth`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  kafka:
    listeners:
      tls:
        authentication:
          type: oauth
          #...
```

您可以配置，和听众，但建议不要使用监听器或禁用了TLS加密听众使用OAuth 2.0，因为这造成网络窃听和通过令牌盗窃未经授权的访问中的漏洞。 `plair` `tls` `external` `plain` `external`

您可以使用侦听器为安全传输层配置以与客户端进行通信。将OAuth 2.0与外部侦听器一起使用 `external` `type: oauth`

```
# ...
listeners:
  tls:
    authentication:
      type: oauth
  external:
    type: loadbalancer
    tls: true
    authentication:
      type: oauth
    #...
```

该属性默认为*true*，因此可以省略。 `tls`

当你定义的身份验证类型的OAuth 2.0，您添加基于验证类型的配置，无论是作为快速的本地JWT验证或使用内省端点令牌验证。

在配置Kafka代理的OAuth 2.0支持中介绍了为侦听器配置OAuth 2.0的过程（包括描述和示例）。

快速本地JWT令牌验证配置

快速本地JWT令牌验证在本地检查JWT令牌签名。

本地检查可确保令牌：

- 通过包含访问令牌的（*typ*）声明值来符合类型 `Bearer`
- 有效（未过期）
- 有一个与 `validIssuerURI`

在配置侦听器时，可以指定一个属性，以便拒绝未由授权服务器发行的所有令牌。 `validIssuerURI`

快速本地JWT令牌验证期间不需要联系授权服务器。您可以通过指定属性（OAuth 2.0授权服务器公开的端点）来激活快速本地JWT令牌验证。端点包含用于验证签名的JWT令牌的公钥，这些公钥由Kafka客户端作为凭据发送。 `jwksEndpointUri`

注意	与授权服务器的所有通信均应使用TLS加密执行。

您可以在Strimzi项目名称空间中将证书信任库配置为Kubernetes Secret，并使用属性指向包含信任库文件的Kubernetes Secret。 `tlsTrustedCertificates`

您可能需要配置A，以从JWT令牌中正确提取用户名。如果要使用Kafka ACL授权，则需要在身份验证期间通过用户名来标识用户。（JWT令牌中的声明通常是唯一ID，而不是用户名。）快速本地JWT令牌验证的示例配置 `userNameClaim` `sub`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  kafka:
    listeners:
      tls:
        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
```

OAuth 2.0自省端点配置

使用OAuth 2.0自省端点进行的令牌验证会将收到的访问令牌视为不透明。Kafka代理将访问令牌发送到自省端点，该端点将以验证所需的令牌信息进行响应。重要的是，如果特定的访问令牌有效，它将返回最新信息，以及有关令牌何时过期的信息。

要配置基于OAuth 2.0自省的验证，请指定一个属性，而不是为快速本地JWT令牌验证指定的属性。取决于授权服务器，您通常必须指定和，因为自省端点通常受到保护。自省端点的示例配置 `introspectionEndpointUri` `jwksEndpointUri` `clientId` `clientSecret`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  kafka:
    listeners:
      tls:
        authentication:
          type: oauth
          clientId: kafka-broker
          clientSecret:
            secretName: my-cluster-oauth
            key: clientSecret
          validIssuerUri: <https://<auth-server-address>/auth/realms/tls>
          introspectionEndpointUri: <https://<auth-server-address>/auth/realms/tls/protocol/openid-connect/token
/introspect>
          userNameClaim: preferred_username
          maxSecondsWithoutReauthentication: 3600
          tlsTrustedCertificates:
          - secretName: oauth-server-cert
            certificate: ca.crt
```

4.4.3。Kafka经纪人的会话重新认证

在*Strimzi*中用于OAuth 2.0身份验证的Kafka *SASL OAUTHBEARER*机制支持一种称为*重新身份验证*机制的Kafka功能。

如果在类型侦听器的配置中启用了重新认证机制，则访问令牌过期时，代理的已认证会话也会过期。然后，客户端必须通过向代理发送新的有效访问令牌来重新认证现有会话，而无需断开连接。 `oauth`

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认证，则如果进一步尝试发送或接收消息，则代理将关闭连接。如果在代理上启用了重新认证机制，则使用Kafka客户端库2.2或更高版本的Java客户端会自动重新认证。

您在资源中启用会话重新认证。设置具有身份验证的TLS侦听器的属性。两种令牌验证类型均支持会话重新认证（快速本地JWT和自省端点）。有关示例配置，请参阅为Kafka代理配置OAuth 2.0支持。 `Kafka` `maxSecondsWithoutReauthentication` `type: oauth`

有关在2.2版本的Kafka中添加的有关重新认证机制的更多信息，请参阅KIP-368。额外资源

- OAuth 2.0 Kafka代理配置
- 为Kafka代理配置OAuth 2.0支持
- KafkaListenerAuthenticationOAuth架构参考

4.4.4。OAuth 2.0 Kafka客户端配置

Kafka客户端配置为：

- 从授权服务器获取有效访问令牌所需的凭据（客户端ID和密钥）
- 使用授权服务器提供的工具获得的有效的长期访问令牌或刷新令牌

曾经发送给Kafka经纪人的唯一信息是访问令牌。用于向授权服务器进行身份验证以获取访问令牌的凭据永远不会发送到代理。

当客户端获得访问令牌时，不需要与授权服务器进行进一步的通信。

最简单的机制是使用客户端ID和密码进行身份验证。使用寿命很长的访问令牌或寿命很长的刷新令牌会增加更多的复杂性，因为对授权服务器工具存在额外的依赖性。

注意	如果您使用的是长期访问令牌，则可能需要在授权服务器中配置客户端，以增加令牌的最大生存期。

如果未直接为Kafka客户端配置访问令牌，则在Kafka会话启动期间，客户端会通过与授权服务器联系来交换访问令牌的凭据。Kafka客户交换以下任何一项：

- 客户编号和机密
- 客户端ID，刷新令牌和（可选）秘密

4.4.5。OAuth 2.0客户端身份验证流程

在本节中，我们将说明和可视化Kafka会话启动期间Kafka客户端，Kafka代理和授权服务器之间的通信流程。该流程取决于客户端和服务器配置。

当Kafka客户端将访问令牌作为凭证发送给Kafka代理时，需要对令牌进行验证。

根据使用的授权服务器和可用的配置选项，您可能更喜欢使用：

- 基于JWT签名检查和本地令牌自省的快速本地令牌验证，而无需联系授权服务器
- 授权服务器提供的OAuth 2.0自省端点

使用快速本地令牌验证要求授权服务器向JWKS端点提供公共证书，这些证书用于验证令牌上的签名。

另一个选择是在授权服务器上使用OAuth 2.0自省端点。每次建立新的Kafka代理连接时，代理会将从客户端收到的访问令牌传递给授权服务器，并检查响应以确认该令牌是否有效。

Kafka客户端凭据也可以配置为：

- 使用先前生成的长期访问令牌进行直接本地访问

- 与授权服务器联系以获取新的访问令牌

注意	授权服务器可能只允许使用不透明的访问令牌，这意味着不可能进行本地令牌验证。

客户端身份验证流程示例

在这里，您可以查看Kafka会话身份验证期间不同配置的Kafka客户端和代理的通信流。

- 客户端使用客户端ID和机密，代理将验证委派给授权服务器
- 客户端使用客户端ID和密码，由代理执行快速本地令牌验证
- 客户端使用长期访问令牌，代理将验证委托给授权服务器
- 客户端使用长期访问令牌，并由代理执行快速本地验证

客户端使用客户端ID和机密，代理将验证委派给授权服务器

1. Kafka客户端使用客户端ID和密钥从授权服务器请求访问令牌，还可以选择刷新令牌。
2. 授权服务器生成一个新的访问令牌。
3. Kafka客户端使用*SASL OAUTHBEARER*机制向Kafka代理进行身份验证，以传递访问令牌。
4. Kafka代理使用自己的客户端ID和密码，通过在授权服务器上调用令牌自省端点来验证访问令牌。
5. 如果令牌有效，则建立Kafka客户会话。

客户端使用客户端ID和密码，由代理执行快速本地令牌验证

1. Kafka客户端使用客户端ID和密钥以及可选的刷新令牌从令牌端点向授权服务器进行身份验证。
2. 授权服务器生成一个新的访问令牌。
3. Kafka客户端使用*SASL OAUTHBEARER*机制向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自省功能在本地验证访问令牌。

客户端使用长期访问令牌，代理将验证委托给授权服务器

1. Kafka客户端使用*SASL OAUTHBEARER*机制向Kafka代理进行身份验证，以传递长期访问令牌。
2. Kafka代理使用自己的客户端ID和密码，通过在授权服务器上调用令牌自省端点来验证访问令牌。
3. 如果令牌有效，则建立Kafka客户会话。

客户端使用长期访问令牌，并由代理执行快速本地验证

1. Kafka客户端使用*SASL OAUTHBEARER*机制向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自省功能在本地验证访问令牌。

警告	快速本地JWT令牌签名验证仅适用于短期令牌，因为授权服务器不检查令牌是否已被吊销。令牌到期已写入令牌中，但是吊销可以随时发生，因此如果不联系授权服务器就无法解决。任何发行的令牌在到期之前都将被视为有效。

4.4.6。配置OAuth 2.0身份验证

OAuth 2.0用于Kafka客户端和Strimzi组件之间的交互。

为了对Strimzi使用OAuth 2.0，您必须：

1. 为Strimzi群集和Kafka客户端配置OAuth 2.0授权服务器
2. 使用配置为使用OAuth 2.0的Kafka代理侦听器部署或更新Kafka集群
3. 更新您的基于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

配置OAuth 2.0授权服务器

此过程大体上描述了配置授权服务器以与Strimzi集成所需执行的操作。

这些说明不是特定于产品的。

步骤取决于所选的授权服务器。请查阅授权服务器的产品文档，以获取有关如何设置OAuth 2.0访问权限的信息。

注意	如果已经部署了授权服务器，则可以跳过部署步骤并使用当前的部署。

程序

1. 将授权服务器部署到您的集群。
2. 访问授权服务器的CLI或管理控制台，以为Strimzi配置OAuth 2.0。

   现在，准备授权服务器以与Strimzi一起使用。
3. 配置客户端。 `kafka-broker`

4. 为应用程序的每个Kafka客户端组件配置客户端。

接下来做什么

部署和配置授权服务器后，将Kafka代理配置为使用OAuth 2.0。

为Kafka代理配置OAuth 2.0支持

此过程描述了如何配置Kafka代理，以便使代理侦听器能够通过授权服务器使用OAuth 2.0身份验证。

我们建议通过TLS侦听器的配置在加密接口上使用OAuth 2.0。不推荐普通的侦听器。

如果授权服务器使用由受信任CA签名的证书，并且与OAuth 2.0服务器主机名匹配，则TLS连接将使用默认设置进行工作。否则，您可能需要使用探测器证书配置信任库或禁用证书主机名验证。

在配置Kafka代理时，对于在新连接的Kafka客户端进行OAuth 2.0身份验证期间用于验证访问令牌的机制，您有两个选项：

- 配置快速本地JWT令牌验证
- 使用自省端点配置令牌验证

在你开始前

有关针对Kafka代理侦听器的OAuth 2.0身份验证配置的详细信息，请参阅：

- KafkaListenerAuthenticationOAuth架构参考
- 管理对Kafka的访问

先决条件

- Strimzi和Kafka正在运行
- 部署了OAuth 2.0授权服务器

程序

1. 在编辑器中更新资源的Kafka代理配置（`Kafka.spec.kafka`）。 `Kafka`

   ```
   kubectl edit kafka my-cluster
   ```

2. 配置Kafka代理配置。 `listeners`

   每种类型的侦听器的配置都不必相同，因为它们是独立的。

   此处的示例显示为外部侦听器配置的配置选项。
   示例1：配置快速本地JWT令牌验证

   ```
   external:
     type: loadbalancer
     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)
       enableECDSA: "true" (11)
   ```

   a. 侦听器类型设置为。 `oauth`
   b. 用于认证的令牌发行者的URI。
   c. 用于本地JWT验证的JWKS证书端点的URI。
   d. 在令牌中包含实际用户名的令牌声明（或密钥）。用户名是用于标识用户的*主体*。该值将取决于身份验证流程和使用的授权服务器。 `userNameClaim`
   e. （可选）激活Kafka重认证机制，该机制将会话过期时间强制为与访问令牌相同的时间长度。如果指定的值小于访问令牌剩余的时间，则客户端将必须在实际令牌到期之前重新进行身份验证。默认情况下，访问令牌过期时会话不会过期，并且客户端不会尝试重新认证。
   f. （可选）用于TLS连接到授权服务器的可信证书。
   g. （可选）禁用TLS主机名验证。默认值为。 `false`
   h. JWKS证书在过期之前被视为有效的持续时间。默认值为秒。如果您指定更长的时间，请考虑允许访问已撤销证书的风险。 `360`
   i. 刷新JWKS证书的间隔时间。该间隔必须至少比到期间隔短60秒。默认值为秒。 `300`
   j. 连续尝试刷新JWKS公共密钥之间的最小暂停时间（以秒为单位）。当遇到未知的签名密钥时，JWKS密钥刷新是在常规定期调度之外调度的，至少自上一次刷新尝试以来已指定了暂停时间。键的刷新遵循指数退避的规则，不成功的刷新将随着暂停的增加而重试，直到达到。预设值为1。 `jwksRefreshSeconds`
   k. （可选）如果使用ECDSA在授权服务器上签署JWT令牌，则需要启用它。它使用BouncyCastle加密库安装其他加密提供程序。默认值为。 `false`

示例2：使用自省端点配置令牌验证

```
external:
  type: loadbalancer
  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)
```

a. 令牌自省端点的URI。
b. 用于标识客户端的客户端ID。
c. 客户端密钥和客户端ID用于身份验证。
d. 在令牌中包含实际用户名的令牌声明（或密钥）。用户名是用于标识用户的*主体*。该值取决于所使用的授权服务器。 `userNameClaim`
e. （可选）激活Kafka重认证机制，该机制将会话过期时间强制为与访问令牌相同的时间长度。如果指定的值小于访问令牌剩余的时间，则客户端将必须在实际令牌到期之前重新进行身份验证。默认情况下，访问令牌过期时会话不会过期，并且客户端不会尝试重新认证。

根据您应用OAuth 2.0身份验证的方式以及授权服务器的类型，可以使用其他（可选）配置设置：

```
  # ...
  authentication:
    type: oauth
    # ...
    checkIssuer: false (1)
    fallbackUserNameClaim: client_id (2)
    fallbackUserNamePrefix: client-account- (3)
    validTokenType: bearer (4)
    userInfoEndpointUri: https://OAUTH-SERVER-ADDRESS/auth/realms/external/protocol/openid-connect/userinfo
(5)
```

a. 如果您的授权服务器未提供声明，则无法执行颁发者检查。在这种情况下，请将设置为，而不要指定。默认值为。 `iss` `checkIssue` `false` `validIssuerUri` `true`
b. 授权服务器可能不提供用于标识常规用户和客户端的单个属性。当客户端以其自己的名称进行身份验证时，服务器可能会提供一个*客户端ID*。当用户使用用户名和密码进行身份验证以获取刷新令牌或访问令牌时，服务器可能会在提供客户端ID的同时提供*用户名*属性。使用此后备选项可指定主要用户ID属性不可用时要使用的用户名声明（属性）。
c. 在适用的情况下，可能还需要防止用户名声明的值与后备用户名声明的值之间的名称冲突。考虑一种情况，其中存在一个称为的客户端，但也存在一个称为的常规用户。为了区分两者，您可以使用此属性为客户端的用户ID添加前缀。 `fallbackUserNameClaim` `produ cer` `producer`
d. （仅在使用时适用）根据所使用的授权服务器，自省端点可能会或可能不会返回*令牌类型*属性，或者它可能包含不同的值。您可以指定自省端点的响应必须包含的有效令牌类型值。 `introspectionEndpointUri`
e. （仅在使用时适用）授权服务器可以以在自省端点响应中不提供任何可识别信息的方式配置或实现。为了获取用户ID，您可以将端点的URI配置为后备。的，和设置应用到的响应端点。 `introspectionEndpointUri` `userinfo` `userNameClaim` `fallbackUserNa meClaim` `fallbackUserNamePrefix` `userinfo`

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观察容器状态转换：

```
kubectl logs -f ${POD_NAME} -c ${CONTAINER_NAME}
kubectl get po -w
```

滚动更新将代理配置为使用OAuth 2.0身份验证。

接下来做什么

- 配置您的Kafka客户端以使用OAuth 2.0

配置Kafka Java客户端以使用OAuth 2.0

此过程描述了如何配置Kafka生产者和使用者API，以使用OAuth 2.0与Kafka代理进行交互。

将客户端回调插件添加到*pom.xml*文件，然后配置系统属性。先决条件

- Strimzi和Kafka正在运行
- 部署并配置了OAuth 2.0授权服务器，以对Kafka代理进行OAuth访问
- Kafka代理针对OAuth 2.0配置

程序

1. 将具有OAuth 2.0支持的客户端库添加到Kafka客户端的文件中： `pom.xml`

```
<dependency>
 <groupId>io.strimzi</groupId>
```

```
 <artifactId>kafka-oauth-client</artifactId>
 <version>0.6.0</version>
</dependency>
```

2. 配置回调的系统属性：

   例如：

   ```
   System.setProperty(ClientConfig.OAUTH_TOKEN_ENDPOINT_URI, “https://<auth-server-address>/auth/realms/master
   /protocol/openid-connect/token”); (1)
   System.setProperty(ClientConfig.OAUTH_CLIENT_ID, "<client-name>"); (2)
   System.setProperty(ClientConfig.OAUTH_CLIENT_SECRET, "<client-secret>"); (3)
   ```

   a. 授权服务器令牌端点的URI。
   b. 客户端ID，这是在授权服务器中创建客户端时使用的名称。
   c. 在授权服务器中创建客户端时创建的客户端密码。

3. 在Kafka客户端配置中的TLS加密连接上启用*SASL OAUTHBEARER*机制：

   例如：

   ```
   props.put("sasl.jaas.config", "org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule
   required;");
   props.put("security.protocol", "SASL_SSL"); (1)
   props.put("sasl.mechanism", "OAUTHBEARER");
   props.put("sasl.login.callback.handler.class", "io.strimzi.kafka.oauth.client.
   JaasClientOauthLoginCallbackHandler");
   ```

   a. 在这里，我们用于TLS连接。在未加密的连接上使用。 `SASL_SSL` `SASL_PLAINTEXT`

4. 验证Kafka客户端可以访问Kafka代理。

接下来做什么

- 配置Kafka组件以使用OAuth 2.0

为Kafka组件配置OAuth 2.0

此过程介绍了如何配置Kafka组件以通过授权服务器使用OAuth 2.0身份验证。

您可以为以下配置身份验证：

- Kafka Connect
- KafkaMirrorMaker
- Kafka桥

在这种情况下，Kafka组件和授权服务器在同一群集中运行。
在你开始前

有关针对Kafka组件的OAuth 2.0身份验证配置的详细信息，请参阅：

- KafkaClientAuthenticationOAuth架构参考

先决条件

- Strimzi和Kafka正在运行
- 部署并配置了OAuth 2.0授权服务器，以对Kafka代理进行OAuth访问
- Kafka代理针对OAuth 2.0配置

程序

1. 创建一个客户端密钥，并将其作为环境变量安装到组件。

   例如，在这里我们为Kafka桥创建一个客户端： `Secret`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Secret
   metadata:
    name: my-bridge-oauth
   type: Opaque
   data:
    clientSecret: MGQ1OTRmMzYtZTllZS00MDY2LWI5OGEtMTM5MzM2NjdlZjQw (1)
   ```

   a. 该密钥必须以Base64格式。 `clientSecret`

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：

   - 客户编号和机密
   - 客户端ID和刷新令牌
   - 访问令牌
   - TLS

KafkaClientAuthenticationOAuth模式参考提供了每个示例。

例如，此处使用客户端ID和密钥以及TLS将OAuth 2.0分配给Kafka Bridge客户端：

```
apiVersion: kafka.strimzi.io/v1beta1
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
```

a. 身份验证类型设置为。 `oauth`
b. 用于认证的令牌端点的URI。
c. TLS连接到授权服务器的受信任证书。

根据您应用OAuth 2.0身份验证的方式以及授权服务器的类型，可以使用其他配置选项：

```
# ...
spec:
  # ...
  authentication:
    # ...
    disableTlsHostnameVerification: true (1)
    checkAccessTokenType: false (2)
    accessTokenIsJwt: false (3)
    scope: any (4)
```

a. （可选）禁用TLS主机名验证。默认值为。 `false`
b. 如果授权服务器未在JWT令牌内返回（类型）声明，则可以申请跳过令牌类型检查。默认值为。 `typ` `checkAccessTokenType: false` `true`
c. 如果您使用的是不透明令牌，则可以申请使访问令牌不被视为JWT令牌。 `accessTokenIsJwt: false`
d. （可选）用于从令牌端点请求令牌。授权服务器可能要求客户端指定范围。在这种情况下是。 `scope` `any`

3. 将更改应用于Kafka资源的部署。

```
kubectl apply -f your-file
```

4. 检查日志中的更新或通过观察容器状态转换：

```
kubectl logs -f ${POD_NAME} -c ${CONTAINER_NAME}
kubectl get pod -w
```

滚动更新将组件配置为使用OAuth 2.0身份验证与Kafka代理进行交互。

4.4.7。授权服务器示例

选择授权服务器时，请考虑最能支持所选身份验证流程配置的功能。

为了使用Strimzi测试OAuth 2.0，将Keycloak和ORY Hydra用作OAuth 2.0授权服务器。

有关更多信息，请参见：

- 使用OAuth 2.0的Kafka身份验证
- 将Keycloak用作OAuth 2.0授权服务器
- 将Hydra用作OAuth 2.0授权服务器

4.5。使用基于OAuth 2.0令牌的授权

如果您将OAuth 2.0与Keycloak一起用于基于令牌的身份验证，则还可以使用Keycloak来配置授权规则，以限制客户端对Kafka代理的访问。身份验证建立用户的身份。授权决定该用户的访问级别。

Strimzi支持通过Keycloak 授权服务使用基于OAuth 2.0令牌的授权，该服务使您可以集中管理安全策略和权限。

Keycloak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访问以对Kafka代理执行特定操作的策略进行匹配。

默认情况下，Kafka允许所有用户完全访问代理，还提供插件以根据访问控制列表（ACL）配置授权。 `AclAuthorizer`

ZooKeeper存储ACL规则，这些规则基于*username*授予或拒绝对资源的访问。但是，带有Keycloak的基于OAuth 2.0令牌的授权为您希望如何实现对Kafka代理的访问控制提供了更大的灵活性。此外，您可以将Kafka代理配置为使用OAuth 2.0授权和ACL。额外资源

- 使用基于OAuth 2.0令牌的身份验证
- ACL授权
- Keycloak文档

4.5.1。OAuth 2.0授权机制

Strimzi中的OAuth 2.0授权使用Keycloak服务器授权服务REST端点，通过在特定用户上应用定义的安全策略，并为该用户提供在不同资源上授予的权限列表，来扩展与Keycloak的基于令牌的身份验证。策略使用角色和组来匹配用户权限。OAuth 2.0授权基于从Keycloak授权服务接收到的用户授予列表，在本地实施权限。

Kafka经纪人定制授权人

Strimzi 提供了Keycloak *授权者*（）。为了能够将Keycloak REST端点用于Keycloak提供的授权服务，请在Kafka代理上配置自定义授权者。 `KeycloakRBACAuthorizer`

授权者根据需要从授权服务器中获取已授予的权限列表，并在Kafka Broker上本地执行授权，从而为每个客户端请求做出快速的授权决策。

4.5.2。配置OAuth 2.0授权支持

此过程介绍了如何配置Kafka代理以使用通过Keycloak授权服务进行的OAuth 2.0授权。
在你开始之前

考虑您需要或想要限制某些用户的访问权限。您可以结合使用Keycloak *组*，*角色*，*客户端*和*用户*来配置Keycloak中的访问。

通常，组用于根据组织部门或地理位置来匹配用户。角色用于根据用户的功能来匹配用户。

使用Keycloak，您可以在LDAP中存储用户和组，而不能以这种方式存储客户端和角色。存储和访问用户数据可能是您选择配置授权策略的一个因素。

注意	超级用户 始终可以不受限制地访问Kafka代理，无论在Kafka代理上实施的授权如何。

先决条件

- 必须将Strimzi配置为使用OAuth 2.0和Keycloak进行基于令牌的身份验证。设置授权时，可以使用相同的Keycloak服务器端点。
- OAuth 2.0身份验证必须配置有启用重新身份验证的选项。 `maxSecondsWithoutReauthentication`
- 您需要了解如何管理Keycloak授权服务的策略和权限，如Keycloak文档中所述。

程序

1. 访问Keycloak管理控制台或使用Keycloak 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. 使用授权服务可以为客户端定义资源，授权范围，策略和权限。
3. 通过为用户和客户端分配角色和组，将权限绑定到用户和客户端。
4. 通过在编辑器中更新资源的Kafka代理配置（`Kafka.spec.kafka`），将Kafka代理配置为使用Keycloak授权。 `Kafka`

   ```
   kubectl edit kafka my-cluster
   ```

5. 配置Kafka代理配置以使用授权，并能够访问授权服务器和授权服务。 `kafka` `keycloak`

   例如：

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   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)
     #...
   ```

   a. 类型启用Keycloak授权。 `keycloak`
   b. Keycloak令牌端点的URI。对于生产，请始终使用HTTP。
   c. 启用了授权服务的Keycloak中的OAuth 2.0客户端定义的客户端ID。通常，用作ID。 `kafka`
   d. （可选）如果Keycloak授权服务策略拒绝访问，则将授权委派给Kafka 。默认值为。 `AclAuthorizer` `false`
   e. （可选）禁用TLS主机名验证。默认值为。 `false`
   f. （可选）指定的超级用户。
   g. （可选）用于TLS连接到授权服务器的可信证书。
   h. （可选）两次连续授予刷新之间的时间间隔。这是活动会话在Keycloak上检测用户的任何权限更改的最长时间。默认值为60。
   i. （可选）用于刷新（并行）活动会话的授予的线程数。预设值为5。
6. 保存并退出编辑器，然后等待滚动更新完成。

7. 检查日志中的更新或通过观察容器状态转换：

```
kubectl logs -f ${POD_NAME} -c kafka
kubectl get po -w
```

滚动更新将代理配置为使用OAuth 2.0授权。
8. 通过以具有特定角色的客户端或用户身份访问Kafka代理来验证配置的权限，确保他们具有必要的访问权限，或者没有他们不应该拥有的访问权限。

5.使用Strimzi operator

使用Strimzi operator 来管理您的Kafka集群以及Kafka Topic 和用户。

5.1。使用集群 operator

群集操作器用于部署Kafka群集和其他Kafka组件。

群集 Operator 是使用YAML安装文件部署的。

有关部署群集 Operator 的信息，请参阅《*部署Strimzi*》指南中的“ *部署*群集 Operator ”。

有关可用于Kafka的部署选项的信息，请参阅Kafka群集配置。

注意	在OpenShift上，Kafka Connect部署可以合并Source2Image功能，以提供添加其他连接器的便捷方法。

5.1.1。集群 Operator 配置

可以通过以下受支持的环境变量和日志记录配置来配置群集 Operator 。

STRIMZI_NAMESPACE operator 应在其中操作的名称空间的逗号分隔列表。如果未设置，则设置为空字符串，或者设置为*集群 operator 将在所有名称空间中操作。集群 Operator 部署可以使用Kubernetes Downward API将其自动设置为部署集群 Operator 的名称空间。请参见以下示例：env：-名称：STRIMZI_NAMESPACE valueFrom：fieldRef：fieldPath：metadata.namespaceSTRIMZI_FULL_RECONCILIATION_INTERVAL_MS可选，默认值为120000 ms。定期对帐之间的间隔（以毫秒为单位）。STRIMZI_OPERATION_TIMEOUT_MS可选，默认值为300000 ms。内部操作的超时时间（以毫秒为单位）。在常规Kubernetes操作花费比平时更长的集群上使用Strimzi时，应增加该值（由于Docker镜像下载缓慢，例如）。STRIMZI_KAFKA_IMAGES为必填项。这提供了从Kafka版本到包含该版本的Kafka代理的对应Docker镜像的映射。所需的语法是空格或逗号分隔的<version> = <image>对。例如2.5.0 = strimzi / kafka：latest-kafka-2.5.0、2.6.0 = strimzi / kafka：latest-kafka-2.6.0。当指定了Kafka.spec.kafka.version属性但未指定Kafka.spec.kafka.image属性时使用此属性，如Container images.STRIMZI_DEFAULT_KAFKA_INIT_IMAGE所述，可选，默认为strimzi / operator：latest。如果在容器镜像中未将图像指定为kafka-init-image，则在初始代理工作开始于代理程序（即机架支持）之前启动的初始容器的默认镜像名称。STRIMZI_KAFKA_CONNECT_IMAGESRequired。这提供了从Kafka版本到包含该版本的Kafka连接的相应Docker镜像的映射。所需的语法是空格或逗号分隔的<version> = <image>对。例如2.5.0 = strimzi / kafka：latest-kafka-2.5.0、2.6.0 = strimzi / kafka：latest-kafka-2.6.0。如容器图像.STRIMZI_KAFKA_CONNECT_S2I_IMAGESRequired中所述，当指定KafkaConnect.spec.version属性但未指定KafkaConnect.spec.image属性时使用此属性。这提供了从Kafka版本到包含该版本的Kafka连接的相应Docker镜像的映射。所需的语法是空格或逗号分隔的<version> = <image>对。例如2.5.0 = strimzi / kafka：latest-kafka-2.5.0、2.6.0 = strimzi / kafka：latest-kafka-2.6.0。当KafkaConnectS2I.spec时使用。如Container images.STRIMZI_KAFKA_MIRROR_MAKER_IMAGESRequired中所述，指定了version属性，但未指定KafkaConnectS2I.spec.image。这提供了从Kafka版本到包含该版本的Kafka镜像制造商的相应Docker镜像的映射。所需的语法是空格或逗号分隔的<version> = <image>对。例如2.5.0 = strimzi / kafka：latest-kafka-2.5.0、2.6.0 = strimzi / kafka：latest-kafka-2.6.0。如指定image.STRIMZI_DEFAULT_TOPIC_OPERATOR_IMAGE可选，默认为strimzi / operator：latest，指定了KafkaMirrorMaker.spec.version属性但未指定KafkaMirrorMaker.spec.image属性时使用此属性。如果未将任何图像指定为Kafka.spec.entityOperator.topicOperator，则在部署 Topic operator 时用作默认图像名称。Kafka资源的容器镜像中的镜像。STRIMZI_DEFAULT_USER_OPERATOR_IMAGE可选，默认为strimzi / operator：latest。如果未在Kafka资源的容器镜像中将Kafka.spec.entityOperator.userOperator.image指定为镜像，则在部署用户 Operator 时用作默认镜像的镜像名。STRIMZI_DEFAULT_TLS_SIDECAR_ENTITY_OPERATOR_IMAGE可选，默认为strimzi / kafka：latest-kafka- 2.6.0。如果未在容器images.STRIMZI_IMAGE_PULL_POLICY可选中为Kafka.spec.entityOperator.tlsSidecar.image指定任何图像，则在部署为Entity Operator提供TLS支持的sidecar容器时用作默认图像名称。ImagePullPolicy，它将应用于由Strimzi Cluster Operator管理的所有吊舱中的容器。有效值为Always，IfNotPresent和Never。如果未指定，将使用Kubernetes默认值。更改策略将导致您的所有Kafka，Kafka Connect和Kafka MirrorMaker群集的滚动更新。STRIMZI_IMAGE_PULL_SECRETS可选。以逗号分隔的“秘密”名称列表。此处引用的机密包含从中提取容器镜像的容器注册表的凭据。机密信息在imagePullSecrets字段中用于集群 operator 创建的所有Pod。更改此列表会导致您的所有Kafka，Kafka Connect和Kafka MirrorMaker群集的滚动更新。STRIMZI_KUBERNETES_VERSION可选。覆盖从API服务器检测到的Kubernetes版本信息。请参见下面的示例：env：-名称：STRIMZI_KUBERNETES_VERSION值：| 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 / amd64KUBERNETES_SERVICE_DNS_DOMAIN可选。覆盖默认的Kubernetes DNS域名后缀。默认情况下，在Kubernetes集群中分配的服务具有使用默认后缀cluster.local的DNS域名，例如，对于代理kafka-0：<cluster-name> -kafka-0 。<群集名称> -kafka-brokers。<名称空间> .svc.cluster.local将DNS域名添加到用于主机名验证的Kafka代理证书中。如果在群集中使用其他DNS域名后缀，请更改KUBERNETES_SERVICE_DNS_DOMAIN环境变量，从默认值到您要使用的环境变量，以便与Kafka代理建立连接。

通过ConfigMap配置

集群 Operator 的日志记录由配置。 `strimzi-cluster-operator` `ConfigMap`

安装群集 Operator 时，将创建一个包含日志记录的配置。这是文件中的描述。您可以通过更改this中的数据字段来配置集群 Operator 日志记录。 `ConfigMap` `ConfigMap` `install/cluster-operator/050-ConfigMap-strimzi-cluster-operator.yaml` `log4j2.properties` `ConfigMap`

要更新日志记录配置，可以编辑文件，然后运行以下命令： `050-ConfigMap-strimzi-cluster-operator.yaml`

```
kubectl apply -f install/cluster-operator/050-ConfigMap-strimzi-cluster-operator.yaml
```

或者，直接编辑： `ConfigMap`

```
kubectl edit cm strimzi-cluster-operator
```

要更改重新加载间隔的频率，请在created 中的选项中设置以秒为单位的时间。 `monitorInterval` `ConfigMap`

如果在部署Cluster Operator时缺少，则使用默认日志记录值。 ConfigMap

如果在部署集群 Operator 之后意外删除了，则使用最新加载的日志记录配置。创建一个新的以加载新的日志记录配置。 ConfigMap ConfigMap

注意	不要从ConfigMap中删除monitorInterval选项。

定期对帐

尽管集群 Operator 会对从Kubernetes集群收到的有关所需集群资源的所有通知做出反应，但如果该 Operator 未运行或由于某种原因未收到通知，则所需资源将与服务器状态不同步。运行Kubernetes集群

为了正确处理故障转移，群集 Operator 执行定期对帐过程，以便它可以将所需资源的状态与当前群集部署进行比较，以便在所有资源之间具有一致的状态。您可以使用变量设置定期对帐的时间间隔。 STRIMZI_FULL_RECONCILIATION_INTERVAL_MS

5.1.2。供应基于角色的访问控制（RBAC）

对于群集操作，以功能它需要的Kubernetes集群内的权限与资源，如互动，等，还有管理的资源，如，，，和。根据Kubernetes基于角色的访问控制（RBAC）资源来描述这种权限： Kafka KafkaConnect ConfigMaps Pods Deployments StatefulSets Services

- ServiceAccount，
- Role 并且， ClusterRole
- RoleBinding 和。 ClusterRoleBinding

除了使用a 单独运行外，集群 Operator 还为需要访问Kubernetes资源的组件管理一些RBAC资源。 ServiceAccount ClusterRoleBinding

Kubernetes还包括特权提升保护，可防止在一个组件下运行的组件授予该授予所没有的其他特权。因为集群 Operator 必须能够创建，并且需要其管理的资源，所以集群 Operator 还必须具有相同的特权。 ServiceAccount ServiceAccounts ServiceAccount ClusterRoleBindings RoleBindings

委托特权

当集群 Operator 部署的资源所需的资源也产生，和，如下所示： Kafka ServiceAccounts RoleBindings ClusterRoleBindings

- Kafka经纪人荚使用名为 ServiceAccount *cluster-name*-kafka
  - 使用机架功能时，通过调用将授予集群中节点的访问权限。 strimzi-*cluster-name*-kafka-init ClusterRoleBinding ServiceAccount ClusterRole strimzi-kafka-broker
  - 不使用机架功能时，不会创建绑定
- 动物园管理员荚使用名为 ServiceAccount *cluster-name*-zookeeper
- 实体经营吊舱采用了所谓的 ServiceAccount *cluster-name*-entity-operator
  - Topic operator 会产生带有状态信息的Kubernetes事件，因此绑定到调用一个，该调用通过授予访问权限 ServiceAccount ClusterRole strimzi-entity-operator strimzi-entity-operator RoleBinding
- 对于吊舱和资源使用一个名为 KafkaConnect KafkaConnectS2I ServiceAccount *cluster-name*-cluster-connect
- 该吊舱使用所谓的 KafkaMirrorMaker ServiceAccount *cluster-name*-mirror-maker
- 该吊舱使用所谓的 KafkaMirrorMaker2 ServiceAccount *cluster-name*-mirrormaker2
- 该吊舱使用所谓的 KafkaBridge ServiceAccount *cluster-name*-bridge

ServiceAccount

群集 Operator 最好使用以下示例运行：群集 Operator 示例 ServiceAccount
ServiceAccount

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: strimzi-cluster-operator
  labels:
    app: strimzi
```

然后， Operator 的需在其中指定：集群 Operator 的部分示例 Deployment spec.template.spec.serviceAccountName
Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-cluster-operator
  labels:
    app: strimzi
spec:
  replicas: 1
  selector:
    matchLabels:
      name: strimzi-cluster-operator
      strimzi.io/kind: cluster-operator
  template:
      # ...
```

请注意第12行，其中将指定为。 strimzi-cluster-operator ServiceAccount serviceAccountName

ClusterRoles

群集 Operator 需要使用来访问必需的资源。根据Kubernetes集群的设置，可能需要集群管理员来创建。 ClusterRoles ClusterRoles

注意	只有创建时才需要群集管理员权限。群集 Operator 将不会在群集管理员帐户下运行。 ClusterRoles

在遵循*最小权限原则*，并且只包含由集群 Operator 操作Kafka，Kafka连接，并动物园管理员集群需要这些特权。第一组分配的权限允许集群 Operator 管理Kubernetes资源，例如，，，和。 ClusterRoles StatefulSets Deployments Pods ConfigMaps

群集 Operator 使用ClusterRoles在名称空间范围的资源级别和群集范围的资源级别上授予权限：
ClusterRole 对于群集 Operator ，使用命名空间资源

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: strimzi-cluster-operator-namespaced
  labels:
    app: strimzi
rules:
  - apiGroups:
      - ""
    resources:
      # The cluster operator needs to access and manage service accounts to grant Strimzi components cluster
permissions
      - serviceaccounts
    verbs:
      - get
      - create
      - delete
      - patch
      - update
  - apiGroups:
      - "rbac.authorization.k8s.io"
    resources:
      # The cluster operator needs to access and manage rolebindings to grant Strimzi components cluster
permissions
      - rolebindings
    verbs:
      - get
      - create
      - delete
      - patch
      - update
  - apiGroups:
      - ""
    resources:
      # The cluster operator needs to access and manage config maps for Strimzi components configuration
      - configmaps
      # The cluster operator needs to access and manage services to expose Strimzi components to network traffic
      - services
      # The cluster operator needs to access and manage secrets to handle credentials
      - secrets
      # The cluster operator needs to access and manage persistent volume claims to bind them to Strimzi
components for persistent data
      - persistentvolumeclaims
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
  - apiGroups:
      - "kafka.strimzi.io"
    resources:
      # The cluster operator runs the KafkaAssemblyOperator, which needs to access and manage Kafka resources
      - kafkas
      - kafkas/status
      # The cluster operator runs the KafkaConnectAssemblyOperator, which needs to access and manage KafkaConnect
resources
      - kafkaconnects
      - kafkaconnects/status
      # The cluster operator runs the KafkaConnectS2IAssemblyOperator, which needs to access and manage
KafkaConnectS2I resources
      - kafkaconnects2is
      - kafkaconnects2is/status
      # The cluster operator runs the KafkaConnectorAssemblyOperator, which needs to access and manage
KafkaConnector resources
      - kafkaconnectors
      - kafkaconnectors/status
      # The cluster operator runs the KafkaMirrorMakerAssemblyOperator, which needs to access and manage
```

```
KafkaMirrorMaker resources
      - kafkamirrormakers
      - kafkamirrormakers/status
      # The cluster operator runs the KafkaBridgeAssemblyOperator, which needs to access and manage BridgeMaker
resources
      - kafkabridges
      - kafkabridges/status
      # The cluster operator runs the KafkaMirrorMaker2AssemblyOperator, which needs to access and manage
KafkaMirrorMaker2 resources
      - kafkamirrormaker2s
      - kafkamirrormaker2s/status
      # The cluster operator runs the KafkaRebalanceAssemblyOperator, which needs to access and manage
KafkaRebalance resources
      - kafkarebalances
      - kafkarebalances/status
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
  - apiGroups:
      - ""
    resources:
      # The cluster operator needs to access and delete pods, this is to allow it to monitor pod health and
coordinate rolling updates
      - pods
    verbs:
      - get
      - list
      - watch
      - delete
  - apiGroups:
      - ""
    resources:
      - endpoints
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      # The cluster operator needs the extensions api as the operator supports Kubernetes version 1.11+
      # apps/v1 was introduced in Kubernetes 1.14
      - "extensions"
    resources:
      # The cluster operator needs to access and manage deployments to run deployment based Strimzi components
      - deployments
      - deployments/scale
      # The cluster operator needs to access replica sets to manage Strimzi components and to determine error
states
      - replicasets
      # The cluster operator needs to access and manage replication controllers to manage replicasets
      - replicationcontrollers
      # The cluster operator needs to access and manage network policies to lock down communication between
Strimzi components
      - networkpolicies
      # The cluster operator needs to access and manage ingresses which allow external access to the services in
a cluster
      - ingresses
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
  - apiGroups:
      - "apps"
    resources:
      # The cluster operator needs to access and manage deployments to run deployment based Strimzi components
      - deployments
      - deployments/scale
      - deployments/status
      # The cluster operator needs to access and manage stateful sets to run stateful sets based Strimzi
components
      - statefulsets
      # The cluster operator needs to access replica-sets to manage Strimzi components and to determine error
states
      - replicasets
    verbs:
      - get
```

```
      - list
      - watch
      - create
      - delete
      - patch
      - update
  - apiGroups:
      - ""
    resources:
      # The cluster operator needs to be able to create events and delegate permissions to do so
      - events
    verbs:
      - create
  - apiGroups:
      # OpenShift S2I requirements
      - apps.openshift.io
    resources:
      - deploymentconfigs
      - deploymentconfigs/scale
      - deploymentconfigs/status
      - deploymentconfigs/finalizers
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
  - apiGroups:
      # OpenShift S2I requirements
      - build.openshift.io
    resources:
      - buildconfigs
      - builds
    verbs:
      - create
      - delete
      - get
      - list
      - patch
      - watch
      - update
  - apiGroups:
      # OpenShift S2I requirements
      - image.openshift.io
    resources:
      - imagestreams
      - imagestreams/status
    verbs:
      - create
      - delete
      - get
      - list
      - watch
      - patch
      - update
  - apiGroups:
      - networking.k8s.io
    resources:
      # The cluster operator needs to access and manage network policies to lock down communication between
Strimzi components
      - networkpolicies
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
  - apiGroups:
      - route.openshift.io
    resources:
      # The cluster operator needs to access and manage routes to expose Strimzi components for external access
      - routes
      - routes/custom-host
    verbs:
      - get
      - list
      - create
      - delete
      - patch
      - update
```

```
  - apiGroups:
      - policy
    resources:
      # The cluster operator needs to access and manage pod disruption budgets this limits the number of
concurrent disruptions
      # that a Strimzi component experiences, allowing for higher availability
      - poddisruptionbudgets
    verbs:
      - get
      - list
      - watch
      - create
      - delete
      - patch
      - update
```

第二个包括群集范围的资源所需的权限。
`ClusterRole` 为集群 Operator 提供集群范围的资源

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: strimzi-cluster-operator-global
  labels:
    app: strimzi
rules:
  - apiGroups:
      - "rbac.authorization.k8s.io"
    resources:
      # The cluster operator needs to create and manage cluster role bindings in the case of an install where a
user
      # has specified they want their cluster role bindings generated
      - clusterrolebindings
    verbs:
      - get
      - create
      - delete
      - patch
      - update
      - watch
  - apiGroups:
      - storage.k8s.io
    resources:
      # The cluster operator requires "get" permissions to view storage class details
      # This is because only a persistent volume of a supported storage class type can be resized
      - storageclasses
    verbs:
      - get
  - apiGroups:
      - ""
    resources:
      # The cluster operator requires "list" permissions to view all nodes in a cluster
      # The listing is used to determine the node addresses when NodePort access is configured
      # These addresses are then exposed in the custom resource states
      - nodes
    verbs:
      - list
```

在表示由一个用于齿条特征在Kafka豆荚初始化容器所需的访问。如委派特权中所述，群集 Operator 也需要此角色才能委派此访问权限。为集群 Operator 提供了将其Kubernetes节点的访问权限委派给Kafka代理Pod的功能 `strimzi-kafka-broker` `ClusterRole`
`ClusterRole`

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: strimzi-kafka-broker
  labels:
    app: strimzi
rules:
  - apiGroups:
      - ""
    resources:
      # The Kafka Brokers require "get" permissions to view the node they are on
      # This information is used to generate a Rack ID that is used for High Availability configurations
      - nodes
    verbs:
      - get
```

该代表由 Topic Operator 所需要的访问权限。如委派特权中所述，群集 Operator 也需要此角色才能委派此访问权限。允许集群 Operator 将对事件的访问权委派给 Topic Operator `strimzi-topic-operator` `ClusterRole`
`ClusterRole`

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
```

```
metadata:
  name: strimzi-entity-operator
  labels:
    app: strimzi
rules:
  - apiGroups:
      - "kafka.strimzi.io"
    resources:
      # The entity operator runs the KafkaTopic assembly operator, which needs to access and manage KafkaTopic
resources
      - kafkatopics
      - kafkatopics/status
      # The entity operator runs the KafkaUser assembly operator, which needs to access and manage KafkaUser
resources
      - kafkausers
      - kafkausers/status
    verbs:
      - get
      - list
      - watch
      - create
      - patch
      - update
      - delete
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      # The entity operator needs to be able to create events
      - create
  - apiGroups:
      - ""
    resources:
      # The entity operator user-operator needs to access and manage secrets to store generated credentials
      - secrets
    verbs:
      - get
      - list
      - create
      - patch
      - update
      - delete
```

ClusterRoleBindings

Operator 需要并将其与其：关联，以包含集群范围的资源。集群 Operator 示例 ClusterRoleBindings RoleBindings ClusterRole S
erviceAccount ClusterRoleBindings ClusterRoles
ClusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: strimzi-cluster-operator
  labels:
    app: strimzi
subjects:
  - kind: ServiceAccount
    name: strimzi-cluster-operator
    namespace: myproject
roleRef:
  kind: ClusterRole
  name: strimzi-cluster-operator-global
  apiGroup: rbac.authorization.k8s.io
```

ClusterRoleBindings 委派所需的代理也需要：集群 Operator 示例 ClusterRoles
RoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: strimzi-cluster-operator-kafka-broker-delegation
  labels:
    app: strimzi
# The Kafka broker cluster role must be bound to the cluster operator service account so that it can delegate the
cluster role to the Kafka brokers.
# This must be done to avoid escalating privileges which would be blocked by Kubernetes.
subjects:
  - kind: ServiceAccount
    name: strimzi-cluster-operator
    namespace: myproject
roleRef:
  kind: ClusterRole
```

```
  name: strimzi-kafka-broker
  apiGroup: rbac.authorization.k8s.io
```

ClusterRoles 仅包含命名空间资源的绑定仅使用。 RoleBindings

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: strimzi-cluster-operator
  labels:
    app: strimzi
subjects:
  - kind: ServiceAccount
    name: strimzi-cluster-operator
    namespace: myproject
roleRef:
  kind: ClusterRole
  name: strimzi-cluster-operator-namespaced
  apiGroup: rbac.authorization.k8s.io

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: strimzi-cluster-operator-entity-operator-delegation
  labels:
    app: strimzi
# The Entity Operator cluster role must be bound to the cluster operator service account so that it can delegate
the cluster role to the Entity Operator.
# This must be done to avoid escalating privileges which would be blocked by Kubernetes.
subjects:
  - kind: ServiceAccount
    name: strimzi-cluster-operator
    namespace: myproject
roleRef:
  kind: ClusterRole
  name: strimzi-entity-operator
  apiGroup: rbac.authorization.k8s.io
```

5.2。使用 Topic operator

使用资源创建，修改或删除 Topic 时， Topic operator 可确保这些更改反映在Kafka集群中。 KafkaTopic

在*部署Strimzi*指南提供的说明部署 Topic 操作：

- 使用群集 operator （推荐）
- 独立运行于非Strimzi管理的Kafka集群

5.2.1。Kafka Topic 资源

该资源用于配置 Topic ，包括分区和副本的数量。 KafkaTopic

模式参考中描述了的完整模式。 KafkaTopic KafkaTopic

识别用于 Topic 处理的Kafka集群

甲资源包括标签定义Kafka簇（来自名称衍生的合适的名称它属于哪个资源）。 KafkaTopic Kafka

例如：

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaTopic
metadata:
  name: topic-name-1
  labels:
    strimzi.io/cluster: my-cluster
```

Topic Operator 使用该标签来标识资源并创建新 Topic ，以及随后对该 Topic 的处理。 KafkaTopic

如果标签与Kafka群集不匹配，则 Topic operator 将无法识别，并且不会创建 Topic 。 KafkaTopic

处理 Topic 更改

Topic operator 必须解决的一个基本问题是没有单一的事实来源：资源和Kafka Topic 都可以独立于 operator 进行修改。复杂的是， Topic operator 可能并不总是能够实时观察每一端的变化（例如，该 operator 可能已关闭）。 KafkaTopic

为了解决这个问题， Operator 维护了自己有关每个 Topic 的信息的私有副本。无论是在Kafka集群中还是在Kubernetes中发生更改时，它都会查看另一个系统的状态及其私有副本，以确定需要进行哪些更改以使所有内容保持同步。每当 Operator 启动时，运行时就会发生相同的事情。

例如，假设 Topic operator 未运行，并且创建了一个。当 Operator 启动时，它将缺少“ my-topic”的私有副本，因此可以推断出自上次运行以来已创建。 Operator 将创建与相对应的 Topic ，并存储的元数据的私有副本。 KafkaTopic my-topic KafkaTopic my-topic my-topic

私有副本允许 Operator 应对在Kafka和Kubernetes中都更改 Topic 配置的情况，只要更改不兼容（例如，都将同一 Topic 配置键更改为不同的值）即可。如果发生不兼容的更改，Kafka配置将获胜，并且将更新来反映这一点。 KafkaTopic

私有副本与Kafka本身使用的ZooKeeper集成在一起。这减轻了可用性问题，因为如果ZooKeeper未运行，则Kafka本身无法运行，因此，即使该 Operator 是无状态的，该 Operator 的可用性也不会低于它。

Kafka Topic 用法建议

处理 Topic 时，请保持一致。始终直接在Kubernetes中处理资源或 Topic 。避免针对给定 Topic 常规在两种方法之间进行切换。 KafkaTopic

使用反映 Topic 性质的 Topic 名称，并记住以后不能更改名称。

如果在Kafka中创建 Topic ，请使用有效的Kubernetes资源名称作为名称，否则Topic Operator将需要使用与Kubernetes规则一致的名称来创建相应的 Topic 。 KafkaTopic

注意	Kubernetes社区中的标识符和名称中概述了有关Kubernetes中标识符和名称的建议。

Kafka Topic 命名约定

Kafka和Kubernetes 分别为在Kafka中命名 Topic 强加了自己的验证规则。每个都有有效的名称，而另一个则无效。 KafkaTopic.metadata.name

使用该属性，可以在Kafka中创建一个有效的 Topic ，其名称对于Kubernetes中的Kafka Topic 无效。 spec.topicName

该属性继承了Kafka命名验证规则： spec.topicName

- 名称不得超过249个字符。
- Kafka Topic 的有效字符为ASCII字母数字，，，和。 . _ -
- 名称不能为或，尽管可以在名称中使用，例如或。 . .. . exampleTopic. .exampleTopic

spec.topicName 不得更改。

例如：

```
apiVersion: {KafkaApiVersion}
kind: KafkaTopic
metadata:
  name: topic-name-1
spec:
  topicName: topicName-1 (1)
  # ...
```

1. 大写在Kubernetes中无效。

不能更改为：

```
apiVersion: {KafkaApiVersion}
kind: KafkaTopic
metadata:
  name: topic-name-1
spec:
  topicName: name-2
  # ...
```

注意	某些Kafka客户端应用程序（例如Kafka Streams）可以以编程方式在Kafka中创建 Topic 。如果这些 Topic 的名称是无效的Kubernetes资源名称，则 Topic operator 将根据Kafka名称为其提供有效名称。替换无效字符，并在名称后附加一个哈希。

5.2.2。配置Kafka Topic

使用资源的属性来配置Kafka Topic 。 KafkaTopic

您可以用来创建或修改 Topic ，以及删除现有 Topic 。 kubectl apply kubectl delete

例如：

- kubectl apply -f <topic-config-file>
- kubectl delete KafkaTopic <topic-name>

此过程说明如何创建具有10个分区和2个副本的 Topic 。
在你开始前

在进行更改之前，请考虑以下几点，这一点很重要：

- Kafka并*没有*支持使得通过以下更改资源： KafkaTopic
  - 使用更改 Topic 名称 spec.topicName
  - 减少分区大小，使用 spec.partitions
- 您不能用于更改最初指定的副本数。 spec.replicas
- 带关键字的 Topic 的增加将改变记录的分区方式，当 Topic 使用*语义分区*时，这尤其成问题。 spec.partitions

先决条件

- 正在运行的Kafka群集，该群集配置了使用TLS身份验证和加密的Kafka代理侦听器。
- 正在运行的 Topic operator （通常与Entity Operator一起部署）。
- 对于删除的话题，在（默认）的的资源。 `delete.topic.enable=true` `spec.kafka.config` `Kafka`

程序

1. 准备一个包含要创建的文件。一个例子 `KafkaTopic`
   `KafkaTopic`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: KafkaTopic
   metadata:
     name: orders
     labels:
       strimzi.io/cluster: my-cluster
   spec:
     partitions: 10
     replicas: 2
   ```

小费	修改 Topic 时，您可以使用来获取资源的当前版本。 `kubectl get kafkatopic orders -o yaml`

2. 在Kubernetes中创建资源。 `KafkaTopic`

   ```
   kubectl apply -f TOPIC-CONFIG-FILE
   ```

5.2.3。使用资源请求和限制配置 Topic operator

您可以将资源（例如CPU和内存）分配给 Topic operator ，并限制其可以消耗的资源量。
先决条件

- 群集 Operator 正在运行。

程序

1. 根据需要在编辑器中更新Kafka集群配置：

   ```
   kubectl edit kafka MY-CLUSTER
   ```

2. 在资源的属性中，为 Topic operator 设置资源请求和限制。 `spec.entityOperator.topicOperator.resources` `Kafka`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   spec:
     # Kafka and ZooKeeper sections...
     entityOperator:
       topicOperator:
         resources:
           request:
             cpu: "1"
             memory: 500Mi
           limit:
             cpu: "1"
             memory: 500Mi
   ```

3. 应用新配置以创建或更新资源。

   ```
   kubectl apply -f KAFKA-CONFIG-FILE
   ```

5.3。使用用户 Operator

使用资源创建，修改或删除用户时，用户 Operator 确保将这些更改反映在Kafka群集中。 `KafkaUser`

在*部署Strimzi*指南提供说明部署用户操作：

- 使用群集 operator （推荐）
- 独立运行于非Strimzi管理的Kafka集群

有关模式的更多信息，请参见模式参考。验证和授权对Kafka的访问 `KafkaUser`

使用启用身份验证和授权机制，一个特定的客户端用来访问Kafka。 `KafkaUser`

有关用于管理用户和保护对Kafka代理的访问的更多信息，请参阅确保对Kafka代理的访问。 `KafkUser`

5.3.1。使用资源请求和限制配置用户 Operator

您可以将资源（例如CPU和内存）分配给用户 Operator ，并可以限制其消耗的资源量。
先决条件

- 群集 Operator 正在运行。

程序

1. 根据需要在编辑器中更新Kafka集群配置：

   ```
   kubectl edit kafka MY-CLUSTER
   ```

2. 在资源的属性中，为用户 Operator 设置资源请求和限制。 `spec.entityOperator.userOperator.resources` `Kafka`

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   spec:
     # Kafka and ZooKeeper sections...
     entityOperator:
       userOperator:
         resources:
           request:
             cpu: "1"
             memory: 500Mi
           limit:
             cpu: "1"
             memory: 500Mi
   ```

   保存文件并退出编辑器。群集 Operator 将自动应用更改。

5.4。使用Prometheus指标监控 Operator

Strimzi operator 公开Prometheus指标。指标将自动启用，并包含有关以下信息：

- 对帐数量
- Operator 正在处理的自定义资源数量
- 对帐期限
- 来自 Operator 的JVM指标

此外，我们提供了示例Grafana仪表板。

有关Prometheus的更多信息，请参阅*Deploying Strimzi*指南中的Kafka度量标准。

6. Kafka桥

本章概述了Strimzi Kafka Bridge，并帮助您开始使用其REST API与Strimzi进行交互。要在您的本地环境中试用Kafka Bridge，请参阅本章后面的Kafka Bridge快速入门。

6.1。Kafka桥概述

您可以将Kafka Bridge用作接口，以向Kafka集群发出特定类型的请求。

6.1.1。Kafka Bridge界面

Strimzi Kafka Bridge提供了RESTful接口，该接口允许基于HTTP的客户端与Kafka群集进行交互。Kafka Bridge提供了与Strimzi的Web API连接的优点，而无需客户端应用程序解释Kafka协议。

该API有两个主要资源- 和-可通过端点公开和访问，以与Kafka集群中的消费者和生产者进行交互。这些资源仅与Kafka桥有关，而与直接连接到Kafka的消费者和生产者无关。 `consumers` `topics`

HTTP请求

Kafka Bridge支持对Kafka集群的HTTP请求，其方法包括：

- 将消息发送到 Topic 。
- 从 Topic 检索消息。
- 创建和删除使用者。
- 为使用者订阅 Topic ，以便他们开始从这些 Topic 接收消息。
- 检索使用者已订阅的 Topic 列表。
- 退订 Topic 的消费者。
- 将分区分配给使用者。
- 提交消费者抵消清单。
- 在分区上进行搜索，以便使用者开始从第一个或最后一个偏移位置或给定的偏移位置接收消息。

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

客户端可以生成和使用消息，而无需使用本机Kafka协议。
额外资源

- 要查看API文档（包括示例请求和响应），请参阅Strimzi网站上的Kafka Bridge API参考。

6.1.2。Kafka桥的受支持客户

您可以使用Kafka Bridge将*内部*和*外部* HTTP客户端应用程序与您的Kafka群集集成在一起。

内部客户端内部客户端是在与Kafka Bridge本身相同的Kubernetes集群中运行的基于容器的HTTP客户端。内部客户端可以在KafkaBridge自定义资源中定义的主机和端口上访问Kafka Bridge。外部客户端外部客户端是在Kubernetes群集之外运行并部署Kafka Bridge的HTTP客户端。外部客户端可以通过OpenShift路由，负载平衡器服务或使用Ingress访问Kafka桥。

HTTP内部和外部客户端集成

6.1.3。保护Kafka桥

Strimzi当前不为Kafka Bridge提供任何加密，身份验证或授权。这意味着从外部客户端发送到Kafka Bridge的请求为：

- 未加密，必须使用HTTP而不是HTTPS
- 未经身份验证发送

但是，您可以使用其他方法来保护Kafka Bridge，例如：

- Kubernetes网络策略，定义了哪些Pod可以访问Kafka桥。
- 具有身份验证或授权的反向代理，例如OAuth2代理。
- API网关。
- TLS终止的入口或OpenShift路由。

连接到Kafka代理时，Kafka桥支持TLS加密以及TLS和SASL身份验证。在您的Kubernetes集群中，您可以配置：

- Kafka Bridge和您的Kafka集群之间基于TLS或SASL的身份验证
- Kafka网桥和您的Kafka群集之间的TLS加密连接。

有关更多信息，请参阅Kafka Bridge中的身份验证支持。

您可以在Kafka代理中使用ACL来限制可以使用Kafka Bridge消费和产生的 Topic 。

6.1.4。在Kubernetes之外访问Kafka桥

部署后，只有在同一Kubernetes集群中运行的应用程序才能访问Strimzi Kafka Bridge。这些应用程序使用服务访问API。 *kafka-bridge-name-bridge-service*

如果要使Kafka Bridge对Kubernetes集群外部运行的应用程序可访问，则可以使用以下功能之一手动将其公开：

- LoadBalancer或NodePort类型的服务
- 入口资源
- OpenShift路线

如果您决定创建服务，请在中使用以下标签来配置服务将流量路由到的Pod： selector

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

1. Kubernetes集群中Kafka Bridge定制资源的名称。

6.1.5。向Kafka桥的要求

指定数据格式和HTTP标头，以确保将有效请求提交给Kafka Bridge。

内容类型标题

API请求和响应主体始终编码为JSON。

- 执行使用者操作时，如果存在非空主体，则请求必须提供以下标头： POST Content-Type

  ```
  Content-Type: application/vnd.kafka.v2+json
  ```

- 当执行生产者操作，请求必须提供标头指定所需的*嵌入的数据格式*，无论是或，如图所示，在下表中。 POST Content-Type json binary

嵌入式数据格式	Content-Type标头
JSON格式	`Content-Type: application/vnd.kafka.json.v2+json`
二元	`Content-Type: application/vnd.kafka.binary.v2+json`

使用端点创建使用者时，您可以设置嵌入式数据格式- 有关更多信息，请参阅下一节。 consumers/*groupid*

该如果不能设置要求有一个空的机构。空主体可用于创建具有默认值的使用者。 Content-Type POST

嵌入式数据格式

嵌入式数据格式是使用Kafka桥通过HTTP从生产者传输到消费者的Kafka消息的格式。支持两种嵌入式数据格式：JSON和二进制。

使用端点创建使用者时，请求主体必须指定JSON或二进制的嵌入式数据格式。这是在字段中指定的，例如： `/consumers/groupid` `POST` `format`

```
{
  "name": "my-consumer",
  "format": "binary", (1)
...
}
```

1. 二进制嵌入式数据格式。

创建使用者时指定的嵌入式数据格式必须与它将使用的Kafka消息的数据格式匹配。

如果选择指定二进制嵌入式数据格式，则随后的生产者请求必须在请求正文中以Base64编码的字符串形式提供二进制数据。例如，使用端点发送消息时，必须在Base64中进行编码： `/topics/topicname` `records.value`

```
{
  "records": [
    {
      "key": "my-key",
      "value": "ZWR3YXJkdGhldGhyZWVsZWdnZWRjYXQ="
    },
  ]
}
```

生产者请求还必须提供与嵌入式数据格式相对应的标头，例如。 `Content-Type` `Content-Type: application/vnd.kafka.binary.v2+json`

接受标题

创建使用者之后，所有后续GET请求都必须提供以下格式的标头： `Accept`

`Accept: application/vnd.kafka.embedded-data-format.v2+json`

该要么是或。 `embedded-data-format` `json` `binary`

例如，在使用JSON的嵌入式数据格式检索订阅的使用者的记录时，请包含以下Accept标头：

`Accept: application/vnd.kafka.json.v2+json`

6.1.6。Kafka Bridge API资源

有关REST API端点和说明的完整列表，包括示例请求和响应，请参阅Strimzi网站上的Kafka Bridge API参考。

6.1.7。Kafka Bridge部署

您可以使用Cluster Operator将Kafka Bridge部署到Kubernetes集群中。

部署Kafka Bridge后，集群 Operator 会在您的Kubernetes集群中创建Kafka Bridge对象。对象包括*Deployment*，*service*和*pod*，每个对象均以Kafka Bridge的自定义资源中给出的名称命名。额外资源

- 有关部署说明，请参阅*Deploying Strimzi*指南中的将Kafka Bridge部署到Kubernetes集群。
- 有关配置Kafka Bridge的详细信息，请参阅Kafka Bridge配置。
- 有关为资源配置主机和端口的信息，请参阅Kafka Bridge HTTP配置。 `KafkaBridge`
- 有关集成外部客户端的信息，请参见访问Kubernetes外部的Kafka桥。

6.2。Kafka Bridge快速入门

使用此快速入门，可以在本地开发环境中试用Strimzi Kafka Bridge。你将学到如何：

- 将Kafka Bridge部署到您的Kubernetes集群
- 使用端口转发将Kafka Bridge服务公开到本地计算机
- 产生消息到您的Kafka集群中的 Topic 和分区
- 创建一个Kafka Bridge使用者
- 执行基本的使用者操作，例如使使用者订阅 Topic 并检索您产生的消息

在此快速入门中，HTTP请求被格式化为curl命令，您可以将它们复制并粘贴到终端。需要访问Kubernetes集群； 要运行和管理本地Kubernetes集群，请使用Minikube，CodeReady Containers或MiniShift之类的工具。

确保您具有先决条件，然后按照本章中提供的顺序执行任务。
关于数据格式

在本快速入门中，您将生成和使用JSON格式（而非二进制）的消息。有关示例请求中使用的数据格式和HTTP标头的更多信息，请参阅对Kafka Bridge的请求。快速入门的先决条件

- 集群管理员可以访问本地或远程Kubernetes集群。
- Strimzi已安装。
- 集群 Operator 在Kubernetes命名空间中部署的运行中的Kafka集群。

- 实体 Operator 作为Kafka集群的一部分进行部署和运行。

6.2.1。将Kafka Bridge部署到您的Kubernetes集群

Strimzi包含一个YAML示例，该示例指定了Strimzi Kafka Bridge的配置。对此文件进行一些最小的更改，然后将Kafka Bridge的实例部署到您的Kubernetes集群。
程序

1. 编辑文件。 `examples/bridge/kafka-bridge.yaml`

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: quickstart (1)
spec:
  replicas: 1
  bootstrapServers: <cluster-name>-kafka-bootstrap:9092 (2)
  http:
    port: 8080
```

   a. 部署Kafka Bridge时，会将其附加到部署名称和其他相关资源。在此示例中，命名了Kafka Bridge部署，并命名了随附的Kafka Bridge服务。 `-bridge` `quickstart-bridge` `quickstart-bridge-service`
   b. 在属性中，输入Kafka群集的名称作为。 `bootstrapServers` `<cluster-name>`
2. 将Kafka Bridge部署到您的Kubernetes集群：

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

   一个部署，服务，以及其他相关资源都在你的Kubernetes集群创建。 `quickstart-bridge`
3. 验证Kafka Bridge是否已成功部署：

```
kubectl get deployments

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
quickstart-bridge                       1/1     1            1           34m
my-cluster-connect                      1/1     1            1           24h
my-cluster-entity-operator              1/1     1            1           24h
#...
```

接下来做什么

将Kafka Bridge部署到您的Kubernetes集群后，将Kafka Bridge服务公开给您的本地计算机。额外资源

- 有关配置Kafka Bridge的更多详细信息，请参阅Kafka Bridge配置。

6.2.2。将Kafka Bridge服务公开到您的本地计算机

接下来，使用端口转发将Strimzi Kafka Bridge服务公开到http：// localhost：8080上的本地计算机。

注意	端口转发仅适用于开发和测试目的。

程序

1. 列出您的Kubernetes集群中的Pod名称：

```
kubectl get pods -o name

pod/kafka-consumer
# ...
pod/quickstart-bridge-589d78784d-9jcnr
pod/strimzi-cluster-operator-76bcf9bc76-8dnfm
```

2. 连接到端口上的Pod ： `quickstart-bridge` `8080`

```
kubectl port-forward pod/quickstart-bridge-589d78784d-9jcnr 8080:8080 &
```

注意	如果本地计算机上的端口8080已被使用，请使用其他HTTP端口，例如。 `8008`

API请求现在从本地计算机上的端口8080转发到Kafka Bridge pod中的端口8080。

6.2.3。产生到 Topic 和分区的消息

接下来，使用 Topic 终结点以JSON格式生成 Topic 消息。您可以在请求正文中为邮件指定目标分区，如下所示。所述分区端点提供了用于指定单个目的地分区的所有消息作为路径参数的替代方法。程序

1. 在文本编辑器中，为具有三个分区的Kafka Topic 创建YAML定义。

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaTopic
metadata:
```

```
  name: bridge-quickstart-topic
  labels:
    strimzi.io/cluster: <kafka-cluster-name> (1)
spec:
  partitions: 3 (2)
  replicas: 1
  config:
    retention.ms: 7200000
    segment.bytes: 1073741824
```

a. 部署Kafka桥的Kafka集群的名称。
b. Topic 的分区数。

2. 将文件另存为目录。 examples/topic bridge-quickstart-topic.yaml
3. 在您的Kubernetes集群中创建 Topic ：

```
kubectl apply -f examples/topic/bridge-quickstart-topic.yaml
```

4. 使用Kafka Bridge，针对您创建的 Topic 生成三则消息：

```
curl -X POST \
  http://localhost:8080/topics/bridge-quickstart-topic \
  -H 'content-type: application/vnd.kafka.json.v2+json' \
  -d '{
    "records": [
        {
            "key": "my-key",
            "value": "sales-lead-0001"
        },
        {
            "value": "sales-lead-0002",
            "partition": 2
        },
        {
            "value": "sales-lead-0003"
        }
    ]
}'
```

- sales-lead-0001 根据密钥的哈希值被发送到分区。
- sales-lead-0002 直接发送到分区2。
- sales-lead-0003 使用循环方法将其发送到 Topic 中的分区。 bridge-quickstart-topic

5. 如果请求成功，则Kafka Bridge将返回一个数组，以及代码和标头。对于每个消息，数组描述： offsets 200 content-type appl
ication/vnd.kafka.v2+json offsets

- 邮件发送到的分区
- 分区的当前消息偏移量
  示例响应

```
#...
{
  "offsets":[
    {
      "partition":0,
      "offset":0
    },
    {
      "partition":2,
      "offset":0
    },
    {
      "partition":0,
      "offset":1
    }
  ]
}
```

接下来做什么

生成 Topic 和分区的消息后，创建一个Kafka Bridge使用者。额外资源

- API参考文档中的POST / topics / {topicname}。
- API参考文档中的POST / topics / {topicname} / partitions / {partitionid}。

6.2.4。创建Kafka Bridge使用者

在Kafka集群中执行任何使用者操作之前，必须首先使用使用者端点创建使用者。该消费者称为*Kafka Bridge消费者*。程序

1. 在名为的新消费者组中创建Kafka Bridge消费者： bridge-quickstart-consumer-group

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group \
  -H 'content-type: application/vnd.kafka.v2+json' \
  -d '{
    "name": "bridge-quickstart-consumer",
```

```
  "auto.offset.reset": "earliest",
  "format": "json",
  "enable.auto.commit": false,
  "fetch.min.bytes": 512,
  "consumer.request.timeout.ms": 30000
}'
```

- 命名使用者，并将嵌入式数据格式设置为。 `bridge-quickstart-consumer` `json`
- 定义了一些基本配置设置。
- 由于设置为，使用者不会自动将偏移量提交到日志。您稍后将在此快速入门中手动提交偏移量。 `enable.auto.commit` `false`

如果请求成功，则Kafka Bridge将在响应正文中返回消费者ID（`instance_id`）和基本URL（`base_uri`）以及代码。回应范例 `200`

```
#...
{
  "instance_id": "bridge-quickstart-consumer",
  "base_uri":"http://<bridge-name>-bridge-service:8080/consumers/bridge-quickstart-consumer-group
/instances/bridge-quickstart-consumer"
}
```

2. `base_uri`在此快速入门中，复制基本URL（）以用于其他使用者操作。

接下来做什么

现在，您已经创建了Kafka Bridge使用者，您可以将其订阅 Topic 。额外资源

- API参考文档中的POST / consumers / {groupid}。

6.2.5。为Kafka Bridge使用者订阅 Topic

创建Kafka Bridge使用者后，使用订阅端点将其订阅一个或多个 Topic 。订阅后，消费者开始接收针对该 Topic 产生的所有消息。程序

- 将使用者订阅您先前创建的 Topic （在将消息生成到 Topic 和分区中）： `bridge-quickstart-topic`

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-
consumer/subscription \
  -H 'content-type: application/vnd.kafka.v2+json' \
  -d '{
    "topics": [
        "bridge-quickstart-topic"
    ]
}'
```

该数组可以包含一个 Topic （如下所示）或多个 Topic 。如果要使使用者订阅与正则表达式匹配的多个 Topic ，则可以使用字符串而不是数组。
`topics` `topic_pattern` `topics`

如果请求成功，则Kafka Bridge仅返回（无内容）代码。 `204`

接下来做什么

为Kafka Bridge使用者订阅 Topic 后，您可以从使用者中检索消息。额外资源

- API参考文档中的POST / consumers / {groupid} / instances / {name} / subscription。

6.2.6。检索来自Kafka Bridge使用者的最新消息

接下来，通过从记录端点请求数据，从Kafka Bridge使用者检索最新消息。在生产中，HTTP客户端可以重复（循环）调用此终结点。程序

1. 如向 Topic 和分区生成消息中所述，向Kafka Bridge使用者生成其他消息。
2. 向端点提交请求： `GET` `records`

```
curl -X GET http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-
consumer/records \
  -H 'accept: application/vnd.kafka.json.v2+json'
```

创建并订阅Kafka Bridge使用者后，第一个GET请求将返回空响应，因为轮询操作将启动重新平衡过程来分配分区。
3. 重复第二步，从Kafka Bridge使用者中检索消息。

Kafka Bridge在响应正文中返回一条消息数组（描述 Topic 名称，键，值，分区和偏移量）以及代码。默认情况下，将从最新的偏移量检索消息。
`200`

```
HTTP/1.1 200 OK
content-type: application/vnd.kafka.json.v2+json
#...
[
  {
    "topic":"bridge-quickstart-topic",
    "key":"my-key",
    "value":"sales-lead-0001",
    "partition":0,
    "offset":0
```

```
  },
  {
    "topic":"bridge-quickstart-topic",
    "key":null,
    "value":"sales-lead-0003",
    "partition":0,
    "offset":1
  },
#...
```

注意	如果返回空响应，请按照向 Topic 和分区产生消息中的描述，向使用者产生更多记录，然后尝试再次检索消息。

接下来做什么

从Kafka Bridge使用方检索消息后，尝试将偏移量提交到log。额外资源

- API参考文档中的GET / consumers / {groupid} / instances / {name} / records。

6.2.7。向日志提交偏移量

接下来，使用offsets端点将Kafka Bridge使用方收到的所有消息的偏移量手动提交到日志。这是必需的，因为您先前在创建Kafka Bridge使用者中创建的Kafka Bridge使用者配置为。程序 `enable.auto.commit` `false`

- 将偏移量提交到日志： `bridge-quickstart-consumer`

  ```
  curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/offsets
  ```

  因为没有提交请求正文，所以将为使用者已经收到的所有记录提交偏移量。或者，请求主体可以包含一个数组（OffsetCommitSeekList），该数组指定要为其提交偏移量的 Topic 和分区。

  如果请求成功，则Kafka Bridge 仅返回一个代码。 `204`

接下来做什么

将偏移量提交到日志后，尝试端点以寻求偏移量。额外资源

- API参考文档中的POST / consumers / {groupid} / instances / {name} / offsets。

6.2.8。寻求分区的偏移量

接下来，使用positions端点将Kafka Bridge使用者配置为从特定偏移量然后从最新偏移量获取分区消息。这在Apache Kafka中称为查找操作。程序

1. 寻找 Topic 的分区0的特定偏移量： `quickstart-bridge-topic`

   ```
   curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/positions \
     -H 'content-type: application/vnd.kafka.v2+json' \
     -d '{
       "offsets": [
           {
               "topic": "bridge-quickstart-topic",
               "partition": 0,
               "offset": 2
           }
       ]
   }'
   ```

   如果请求成功，则Kafka Bridge 仅返回一个代码。 `204`
2. 向端点提交请求： `GET` `records`

   ```
   curl -X GET http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/records \
     -H 'accept: application/vnd.kafka.json.v2+json'
   ```

   Kafka Bridge从您寻求的偏移量返回消息。
3. 通过查找相同分区的最后一个偏移量来恢复默认的消息检索行为。这次，使用头寸/终点端点。

   ```
   curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/positions/end \
     -H 'content-type: application/vnd.kafka.v2+json' \
     -d '{
       "partitions": [
           {
               "topic": "bridge-quickstart-topic",
               "partition": 0
           }
       ]
   }'
   ```

如果请求成功，则Kafka Bridge将返回另一个代码。 `204`

注意	您也可以使用positions / beginning端点查找一个或多个分区的第一个偏移量。

接下来做什么

在本快速入门中，您已使用Strimzi Kafka桥在Kafka群集上执行了一些常用操作。现在，您可以删除之前创建的Kafka Bridge使用者。额外资源

- API参考文档中的POST / consumers / {groupid} / instances / {name} /位置。
- API参考文档中的POST / consumers / {groupid} / instances / {name} / positions / begin。
- API参考文档中的POST / consumers / {groupid} / instances / {name} / positions / end。

6.2.9。删除Kafka Bridge使用者

最后，删除在本快速入门中使用的Kafa Bridge使用者。
程序

- 通过向实例端点发送请求来删除Kafka Bridge使用者。 `DELETE`

  ```
  curl -X DELETE http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer
  ```

  如果请求成功，则Kafka Bridge 仅返回一个代码。 `204`

额外资源

- 在API参考文档中删除/ consumers / {groupid} / instances / {name}。

7. 用于集群重新平衡的巡航控制

您可以将Cruise Control部署到您的Strimzi群集，并使用它来*重新平衡* Kafka群集。

Cruise Control是一个开源系统，用于自动化Kafka操作，例如监视群集工作负载，基于预定义的约束重新平衡群集以及检测和修复异常。它包括四个主要组件-负载监视器，分析器，异常检测器和执行器-以及用于客户端交互的REST API。Strimzi利用REST API支持以下Cruise Control功能：

- 从多个*优化目标*中生成*优化建议*。
- 根据优化建议重新平衡Kafka集群。

当前不支持其他Cruise Control功能，包括异常检测，通知，自己编写目标以及更改 Topic 复制因子。

在中提供了用于Cruise Control的示例YAML文件。 `examples/cruise-control/`

7.1。为什么要使用定速巡航？

巡航控制减少了运行高效且平衡的Kafka集群所需的时间和精力。

随着时间的流逝，典型的群集可能会变得负载不均。处理大量消息流量的分区可能在可用代理之间分布不均。要重新平衡群集，管理员必须监视代理上的负载，并手动将繁忙的分区重新分配给具有备用容量的代理。

Cruise Control使集群重新平衡过程自动化。它基于CPU，磁盘和网络负载为集群构建了资源利用的*工作负载模型*，并生成了优化建议（您可以批准或拒绝）以实现更均衡的分区分配。一组可配置的优化目标用于计算这些建议。

批准优化建议后，Cruise Control会将其应用于您的Kafka集群。群集重新平衡操作完成后，将更有效地使用代理容器，并且Kafka群集将更均匀地平衡。
额外资源

- 巡航控制Wiki

7.2。优化目标概述

为了重新平衡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。

优化目标是限制Kafka集群中工作负载的重新分配和资源利用。Strimzi支持Cruise Control项目中开发的大多数优化目标。支持的目标（按优先级的默认降序排列）如下：

1. 机架意识
2. 复制容量
3. *容量*：磁盘容量，网络入站容量，网络出站容量，CPU容量
4. 副本分发
5. 潜在的网络输出
6. *资源分配*：磁盘利用率分配，网络入站利用率分配，网络出站利用率分配，CPU利用率分配

注意	使用代理资源的容量限制来控制资源分配目标。

7. 领导者字节输入速率分布
8. Topic 副本分发
9. 领导者副本分发
10. 首选领导人选举

有关每个优化目标的更多信息，请参阅Cruise Control Wiki中的Goals。

注意	尚不支持经纪内磁盘目标，“编写自己的目标”和Kafka分配器目标。

Strimzi自定义资源中的目标配置

您可以在和自定义资源中配置优化目标。Cruise Control具有必须满足的硬性优化目标以及master，default和用户提供的优化目标的配置。资源分配（磁盘，网络入站，网络出站和CPU）的优化目标受代理资源容量的限制。 Kafka KafkaRebalance

以下各节将详细介绍每个目标配置。

硬目标和软目标

硬目标是优化建议中*必须*满足的目标。未配置为硬目标的目标称为*软目标*。你能想到的软目标，*尽最大努力*的目标：他们根本*没有*需要优化的建议得到满足，但包括在优化计算。违反一个或多个软目标但满足所有硬目标的优化建议是有效的。

巡航控制系统将计算满足所有硬性目标和尽可能多的软性目标（按优先顺序排列）的优化建议。*不能*满足所有艰巨目标的优化建议将被Cruise Control拒绝，并且不会发送给用户以供批准。

注意	例如，您可能有一个软目标，即在集群中均匀分布 Topic 的副本（ Topic 副本分发目标）。如果这样做使所有已配置的硬目标都可以实现，那么Cruise Control将忽略该目标。

在Cruise Control中，以下主要优化目标已预设为硬目标：

```
RackAwareGoal; ReplicaCapacityGoal; DiskCapacityGoal; NetworkInboundCapacityGoal; NetworkOutboundCapacityGoal;
CPUCapacityGoal
```

通过在中编辑属性，可以在Cruise Control部署配置中配置硬目标。 hard.goals Kafka.spec.cruiseControl.config

- 要从Cruise Control继承预设的硬目标，请不要在 hard.goals Kafka.spec.cruiseControl.config
- 要更改预设的硬目标，请使用其完全限定的域名在属性中指定所需的目标。 hard.goals

硬优化目标的示例配置 Kafka

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    topicOperator: {}
    userOperator: {}
  cruiseControl:
    brokerCapacity:
      inboundNetwork: 10000KB/s
      outboundNetwork: 10000KB/s
    config:
      hard.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundCapacityGoal
      # ...
```

增加配置的硬目标的数量将减少定速巡航系统生成有效优化建议的可能性。

如果在指定自定义资源，巡航控制并*没有*检查用户提供的优化目标（列表）中包含*的所有*配置的硬目标（）。因此，如果列表中列出了用户提供的一些（但不是全部）优化目标，则即使指定了巡航控制系统，Cruise Control仍会将其视为硬目标。 skipHardGoalCheck: true KafkaRebalance KafkaRebalance.spec.goals hard.goals hard.goals skipHardGoalCheck: true

掌握优化目标

在*优化大师的目标*是提供给所有用户。未在主优化目标中列出的目标不可用于定速巡航控制操作。

除非您更改Cruise Control 部署配置，否则Strimzi将以优先级降序从Cruise Control继承以下主要优化目标：

```
RackAwareGoal; ReplicaCapacityGoal; DiskCapacityGoal; NetworkInboundCapacityGoal; NetworkOutboundCapacityGoal;
CPUCapacityGoal; ReplicaDistributionGoal; PotentialNwOutGoal; DiskUsageDistributionGoal;
NetworkInboundUsageDistributionGoal; NetworkOutboundUsageDistributionGoal; CpuUsageDistributionGoal;
TopicReplicaDistributionGoal; LeaderReplicaDistributionGoal; LeaderBytesInDistributionGoal;
PreferredLeaderElectionGoal
```

这些目标中有六个预设为硬目标。

为了降低复杂性，建议您使用继承的主优化目标，除非您需要从资源中完全排除一个或多个目标。如果需要，可以在默认优化目标的配置中修改主优化目标的优先级顺序。 KafkaRebalance

您可以在Cruise Control部署配置中配置主优化目标： Kafka.spec.cruiseControl.config.goals

- 要接受继承的主优化目标，请不要在中指定属性。 goals Kafka.spec.cruiseControl.config
- 如果需要修改继承的主优化目标，请在配置选项中以降序排列指定目标列表。 goals

注意	如果更改继承的主优化目标，则必须确保硬目标（如果在中的属性中进行了配置）是您配置的主优化目标的子集。否则，生成优化建议时将发生错误。 hard.goals Kafka.spec.cruiseControl.config

默认优化目标

Cruise Control使用*默认的优化目标*来生成*缓存的优化建议*。有关缓存的优化建议的更多信息，请参见优化建议概述。

您可以通过在自定义资源中设置用户提供的优化目标来覆盖默认优化目标。 KafkaRebalance

除非您在Cruise Control 部署配置中指定，否则主优化目标将用作默认优化目标。在这种情况下，使用主优化目标生成缓存的优化建议。 default.goals

- 要将主优化目标用作默认目标，请不要在中指定属性。 default.goals Kafka.spec.cruiseControl.config
- 要修改默认优化目标，请在中编辑属性。您必须使用主优化目标的子集。 default.goals Kafka.spec.cruiseControl.config

默认优化目标的示例配置 Kafka

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    topicOperator: {}
    userOperator: {}
  cruiseControl:
    brokerCapacity:
      inboundNetwork: 10000KB/s
      outboundNetwork: 10000KB/s
    config:
      default.goals: >
         com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
         com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal,
         com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal
      # ...
```

如果未指定默认优化目标，则使用主优化目标生成缓存的投标。

用户提供的优化目标

用户提供的优化目标可以 缩小为特定优化建议配置的默认目标。您可以根据需要在自定义资源中设置它们： spec.goals KafkaRebalance

KafkaRebalance.spec.goals

用户提供的优化目标可以针对不同情况生成优化建议。例如，您可能想在整个Kafka群集上优化领导者副本的分布，而不考虑磁盘容量或磁盘利用率。因此，您将创建一个自定义资源，其中包含一个由用户提供的领导者副本分发目标。 KafkaRebalance

用户提供的优化目标必须：

- 包括所有已配置的硬目标，否则会发生错误
- 成为主要优化目标的子集

要在生成优化建议时忽略配置的硬目标，请将属性添加到自定义资源。请参阅生成优化建议。额外资源 skipHardGoalCheck: true KafkaRebalance

- 巡航控制配置
- 巡航控制Wiki中的配置。

7.3。优化建议概述

一个*优化的建议*是，将产生更平衡Kafka集群，与券商之间的分布比较均匀工作负载分区修改建议的摘要。每个优化建议均基于用于生成该优化建议的一组优化目标，但要遵守对代理资源配置的任何容量限制。

优化建议包含在定制资源的属性中。提供的信息是完整优化建议的摘要。使用摘要来决定是否： Status.Optimization Result KafkaRebalance

- 批准优化建议。这指示Cruise Control将建议应用到Kafka集群并开始集群重新平衡操作。
- 拒绝优化建议。您可以更改优化目标，然后生成另一个建议。

所有优化建议都是*空运*：您必须先生成优化建议，才能批准集群重新平衡。可以生成的优化建议数量没有限制。

缓存的优化建议

Cruise Control 根据配置的默认优化目标维护一个*缓存的优化建议*。根据工作负载模型生成的缓存优化建议每15分钟更新一次，以反映Kafka集群的当前状态。如果使用默认优化目标生成优化建议，则Cruise Control将返回最新的缓存建议。

要更改缓存的优化建议刷新间隔，请在Cruise Control部署配置中编辑设置。对于快速变化的群集，请考虑较短的间隔，尽管这会增加Cruise Control服务器上的负载。 `proposal.expiration.ms`

优化方案的内容

下表描述了优化建议中包含的属性：

表3.优化建议中包含的属性

JSON属性	描述
`numIntraBrokerReplicaMovements`	将在群集的代理的磁盘之间传输的分区副本的总数。 **重新平衡操作期间对性能的影响**：相对较高，但低于。 `numReplicaMovements`
`excludedBrokersForLeadership`	尚不支持。返回一个空列表。
`numReplicaMovements`	将在单独的代理之间移动的分区副本的数量。 **重新平衡操作期间对性能的影响**：相对较高。
`onDemandBalancednessScoreBefore`, `onDemandBalancednessScoreAfter`	在生成优化建议之前和之后，对Kafka群集的总体*平衡性*进行的度量。 得分是通过从100中减去每个违反的软目标的总和而得出的.Cruise Control 根据几个因素（包括优先级-目标在目标或用户提供的目标中的位置）将a分配给每个优化目标。 `BalancednessScore` `BalancednessScore` `default.goals` 所述得分是基于Kafka群集的当前配置。该分数是基于所产生的优化建议。 `Before` `After`
`intraBrokerDataToMoveMB`	将在同一代理上的磁盘之间移动的每个分区副本的大小总和（另请参见）。 `numIntraBrokerReplicaMovements` **重新平衡操作期间对性能的影响**：可变。数字越大，集群重新平衡所需的时间就越长。在同一代理上的磁盘之间移动大量数据的影响要小于单独的代理之间的影响（请参阅参考资料）。 `dataToMoveMB`
`recentWindows`	优化提议所基于的度量标准窗口的数量。
`dataToMoveMB`	将移至单独的代理的每个分区副本的大小总和（另请参见）。 `numReplicaMovements` **重新平衡操作期间对性能的影响**：可变。数字越大，集群重新平衡所需的时间就越长。
`monitoredPartitionsPercentage`	优化建议涵盖的Kafka群集中的分区百分比。受数量影响。 `excludedTopics`
`excludedTopics`	如果您在资源的属性中指定了正则表达式，则在此列出与该表达式匹配的所有 Topic 名称。这些 Topic 从优化建议中的分区副本/领导者移动的计算中排除。 `spec.excludedTopicsRegex` `KafkaRebalance`
`numLeaderMovements`	其领导者将切换到不同副本的分区数。这涉及对ZooKeeper配置的更改。 **重新平衡操作期间对性能的影响**：相对较低。
`excludedBrokersForReplicaMove`	尚不支持。返回一个空列表。

额外资源

- 优化目标概述
- 生成优化建议
- 批准优化建议

7.4。重新平衡性能调整概述

您可以调整几个性能调整选项以实现群集重新平衡。这些选项控制重新平衡中分区副本和领导者移动的执行方式，以及分配给重新平衡操作的带宽。

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。当你批准一项提案，巡航控制服务器将这些命令的Kafka集群。

分区重新分配命令由以下两种操作之一组成：

- 分区移动：涉及将分区副本及其数据传输到新位置。分区移动可以采用以下两种形式之一：

    - 代理间移动：将分区副本移动到其他代理上的日志目录中。
    - 代理内移动：将分区副本移动到同一代理上的其他日志目录中。
- 领导者运动：这涉及切换分区副本的领导者。

Cruise Control批量向Kafka集群发出分区重新分配命令。重新平衡期间群集的性能受每个批次中包含的每种移动类型的数量影响。

副本移动策略

群集重新平衡性能还受到应用于批量分区重新分配命令的*副本移动策略*的影响。默认情况下，Cruise Control使用，它仅按命令生成的顺序应用命令。但是，如果提案中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。 `BaseReplicaMovementStrategy`

巡航控制提供了三种替代的副本移动策略，可以将它们应用于优化建议：

- `PrioritizeSmallReplicaMovementStrategy`：按大小递增的顺序重新分配订单。
- `PrioritizeLargeReplicaMovementStrategy`：按大小递减的顺序重新分配订单。
- `PostponeUrpReplicaMovementStrategy`：优先为没有同步副本的分区副本分配重新分配。

这些策略可以配置为序列。第一种策略尝试使用其内部逻辑比较两个分区重新分配。如果重新分配是等效的，则它将它们传递到序列中的下一个策略以决定顺序，依此类推。

重新平衡调整选项

巡航控制系统提供了几种配置选项，用于调整上述的重新平衡参数。您可以在Cruise Control服务器或优化建议级别上设置以下调整选项：

- 可以在的Kafka自定义资源中设置Cruise Control服务器设置。 `Kafka.spec.cruiseControl.config`
- 可以在下设置各个重新平衡性能配置。 `KafkaRebalance.spec`

相关配置总结如下：

服务器和配置 `KafkaRebalance`	**描述**	**默认值**
`num.concurrent.partition.movements.per.broker`	每个分区重新分配批处理中经纪人分区之间的最大移动次数	5
`concurrentPartitionMovementsPerBroker`		
`num.concurrent.intra.broker.partition.movements`	每个分区重新分配批次中经纪内分区移动的最大数量	2
`concurrentIntraBrokerPartitionMovements`		
`num.concurrent.leader.movements`	每个分区重新分配批处理中分区领导最大更改数	1000
`concurrentLeaderMovements`		
`default.replication.throttle`	要分配给分区重新分配的带宽（每秒字节）	没有限制
`replicationThrottle`		
`default.replica.movement.strategies`	策略列表（优先级顺序），用于确定对生成的投标执行分区重新分配命令的顺序。对于服务器，请使用逗号分隔的字符串，对于资源，请使用YAML数组。 `KafkaRebalance`	`BaseReplicaMovementStrategy`
`replicaMovementStrategies`		

更改默认设置会影响完成重新平衡所需的时间，以及重新平衡期间放置在Kafka群集上的负载。使用较低的值可以减少负载，但会增加花费的时间，反之亦然。
额外资源

- `CruiseControlSpec` 模式参考。
- `KafkaRebalanceSpec` 模式参考。

7.5。巡航控制配置

中的属性包含配置选项作为键，其值是以下JSON类型之一： `config` `Kafka.spec.cruiseControl`

- 串
- 数
- 布尔型

注意	看起来像JSON或YAML的字符串将需要显式引用。

除了由Strimzi直接管理的选项之外，您还可以指定和配置Cruise Control文档 “配置”部分中列出的所有选项。特别是，您**不能**使用等于或以下列字符串之一开头的键来修改配置选项：

- bootstrap.servers
- zookeeper.
- ssl.
- security.
- failed.brokers.zk.path
- webserver.http.port
- webserver.http.address
- webserver.api.urlprefix
- metric.reporter.sampler.bootstrap.servers
- metric.reporter.topic
- metric.reporter.topic.pattern
- partition.metric.sample.store.topic
- broker.metric.sample.store.topic
- capacity.config.file
- skip.sample.store.topic.rack.awareness.check
- cruise.control.metrics.topic
- sasl.

如果指定了受限制的选项，则将忽略这些选项，并在Cluster Operator日志文件中显示一条警告消息。所有受支持的选项都传递给Cruise Control。
巡航控制配置示例

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    # ...
    config:
      default.goals: >
         com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
         com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal
      cpu.balance.threshold: 1.1
      metadata.max.age.ms: 300000
      send.buffer.bytes: 131072
    # ...
```

容量配置

巡航控制使用*容量限制*来确定资源分配的优化目标是否被破坏。此类型有四个目标：

- `DiskUsageDistributionGoal` -磁盘利用率分布
- `CpuUsageDistributionGoal` -CPU利用率分布
- `NetworkInboundUsageDistributionGoal` -网络入站利用率分布
- `NetworkOutboundUsageDistributionGoal` -网络出站利用率分布

您可以在中的属性中指定Kafka经纪人资源的容量限制。默认情况下启用它们，您可以更改其默认值。可以使用标准的Kubernetes字节单位（K，M，G和T）或它们的等值双字节（2的幂）（Ki，Mi，Gi和Ti）为以下代理资源设置容量限制： `brokerCapacity` `Kafka.spec.cruiseControl`

- `disk` -每个代理的磁盘存储（默认值：100000Mi）
- `cpuUtilization` -CPU利用率百分比（默认值：100）
- `inboundNetwork` -入站网络吞吐量，以每秒字节为单位（默认值：10000KiB / s）
- `outboundNetwork` -出站网络吞吐量，以每秒字节为单位（默认值：10000KiB / s）

由于Strimzi Kafka经纪人是同质的，因此Cruise Control将相同的容量限制应用于它所监视的每个经纪人。
使用双字节单元的示例Cruise Control brokerCapacity配置

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    # ...
    brokerCapacity:
      disk: 100Gi
      cpuUtilization: 100
      inboundNetwork: 10000KiB/s
      outboundNetwork: 10000KiB/s
    # ...
```

额外资源

有关更多信息，请参考架构参考。 `BrokerCapacity`

记录配置

Cruise Control有其自己的可配置记录器：

- `cruisecontrol.root.logger`

Cruise Control使用Apache 记录器实现。 `log4j`

使用该属性来配置记录器和记录器级别。 `logging`

您可以通过指定记录器和直接（内联）级别或使用自定义（外部）ConfigMap来设置日志级别。如果使用ConfigMap，则将属性设置为包含外部日志记录配置的ConfigMap的名称。在ConfigMap内部，使用来描述日志记录配置。 `logging.name` `log4j.properties`

在这里，我们看到和的示例。内联记录 `inline` `external`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
# ...
spec:
  cruiseControl:
    # ...
    logging:
      type: inline
      loggers:
        cruisecontrol.root.logger: "INFO"
    # ...
```

外部记录

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
# ...
spec:
  cruiseControl:
    # ...
    logging:
      type: external
      name: customConfigMap
    # ...
```

7.6。部署巡航控制

要将Cruise Control部署到您的Strimzi群集，请使用资源中的属性定义配置，然后创建或更新资源。 `cruiseControl` `Kafka`

每个Kafka集群部署一个Cruise Control实例。
先决条件

- Kubernetes集群
- 正在运行的集群 Operator

程序

1. 编辑资源并添加属性。 `Kafka` `cruiseControl`

   此示例配置中显示了可以配置的属性：

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     # ...
     cruiseControl:
       brokerCapacity: (1)
         inboundNetwork: 10000KB/s
         outboundNetwork: 10000KB/s
         # ...
       config: (2)
         default.goals: >
            com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
            com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal
            # ...
         cpu.balance.threshold: 1.1
         metadata.max.age.ms: 300000
         send.buffer.bytes: 131072
         # ...
       resources: (3)
         requests:
           cpu: 200m
           memory: 64Mi
         limits:
           cpu: 500m
           memory: 128Mi
       logging: (4)
           type: inline
   ```

```
      loggers:
        cruisecontrol.root.logger: "INFO"
    template: (5)
      pod:
        metadata:
          labels:
            label1: value1
        securityContext:
          runAsUser: 1000001
          fsGroup: 0
        terminationGracePeriodSeconds: 120
    readinessProbe: (6)
      initialDelaySeconds: 15
      timeoutSeconds: 5
    livenessProbe: (7)
      initialDelaySeconds: 15
      timeoutSeconds: 5
# ...
```

a. 指定代理资源的容量限制。有关更多信息，请参阅容量配置。
b. 定义定速巡航配置，包括默认优化目标（in ）和对主优化目标（in ）或硬性目标（in ）的任何自定义。除了由Strimzi直接管理的配置之外，您还可以提供任何标准的Cruise Control配置选项。有关配置优化目标的更多信息，请参阅《优化目标概述》。 `default.goals goals hard.goals`
c. 为定速巡航保留的CPU和内存资源。有关更多信息，请参阅CPU和内存资源。
d. 通过ConfigMap直接（内联）或间接（外部）添加定义的记录器和日志级别。必须将自定义ConfigMap放置在log4j.properties键下。Cruise Control有一个名为的记录器。您可以将日志级别设置为INFO，ERROR，WARN，TRACE，DEBUG，FATAL或OFF。有关更多信息，请参阅日志记录配置。 `cruisecontrol.root.logger`
e. 自定义部署模板和Pod。
f. 健康检查准备探针。
g. Healthcheck活力探针。

2. 创建或更新资源：

```
kubectl apply -f kafka.yaml
```

3. 验证Cruise Control已成功部署：

```
kubectl get deployments -l app.kubernetes.io/name=strimzi
```

自动创建的 Topic

下表显示了在部署Cruise Control时自动创建的三个 Topic 。这些 Topic 是Cruise Control正常运行所必需的，不能删除或更改。

表4.自动创建的 Topic

自动创建的 Topic	由...制作	功能
strimzi.cruisecontrol.metrics	Strimzi指标记者	将来自Metrics Reporter的原始指标存储在每个Kafka代理中。
strimzi.cruisecontrol.partitionmetricsamples	巡航控制	存储每个分区的派生指标。这些由Metric Sample Aggregator创建。
strimzi.cruisecontrol.modeltrainingsamples	巡航控制	存储用于创建集群工作量模型的指标样本。

为了防止删除Cruise Control所需的记录，在自动创建的 Topic 中禁用了日志压缩。
接下来做什么

配置和部署Cruise Control之后，您可以生成优化建议。额外资源

`CruiseControlTemplate` 模式参考。

7.7。生成优化建议

创建或更新资源时，Cruise Control会根据配置的优化目标为Kafka集群生成优化建议。 `KafkaRebalance`

分析优化建议中的信息并决定是否批准。
先决条件

- 您已经将Cruise Control部署到了Strimzi集群。
- 您已经配置了优化目标，并且可以选择配置代理资源的容量限制。

程序

1. 创建资源： `KafkaRebalance`

   a. 要使用资源中定义的*默认优化目标*，请将属性留空： `Kafka` `spec`

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaRebalance
metadata:
  name: my-rebalance
  labels:
```

```
    strimzi.io/cluster: my-cluster
spec: {}
```

b. 要配置*用户提供的优化目标*而不是使用默认目标，请添加属性并输入一个或多个目标。 `goals`

在以下示例中，机架感知和副本容量被配置为用户提供的优化目标：

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaRebalance
metadata:
  name: my-rebalance
  labels:
    strimzi.io/cluster: my-cluster
spec:
  goals:
    - RackAwareGoal
    - ReplicaCapacityGoal
```

c. 要忽略配置的硬目标，请添加属性： `skipHardGoalCheck: true`

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaRebalance
metadata:
  name: my-rebalance
  labels:
    strimzi.io/cluster: my-cluster
spec:
  goals:
    - RackAwareGoal
    - ReplicaCapacityGoal
skipHardGoalCheck: true
```

2. 创建或更新资源：

```
kubectl apply -f your-file
```

集群 Operator 向Cruise Control请求优化建议。这可能需要几分钟，具体取决于Kafka群集的大小。

3. 检查资源状态： `KafkaRebalance`

```
kubectl describe kafkarebalance rebalance-cr-name
```

巡航控制系统返回以下两种状态之一：

- `PendingProposal`：重新平衡 operator 正在轮询Cruise Control API，以检查优化建议是否准备就绪。
- `ProposalReady`：优化建议已准备好进行审核，并在需要时批准。优化建议包含在资源的属性中。 `Status.Optimization Result` `KafkaRebalance`

4. 查看优化建议。

```
kubectl describe kafkarebalance rebalance-cr-name
```

这是一个示例建议：

```
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
```

本节中的属性描述了挂起的群集重新平衡操作。有关每个属性的描述，请参见优化建议的内容。 `Optimization Result`

接下来做什么

批准优化建议

附加资源

- 优化建议概述

7.8。批准优化建议

如果状态为，则可以批准Cruise Control生成的优化建议。然后，Cruise Control将优化建议应用于Kafka集群，将分区重新分配给代理，并更改分区领导地位。 `ProposalReady`

警告	**这不是空想。** 在批准优化建议之前，您必须： • 如果提案已过时，请刷新。 • 仔细检查提案的内容。

先决条件

- 您已经从Cruise Control 生成了优化建议。
- 在自定义资源的状态。 `KafkaRebalance` `ProposalReady`

程序

对要批准的优化建议执行以下步骤：

1. 除非是新生成优化建议，否则请检查它是否基于有关Kafka集群状态的当前信息。为此，请刷新优化建议以确保其使用最新的集群指标：
    a. 使用以下注释在Kubernetes中的资源： `KafkaRebalance` `refresh`

    ```
    kubectl annotate kafkarebalance rebalance-cr-name strimzi.io/rebalance=refresh
    ```

    b. 检查资源状态： `KafkaRebalance`

    ```
    kubectl describe kafkarebalance rebalance-cr-name
    ```

    c. 等待状态更改为。 `ProposalReady`
2. 批准您要应用Cruise Control的优化建议。

    在Kubernetes中注释资源： `KafkaRebalance`

    ```
    kubectl annotate kafkarebalance rebalance-cr-name strimzi.io/rebalance=approve
    ```

3. 群集 Operator 检测带注释的资源，并指示Cruise Control重新平衡Kafka群集。
4. 检查资源状态： `KafkaRebalance`

    ```
    kubectl describe kafkarebalance rebalance-cr-name
    ```

5. 巡航控制返回以下三种状态之一：
    - 重新平衡：群集重新平衡操作正在进行中。
    - 就绪：集群重新平衡操作已成功完成。该自定义资源不能重复使用。 `KafkaRebalance`
    - NotReady：发生错误-请参阅解决资源问题。 `KafkaRebalance`

额外资源

- 优化建议概述
- 停止集群重新平衡

7.9。停止集群重新平衡

启动后，群集重新平衡操作可能需要一些时间才能完成并影响Kafka群集的整体性能。

如果要停止正在进行的群集重新平衡操作，请将注释应用于自定义资源。这指示Cruise Control完成当前批次的分区重新分配，然后停止重新平衡。当重新平衡停止时，已经应用了完整的分区重新分配；因此，与重新平衡操作开始之前相比，Kafka群集的状态有所不同。如果需要进一步的平衡，则应生成新的优化建议。 `stop` `KafkaRebalance`

注意	Kafka群集在中间（停止）状态下的性能可能会比在初始状态下差。

先决条件

- 通过使用注释自定义资源，您已经批准了优化建议。 `KafkaRebalance` `approve`
- 自定义资源的状态为。 `KafkaRebalance` `Rebalancing`

程序

1. 在Kubernetes中注释资源： `KafkaRebalance`

    ```
    kubectl annotate kafkarebalance rebalance-cr-name strimzi.io/rebalance=stop
    ```

2. 检查资源状态： `KafkaRebalance`

```
kubectl describe kafkarebalance rebalance-cr-name
```

3. 等待状态更改为。 `Stopped`

额外资源

- 优化建议概述

7.10。解决资源问题 `KafkaRebalance`

如果在创建资源或与Cruise Control交互时发生问题，则会在资源状态中报告错误以及如何修复该错误的详细信息。资源也移至状态。 `KafkaRebalance` `NotReady`

要继续集群重新平衡操作，必须在资源本身中解决该问题。问题可能包括以下内容： `KafkaRebalance`

- 参数配置错误。
- 巡航控制服务器无法访问。

解决此问题后，您需要将注释添加到资源中。在“刷新”期间，从定速巡航服务器请求新的优化建议。先决条件 `refresh` `KafkaRebalance`

- 您已经批准了优化建议。
- 用于重新平衡操作的自定义资源的状态为。 `KafkaRebalance` `NotReady`

程序

1. 从状态获取有关错误的信息： `KafkaRebalance`

   ```
   kubectl describe kafkarebalance rebalance-cr-name
   ```

2. 尝试解决资源中的问题。 `KafkaRebalance`
3. 在Kubernetes中注释资源： `KafkaRebalance`

   ```
   kubectl annotate kafkarebalance rebalance-cr-name strimzi.io/rebalance=refresh
   ```

4. 检查资源状态： `KafkaRebalance`

   ```
   kubectl describe kafkarebalance rebalance-cr-name
   ```

5. 等待状态更改为或直接更改为。 `PendingProposal` `ProposalReady`

额外资源

- 优化建议概述

8. 分布式跟踪

分布式跟踪使您可以跟踪分布式系统中应用程序之间的事务处理进度。在微服务架构中，跟踪跟踪服务之间的事务进度。跟踪数据对于监视应用程序性能以及调查目标系统和最终用户应用程序的问题很有用。

在Strimzi中，跟踪有助于消息的端到端跟踪：从源系统到Kafka，再从Kafka到目标系统和应用程序。它补充了可在Grafana仪表板以及组件记录器中查看的指标。

Strimzi如何支持跟踪

以下组件内置了对跟踪的支持：

- Kafka Connect（包括具有Source2Image支持的Kafka Connect）
- 镜匠
- MirrorMaker 2.0
- Strimzi Kafka桥

您可以使用自定义资源中的模板配置属性来启用和配置这些组件的跟踪。

为了能够在Kafka生产者，消费者和Kafka流API应用程序跟踪，您仪器使用的应用程序代码OpenTracing阿帕奇Kafka客户端工具库（包含Strimzi）。进行检测后，客户端会生成跟踪数据：例如，在生成消息或将偏移量写入日志时。

根据采样策略对轨迹进行采样，然后在Jaeger用户界面中将其可视化。

注意	Kafka经纪人不支持跟踪。 为Strimzi以外的应用程序和系统设置跟踪超出了本章的范围。要了解有关此 Topic 的更多信息，请在OpenTracing文档中搜索“注入和提取” 。

程序概要

要为Strimzi设置跟踪，请按以下步骤进行操作：

- 为客户设置跟踪：
    - 为Kafka客户端初始化Jaeger跟踪器
- 使用跟踪器的工具客户：
    - 仪器生产者和消费者进行追踪
    - Instrument Kafka Streams应用程序进行跟踪
- 设置MirrorMaker，Kafka Connect和Kafka Bridge的跟踪

先决条件

- Jaeger后端组件已部署到您的Kubernetes集群中。有关部署说明，请参阅Jaeger部署文档。

8.1。OpenTracing和Jaeger概述

Strimzi使用OpenTracing和Jaeger项目。

OpenTracing是一个API规范，独立于跟踪或监视系统。

- OpenTracing API用于*检测*应用程序代码
- 仪器化的应用程序为分布式系统中的单个事务生成*跟踪*
- 迹线由*跨度*组成，*跨度*定义了一段时间内的特定工作单元

Jaeger是用于基于微服务的分布式系统的跟踪系统。

- Jaeger实现了OpenTracing API，并提供了用于检测的客户端库
- Jaeger用户界面允许您查询，过滤和分析跟踪数据

额外资源

- 开放追踪
- 积家

8.2。为Kafka客户端设置跟踪

初始化Jaeger跟踪器，以对客户端应用程序进行分布式跟踪。

8.2.1。为Kafka客户端初始化Jaeger跟踪器

使用一组跟踪环境变量来配置和初始化Jaeger跟踪器。程序

在每个客户端应用程序中：

1. 将Jaeger的Maven依赖项添加到客户端应用程序的文件中： `pom.xml`

```
<dependency>
    <groupId>io.jaegertracing</groupId>
    <artifactId>jaeger-client</artifactId>
    <version>1.1.0</version>
</dependency>
```

2. 使用跟踪环境变量定义Jaeger跟踪器的配置。
3. 根据您在第二步中定义的环境变量创建Jaeger跟踪器：

```
Tracer tracer = Configuration.fromEnv().getTracer();
```

注意	有关初始化Jaeger跟踪器的替代方法，请参阅Java OpenTracing库文档。

4. 将Jaeger跟踪器注册为全局跟踪器：

```
GlobalTracer.register(tracer);
```

现在已初始化Jaeger跟踪器供客户端应用程序使用。

8.2.2。用于跟踪的环境变量

为Kafka客户端配置Jaeger跟踪器时，请使用这些环境变量。

注意	跟踪环境变量是Jaeger项目的一部分，并且会随时更改。有关最新的环境变量，请参见Jaeger文档。

属性	需要	描述
`JAEGER_SERVICE_NAME`	是	Jaeger跟踪器服务的名称。
`JAEGER_AGENT_HOST`	没有	通过用户数据报协议（UDP）与之通信的主机名。 `jaeger-agent`
`JAEGER_AGENT_PORT`	没有	用于通过UDP 与之通信的端口。 `jaeger-agent`

JAEGER_ENDPOINT	没有	该端点。仅当客户端应用程序将绕过并直接连接到时，才定义此变量。 traces jaeger-agent jaeger-collector
JAEGER_AUTH_TOKEN	没有	身份验证令牌作为承载令牌发送到端点。
JAEGER_USER	没有	如果使用基本身份验证，则发送到端点的用户名。
JAEGER_PASSWORD	没有	如果使用基本身份验证，则发送到端点的密码。
JAEGER_PROPAGATION	没有	以逗号分隔的格式列表，用于传播跟踪上下文。默认为标准Jaeger格式。有效值为和。 jaeger b3
JAEGER_REPORTER_LOG_SPANS	没有	指示报告程序是否也应记录跨度。
JAEGER_REPORTER_MAX_QUEUE_SIZE	没有	报告者的最大队列大小。
JAEGER_REPORTER_FLUSH_INTERVAL	没有	报告程序的刷新间隔，以毫秒为单位。定义Jaeger报告程序刷新间隔批次的频率。
JAEGER_SAMPLER_TYPE	没有	用于客户端跟踪的采样策略： • 不变 • 概率论 • 限速 • 远程（默认） 要对所有迹线进行采样，请使用常数采样策略（参数为1）。 有关更多信息，请参阅Jaeger文档。
JAEGER_SAMPLER_PARAM	没有	采样器参数（数字）。
JAEGER_SAMPLER_MANAGER_HOST_PORT	没有	如果选择了远程采样策略，则要使用的主机名和端口。
JAEGER_TAGS	没有	以逗号分隔的跟踪器级别的标记列表，这些标记已添加到所有报告的跨度中。 该值还可以使用format引用环境变量。是可选的，它标识在找不到环境变量时要使用的值。 ${envVarName:default} :default

额外资源

- 为Kafka客户端初始化Jaeger跟踪器

8.3。使用跟踪器检测Kafka客户

仪器Kafka生产者和消费者客户端，以及用于分布式跟踪的Kafka Streams API应用程序。

8.3.1。指导生产者和消费者进行跟踪

使用Decorator模式或Interceptor来检测Java生产者和使用者应用程序代码以进行跟踪。
程序

在每个生产者和使用者应用程序的应用程序代码中：

1. 将针对OpenTracing的Maven依赖项添加到生产者或使用者的文件中。 pom.xml

```
<dependency>
    <groupId>io.opentracing.contrib</groupId>
    <artifactId>opentracing-kafka-client</artifactId>
    <version>0.1.12</version>
</dependency>
```

2. 使用Decorator模式或Interceptors来检测您的客户端应用程序代码。

   - 要使用装饰器模式：

```
// Create an instance of the KafkaProducer:
KafkaProducer<Integer, String> producer = new KafkaProducer<>(senderProps);

// Create an instance of the TracingKafkaProducer:
TracingKafkaProducer<Integer, String> tracingProducer = new TracingKafkaProducer<>(producer,
        tracer);

// Send:
tracingProducer.send(...);

// Create an instance of the KafkaConsumer:
KafkaConsumer<Integer, String> consumer = new KafkaConsumer<>(consumerProps);

// Create an instance of the TracingKafkaConsumer:
```

```
TracingKafkaConsumer<Integer, String> tracingConsumer = new TracingKafkaConsumer<>(consumer,
        tracer);

// Subscribe:
tracingConsumer.subscribe(Collections.singletonList("messages"));

// Get messages:
ConsumerRecords<Integer, String> records = tracingConsumer.poll(1000);

// Retrieve SpanContext from polled record (consumer side):
ConsumerRecord<Integer, String> record = ...
SpanContext spanContext = TracingKafkaUtils.extractSpanContext(record.headers(), tracer);
```

- 要使用拦截器：

```
// Register the tracer with GlobalTracer:
GlobalTracer.register(tracer);

// Add the TracingProducerInterceptor to the sender properties:
senderProps.put(ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,
          TracingProducerInterceptor.class.getName());

// Create an instance of the KafkaProducer:
KafkaProducer<Integer, String> producer = new KafkaProducer<>(senderProps);

// Send:
producer.send(...);

// Add the TracingConsumerInterceptor to the consumer properties:
consumerProps.put(ConsumerConfig.INTERCEPTOR_CLASSES_CONFIG,
          TracingConsumerInterceptor.class.getName());

// Create an instance of the KafkaConsumer:
KafkaConsumer<Integer, String> consumer = new KafkaConsumer<>(consumerProps);

// Subscribe:
consumer.subscribe(Collections.singletonList("messages"));

// Get messages:
ConsumerRecords<Integer, String> records = consumer.poll(1000);

// Retrieve the SpanContext from a polled message (consumer side):
ConsumerRecord<Integer, String> record = ...
SpanContext spanContext = TracingKafkaUtils.extractSpanContext(record.headers(), tracer);
```

装饰器模式中的自定义跨度名称

甲*跨度*是在积工作，与操作名称的逻辑单元，开始时间和持续时间。

要使用Decorator模式来检测生产者和使用者应用程序，请在创建和对象时通过将对象作为附加参数传递来定义自定义范围名称。OpenTracing Apache Kafka客户端工具库包含几个内置的span名称。示例：使用自定义跨度名称以装饰器模式检测客户端应用程序代码 `BiFunction` `TracingKafkaProducer` `TracingKafkaConsumer`

```
// Create a BiFunction for the KafkaProducer that operates on (String operationName, ProducerRecord
consumerRecord) and returns a String to be used as the name:

BiFunction<String, ProducerRecord, String> producerSpanNameProvider =
    (operationName, producerRecord) -> "CUSTOM_PRODUCER_NAME";

// Create an instance of the KafkaProducer:
KafkaProducer<Integer, String> producer = new KafkaProducer<>(senderProps);

// Create an instance of the TracingKafkaProducer
TracingKafkaProducer<Integer, String> tracingProducer = new TracingKafkaProducer<>(producer,
        tracer,
        producerSpanNameProvider);

// Spans created by the tracingProducer will now have "CUSTOM_PRODUCER_NAME" as the span name.

// Create a BiFunction for the KafkaConsumer that operates on (String operationName, ConsumerRecord
consumerRecord) and returns a String to be used as the name:

BiFunction<String, ConsumerRecord, String> consumerSpanNameProvider =
    (operationName, consumerRecord) -> operationName.toUpperCase();

// Create an instance of the KafkaConsumer:
KafkaConsumer<Integer, String> consumer = new KafkaConsumer<>(consumerProps);

// Create an instance of the TracingKafkaConsumer, passing in the consumerSpanNameProvider BiFunction:

TracingKafkaConsumer<Integer, String> tracingConsumer = new TracingKafkaConsumer<>(consumer,
        tracer,
        consumerSpanNameProvider);
```

```
// Spans created by the tracingConsumer will have the operation name as the span name, in upper-case.
// "receive" -> "RECEIVE"
```

内置范围名称

定义自定义范围名称时，可以在该类中使用以下内容。如果没有指定，并且被使用。 `BiFunctions` `ClientSpanNameProvider` `spanNameProvider` `CONSUMER_OPERATION_NAME` `PRODUCER_OPERATION_NAME`

双功能	描述
`CONSUMER_OPERATION_NAME, PRODUCER_OPERATION_NAME`	作为跨度名称返回：对于消费者，“接收”，对于生产者，“发送”。 `operationName`
`CONSUMER_PREFIXED_OPERATION_NAME(String prefix), PRODUCER_PREFIXED_OPERATION_NAME(String prefix)`	返回的字符串连接和。 `prefix` `operationName`
`CONSUMER_TOPIC, PRODUCER_TOPIC`	以格式返回消息发送到或从中检索到的 Topic 的名称。 `(record.topic())`
`PREFIXED_CONSUMER_TOPIC(String prefix), PREFIXED_PRODUCER_TOPIC (String prefix)`	以格式返回的字符串串联和 Topic 名称。 `prefix (record.topic())`
`CONSUMER_OPERATION_NAME_TOPIC, PRODUCER_OPERATION_NAME_TOPIC`	返回操作名称和 Topic 名称：。 `"operationName - record.topic()"`
`CONSUMER_PREFIXED_OPERATION_NAME_TOPIC(String prefix), PRODUCER_PREFIXED_OPERATION_NAME_TOPIC(String prefix)`	返回的字符串连接和。 `prefix` `"operationName - record.topic()"`

8.3.2。检测Kafka Streams应用程序以进行跟踪

本节介绍如何检测Kafka Streams API应用程序以进行分布式跟踪。
程序

在每个Kafka Streams API应用程序中：

1. 将依赖项添加到您的Kafka Streams API应用程序的pom.xml文件中： `opentracing-kafka-streams`

   ```
   <dependency>
       <groupId>io.opentracing.contrib</groupId>
       <artifactId>opentracing-kafka-streams</artifactId>
       <version>0.1.12</version>
   </dependency>
   ```

2. 创建供应商界面的实例： `TracingKafkaClientSupplier`

   ```
   KafkaClientSupplier supplier = new TracingKafkaClientSupplier(tracer);
   ```

3. 提供供应商界面： `KafkaStreams`

   ```
   KafkaStreams streams = new KafkaStreams(builder.build(), new StreamsConfig(config), supplier);
   streams.start();
   ```

8.4。设置MirrorMaker，Kafka Connect和Kafka Bridge的跟踪

MirrorMaker，MirrorMaker 2.0，Kafka Connect（包括支持Source2Image的Kafka Connect）和Strimzi Kafka Bridge支持分布式跟踪。
MirrorMaker和MirrorMaker 2.0中的跟踪

对于MirrorMaker和MirrorMaker 2.0，消息是从源群集跟踪到目标群集的。跟踪数据记录进入和离开MirrorMaker或MirrorMaker 2.0组件的消息。
Kafka Connect中的跟踪

仅跟踪由Kafka Connect本身产生和使用的消息。要跟踪在Kafka Connect和外部系统之间发送的消息，必须在这些系统的连接器中配置跟踪。有关更多信息，请参阅Kafka Connect群集配置。在Kafka桥中追踪

Kafka桥产生和使用的消息将被跟踪。还将跟踪来自客户端应用程序的传入HTTP请求，以通过Kafka Bridge发送和接收消息。要进行端到端跟踪，必须在HTTP客户端中配置跟踪。

8.4.1。在MirrorMaker，Kafka Connect和Kafka Bridge资源中启用跟踪

更新的配置，，，，和自定义的资源，并指定每个资源配置积追踪服务。在Kubernetes集群中更新启用了跟踪的资源会触发两个事件： `KafkaMirrorMaker` `KafkaMirrorMaker2` `KafkaConnect` `KafkaConnectS2I` `KafkaBridge`

- MirrorMaker，MirrorMaker 2.0，Kafka Connect或Strimzi Kafka Bridge中集成的使用者和生产者中的拦截器类已更新。
- 对于MirrorMaker，MirrorMaker 2.0和Kafka Connect，跟踪代理根据资源中定义的跟踪配置初始化Jaeger跟踪器。
- 对于Kafka Bridge，由Kafka Bridge本身初始化基于资源中定义的跟踪配置的Jaeger跟踪器。

程序

执行这些步骤为每个，，，，和资源。 `KafkaMirrorMaker` `KafkaMirrorMaker2` `KafkaConnect` `KafkaConnectS2I` `KafkaBridge`

1. 在属性中，配置Jaeger跟踪器服务。例如：Kafka Connect的Jaeger跟踪器配置 `spec.template`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  template:
    connectContainer: (1)
      env:
        - name: JAEGER_SERVICE_NAME
          value: my-jaeger-service
        - name: JAEGER_AGENT_HOST
          value: jaeger-agent-name
        - name: JAEGER_AGENT_PORT
          value: "6831"
  tracing: (2)
    type: jaeger
  #...
```

MirrorMaker的Jaeger跟踪器配置

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaMirrorMaker
metadata:
  name: my-mirror-maker
spec:
  #...
  template:
    mirrorMakerContainer:
      env:
        - name: JAEGER_SERVICE_NAME
          value: my-jaeger-service
        - name: JAEGER_AGENT_HOST
          value: jaeger-agent-name
        - name: JAEGER_AGENT_PORT
          value: "6831"
  tracing:
    type: jaeger
#...
```

MirrorMaker 2.0的Jaeger跟踪器配置

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaMirrorMaker2
metadata:
  name: my-mm2-cluster
spec:
  #...
  template:
    connectContainer:
      env:
        - name: JAEGER_SERVICE_NAME
          value: my-jaeger-service
        - name: JAEGER_AGENT_HOST
          value: jaeger-agent-name
        - name: JAEGER_AGENT_PORT
          value: "6831"
  tracing:
    type: jaeger
#...
```

Kafka桥的Jaeger示踪剂配置

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  #...
  template:
    bridgeContainer:
      env:
        - name: JAEGER_SERVICE_NAME
          value: my-jaeger-service
        - name: JAEGER_AGENT_HOST
          value: jaeger-agent-name
        - name: JAEGER_AGENT_PORT
          value: "6831"
  tracing:
```

```
      type: jaeger
#...
```

a. 使用跟踪环境变量作为模板配置属性。
b. 将属性设置为。 `spec.tracing.type` `jaeger`

2. 创建或更新资源：

```
kubectl apply -f your-file
```

额外资源

- `ContainerTemplate` 模式参考
- 自定义Kubernetes资源

9. 安全性

Strimzi支持使用TLS协议在Kafka和Strimzi组件之间进行加密通信。Kafka代理之间，代理ZooKeeper节点之间（节点间通信）以及这些代理与Strimzi operator 之间的通信始终进行加密。Kafka客户端和Kafka代理之间的通信根据群集的配置方式进行加密。对于Kafka和Strimzi组件，TLS证书也用于身份验证。

群集 Operator 会自动设置并更新TLS证书，以在群集内启用加密和身份验证。如果要在Kafka代理和客户端之间启用加密或TLS身份验证，它还会设置其他TLS证书。用户提供的证书不会更新。

您可以为TLS侦听器或启用了TLS加密的外部侦听器提供自己的服务器证书，称为*Kafka侦听器证书*。有关更多信息，请参阅Kafka侦听器证书。

图5. 由TLS保护的通信的示例架构图。

9.1。证书颁发机构

为了支持加密，每个Strimzi组件都需要自己的私钥和公钥证书。所有组件证书均由称为*群集CA*的内部证书颁发机构（CA）签名。

同样，使用TLS客户端身份验证连接到Strimzi的每个Kafka客户端应用程序都需要提供私钥和证书。第二个内部CA称为*客户端CA*，用于为Kafka客户端签名证书。

9.1.1。CA证书

群集CA和客户端CA都具有自签名的公钥证书。

Kafka代理配置为信任由群集CA或客户端CA签名的证书。客户端不需要连接的组件（例如ZooKeeper）仅信任由群集CA签名的证书。除非为外部侦听器禁用了TLS加密，否则客户端应用程序必须信任由群集CA签名的证书。对于执行相互TLS身份验证的客户端应用程序也是如此。

默认情况下，Strimzi自动生成并更新由群集CA或客户端CA颁发的CA证书。您可以在和对象中配置这些CA证书的管理。用户提供的证书不会更新。 `Kafka.spec.clusterCa` `Kafka.spec.clientsCa`

您可以为群集CA或客户端CA提供自己的CA证书。有关更多信息，请参阅安装自己的CA证书。如果您提供自己的证书，则必须在需要时手动更新它们。

9.1.2。安装自己的CA证书

此过程介绍如何安装自己的CA证书和密钥，而不是使用群集 Operator 生成的CA证书和私钥。

您可以使用此过程来安装自己的群集或客户端CA证书。

该过程描述了以PEM格式更新CA证书。您也可以使用PKCS＃12格式的证书。
先决条件

- 群集 Operator 正在运行。
- Kafka集群尚未部署。
- 群集CA或客户端CA的PEM格式的X.509证书和密钥。
    - 如果要使用不是根CA的群集或客户端CA，则必须在证书文件中包括整个链。链条应按以下顺序排列：
        1. 群集或客户端CA
        2. 一个或多个中间CA
        3. 根CA
    - 在X509v3基本约束中，应将链中的所有CA配置为CA。

程序

1. 将您的CA证书放在相应的中。 `Secret`
    a. 删除现有机密：

    ```
    kubectl delete secret CA-CERTIFICATE-SECRET
    ```

    CA证书秘密 是的名称，这是针对集群CA证书，并为客户CA证书。 `Secret` *`CLUSTER-NAME`*`-cluster-ca-cert` *`CLUSTER-NAME`*`-clients-ca-cert`

    忽略任何“不存在”错误。

b. 创建并标记新秘密

```
kubectl create secret generic CA-CERTIFICATE-SECRET --from-file=ca.crt=CA-CERTIFICATE-FILENAME
```

2. 将您的CA密钥放入相应的中。 Secret

a. 删除现有机密：

```
kubectl delete secret CA-KEY-SECRET
```

CA-KEY-SECRET 是CA密钥的名称，用于群集CA密钥和客户端CA密钥。 *CLUSTER-NAME*-cluster-ca *CLUSTER-NAME*-clients-ca

b. 创建新的秘密：

```
kubectl create secret generic CA-KEY-SECRET --from-file=ca.key=CA-KEY-SECRET-FILENAME
```

3. 用标签和标记秘密： strimzi.io/kind=Kafka strimzi.io/cluster=*CLUSTER-NAME*

```
kubectl label secret CA-CERTIFICATE-SECRET strimzi.io/kind=Kafka strimzi.io/cluster=CLUSTER-NAME
kubectl label secret CA-KEY-SECRET strimzi.io/kind=Kafka strimzi.io/cluster=CLUSTER-NAME
```

4. 为群集创建资源，将或对象配置为不使用生成的CA：片段资源示例，将群集CA配置为使用您自己提供的证书 Kafka Kafka.spec.clusterCa Kafka.spec.clientsCa Kafka

```
kind: Kafka
version: kafka.strimzi.io/v1beta1
spec:
  # ...
  clusterCa:
    generateCertificateAuthority: false
```

额外资源

- 要续订以前安装的CA证书，请参阅续订自己的CA证书。
- 提供您自己的Kafka侦听器证书。

9.2。机密

Strimzi使用*Secrets*为Kafka群集组件和客户端存储私钥和证书。机密用于在Kafka代理之间以及代理与客户端之间建立TLS加密连接。它们还用于相互TLS身份验证。

- 甲*集群机密*包含一个集群CA证书来签名Kafka经纪人证书，并且用于通过连接的客户端建立与Kafka簇来验证代理标识的TLS加密连接。
- 一个*客户端密钥*包含用户登录自己的客户端证书，以便对Kafka集群相互认证客户端CA证书。代理通过客户端CA证书本身验证客户端身份。
- 一个*用户的机密*包含私钥和证书，创建一个新用户时被生成并签署客户端CA证书。访问集群时，密钥和证书用于身份验证和授权。

机密提供PEM和PKCS＃12格式的私钥和证书。使用PEM格式的私钥和证书意味着用户必须从Secrets中获取它们，并生成相应的信任库（或密钥库）以在其Java应用程序中使用。PKCS＃12存储提供了可直接使用的信任库（或密钥库）。

所有密钥的大小均为2048位。

9.2.1。PKCS＃12存储

PKCS＃12定义了存档文件格式（.p12），用于将密码对象存储到具有密码保护的单个文件中。您可以使用PKCS＃12在一个地方管理证书和密钥。

每个密钥包含特定于PKCS＃12的字段。

- 该字段包含证书和密钥。 .p12
- 该字段是保护档案的密码。 .password

9.2.2。群集CA机密

下表描述了由Kafka群集中的群集 Operator 管理的群集机密。

客户只需要使用秘密。描述的所有其他内容仅需要由Strimzi组件访问。您可以根据需要使用Kubernetes基于角色的访问控制来强制执行此操作。 *<cluster>*-cluster-ca-cert Secrets

表5. 机密中的字段 *<cluster>*-cluster-ca

领域	描述
ca.key	群集CA的当前私钥。

表6. 机密中的字段 *<cluster>*-cluster-ca-cert

领域	描述
ca.p12	PKCS＃12存档文件，用于存储证书和密钥。

ca.password	用于保护PKCS＃12存档文件的密码。
ca.crt	群集CA的当前证书。

注意	Kafka客户端应用程序必须信任其中的CA证书，以便在通过TLS连接到Kafka代理时，它们可以验证Kafka代理证书。 <cluster>-cluster-ca-cert

表7. 机密中的字段 <cluster>-kafka-brokers

领域	描述
<cluster>-kafka-<num>.p12	PKCS＃12存档文件，用于存储证书和密钥。
<cluster>-kafka-<num>.password	用于保护PKCS＃12存档文件的密码。
<cluster>-kafka-<num>.crt	Kafka经纪人pod <num>的证书。由中的当前或以前的群集CA私钥签名。 <cluster>-cluster-ca
<cluster>-kafka-<num>.key	Kafka经纪人pod的私钥。 <num>

表8. 秘密中的字段 <cluster>-zookeeper-nodes

领域	描述
<cluster>-zookeeper-<num>.p12	PKCS＃12存档文件，用于存储证书和密钥。
<cluster>-zookeeper-<num>.password	用于保护PKCS＃12存档文件的密码。
<cluster>-zookeeper-<num>.crt	ZooKeeper节点<num>的证书。由中的当前或以前的群集CA私钥签名。 <cluster>-cluster-ca
<cluster>-zookeeper-<num>.key	ZooKeeper pod的私钥。 <num>

表9. 机密中的字段 <cluster>-entity-operator-certs

领域	描述
entity-operator_.p12	PKCS＃12存档文件，用于存储证书和密钥。
entity-operator_.password	用于保护PKCS＃12存档文件的密码。
entity-operator_.crt	实体 Operator 与Kafka或ZooKeeper之间的TLS通信证书。由中的当前或以前的群集CA私钥签名。 <cluster>-cluster-ca
entity-operator.key	实体 Operator 与Kafka或ZooKeeper之间的TLS通信的私钥。

9.2.3。客户CA机密

表10. <cluster>中由群集 Operator 管理的客户端CA Secrets

秘密名称	秘密内的领域	描述
<cluster>-clients-ca	ca.key	客户端CA的当前私钥。
<cluster>-clients-ca-cert	ca.p12	PKCS＃12存档文件，用于存储证书和密钥。
	ca.password	用于保护PKCS＃12存档文件的密码。
	ca.crt	客户端CA的当前证书。

其中的证书是Kafka经纪人信任的证书。 <cluster>-clients-ca-cert

注意	<cluster>-clients-ca 用于签署客户端应用程序的证书。如果您打算在不使用用户 Operator 的情况下颁发应用程序证书，则Strimzi组件必须可以访问它并进行管理访问。您可以根据需要使用Kubernetes基于角色的访问控制来强制执行此操作。

9.2.4。用户机密

表11. 由用户 Operator 管理 Secrets

秘密名称	秘密内的领域	描述
<user>	user.p12	PKCS＃12存档文件，用于存储证书和密钥。
	user.password	用于保护PKCS＃12存档文件的密码。

	user.crt	用户证书，由客户端CA签名
	user.key	用户的私钥

9.3。证书续签和有效期

群集CA和客户端CA证书仅在有限的时间段（即有效期）内有效。通常将其定义为自生成证书以来的天数。

对于自动生成的CA证书，您可以配置以下有效期：

- 中的群集CA证书 `Kafka.spec.clusterCa.validityDays`
- 中的客户端CA证书 `Kafka.spec.clientsCa.validityDays`

这两个证书的默认有效期为365天。手动安装的CA证书应定义自己的有效期限。

当CA证书过期时，仍然信任该证书的组件和客户端将不接受来自其证书由CA私钥签名的对等方的TLS连接。组件和客户端需要改为信任*新的* CA证书。

为了在不丢失服务的情况下更新CA证书，群集 Operator 将在旧的CA证书过期之前启动证书更新。

您可以配置以下内容的续订时间：

- 中的群集CA证书 `Kafka.spec.clusterCa.renewalDays`
- 中的客户端CA证书 `Kafka.spec.clientsCa.renewalDays`

这两个证书的默认续订期限为30天。

续订期限从当前证书的到期日期开始倒计时。
有效期与续期

```
Not Before                                      Not After
    |                                               |
    |<--------------- validityDays ---------------->|
                              <--- renewalDays --->|
```

要在创建Kafka集群后更改有效期和续订期限，请配置并应用自定义资源，然后手动续订CA证书。如果您不手动更新证书，则下一次自动更新证书时将使用新的期限。证书有效期和续订时间的示例Kafka配置 `Kafka`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
# ...
spec:
# ...
  clusterCa:
    renewalDays: 30
    validityDays: 365
    generateCertificateAuthority: true
  clientsCa:
    renewalDays: 30
    validityDays: 365
    generateCertificateAuthority: true
# ...
```

群集 Operator 在续订期间的行为取决于是否在或中启用了相关设置。 `Kafka.spec.clusterCa.generateCertificateAuthority` `Kafka.spec.clientsCa.generateCertificateAuthority`

9.3.1。生成CA的续订过程

群集 Operator 执行以下过程来更新CA证书：

1. 生成一个新的CA证书，但保留现有的密钥。新证书将旧证书替换为相应证书中的名称。 `ca.crt` `Secret`
2. 生成新的客户端证书（用于ZooKeeper节点，Kafka代理和实体 Operator ）。由于签名密钥没有更改，因此这不是严格必需的，但是它使客户端证书的有效期与CA证书保持同步。
3. 重新启动ZooKeeper节点，以便它们将信任新的CA证书并使用新的客户端证书。
4. 重新启动Kafka代理，以便它们将信任新的CA证书并使用新的客户端证书。
5. 重新启动 Topic 和用户 Operator ，以便他们将信任新的CA证书并使用新的客户端证书。

9.3.2。客户申请

群集 Operator 不知道使用Kafka群集的客户端应用程序。

连接到群集并确保它们正确运行时，客户端应用程序必须：

- 信任在*<cluster>* -cluster-ca-cert Secret中发布的群集CA证书。
- 使用在其*<用户名>*机密中发布的凭据连接到群集。

  用户密钥以PEM和PKCS＃12格式提供凭据，或者在使用SCRAM-SHA身份验证时可以提供密码。创建用户时，用户 Operator 将创建用户凭据。

对于在同一Kubernetes集群和名称空间中运行的工作负载，可以将Secrets作为卷安装，因此客户端Pod可以根据Secrets的当前状态构造其密钥库和信任库。
有关此过程的更多详细信息，请参阅配置内部客户信任集群CA。

客户证书续订

您必须确保客户端在证书续订后继续工作。续订过程取决于客户端的配置方式。

如果要手动设置客户端证书和密钥，则必须生成新的客户端证书，并确保客户端在续订期内使用新证书。在续订期结束之前未能执行此操作可能会导致客户端应用程序无法连接到群集。

9.3.3。手动续订CA证书

群集和客户端CA证书会在其各自的证书续订期开始时自动续订。如果和设置为，则CA证书不会自动更新。 `Kafka.spec.clusterCa.generateCertificateAuthority` `Kafka.spec.clientsCa.generateCertificateAuthority` `false`

您可以在证书续订期开始之前手动续订这两个证书中的一个或两个。您可能出于安全原因，或者如果您更改了证书的续约或有效期，可能会这样做。

续订的证书使用与旧证书相同的私钥。
先决条件

- 群集 Operator 正在运行。
- 安装有CA证书和私钥的Kafka群集。

程序

1. 将注释应用于包含要续订的CA证书的。 `strimzi.io/force-renew` `Secret`

   表12.强制更新证书的机密注释

证书	秘密	注释命令
群集CA	*KAFKA群集名称* `-cluster-ca-cert`	`kubectl annotate secret KAFKA-CLUSTER-NAME-cluster-ca-cert strimzi.io/force-renew=true`
客户CA	*KAFKA群集名称* `-clients-ca-cert`	`kubectl annotate secret KAFKA-CLUSTER-NAME-clients-ca-cert strimzi.io/force-renew=true`

   在下一个对帐中，群集 Operator 将为您注释的生成新的CA证书。如果配置了维护时间窗口，则群集 Operator 将在下一个维护时间窗口内的第一次对帐中生成新的CA证书。 `Secret`

   客户端应用程序必须重新加载群集和由群集 Operator 更新的客户端CA证书。
2. 检查CA证书有效期：

   例如，使用命令： `openssl`

   ```
   kubectl get secret CA-CERTIFICATE-SECRET -o 'jsonpath={.data.CA-CERTIFICATE}' | base64 -d | openssl x509 -subject -issuer -startdate -enddate -noout
   ```

   CA证书秘密 是的名称，这是针对集群CA证书，并为客户CA证书。 `Secret` `KAFKA-CLUSTER-NAME-cluster-ca-cert` `KAFKA-CLUSTER-NAME-clients-ca-cert`

   CA-CERTIFICATE 是CA证书的名称，例如。 `jsonpath={.data.ca\.crt}`

   该命令返回和日期，这是CA证书的有效期。 `notBefore` `notAfter`

   例如，对于群集CA证书：

   ```
   subject=O = io.strimzi, CN = cluster-ca v0
   issuer=O = io.strimzi, CN = cluster-ca v0
   notBefore=Jun 30 09:43:54 2020 GMT
   notAfter=Jun 30 09:43:54 2021 GMT
   ```
3. 从机密中删除旧证书。

   当组件使用新证书时，旧证书可能仍处于活动状态。删除旧证书以消除任何潜在的安全风险。

额外资源

- 机密
- 维护时间窗口，用于滚动更新
- `CertificateAuthority` 模式参考

9.3.4。续订自己的CA证书

此过程描述了如何续订先前安装的CA证书和密钥。您需要在续订期间遵循此过程，以替换即将过期的CA证书。

您可以使用此过程来安装自己的群集或客户端CA证书。

该过程描述了以PEM格式更新CA证书。您也可以使用PKCS＃12格式的证书。
先决条件

- 群集 Operator 正在运行。
- 一个Kafka群集，您之前在其中安装了自己的CA证书和私钥。
- PEM格式的新群集和客户端X.509证书和密钥。

这些可以使用命令生成，例如： `openssl`

```
openssl req -x509 -new -days NUMBER-OF-DAYS-VALID --nodes -out ca.crt -keyout ca.key
```

程序

1. 在中检查当前CA证书的详细信息： `Secret`

   ```
   kubectl describe secret CA-CERTIFICATE-SECRET
   ```

   CA证书秘密 是的名称，这是针对集群CA证书，并为客户CA证书。 `Secret` *KAFKA-CLUSTER-NAME*-cluster-ca-cert *KAFKA-CLUSTER-NAME*-clients-ca-cert
2. 创建一个目录，以在机密中包含现有的CA证书。

   ```
   mkdir new-ca-cert-secret
   cd new-ca-cert-secret
   ```

3. 为您要续订的每个CA证书获取机密：

   ```
   kubectl get secret CA-CERTIFICATE-SECRET -o 'jsonpath={.data.CA-CERTIFICATE}' | base64 -d > CA-CERTIFICATE
   ```

   将*CA-CERTIFICATE*替换为每个CA证书的名称。
4. 将旧文件重命名为，其中*DATE*是格式为*YEAR-MONTH-DAYTHOUR-MINUTE-SECONDZ*的证书到期日期。 `ca.crt` `ca-DATE.crt`

   例如。 `ca-2018-09-27T17-32-00Z.crt`

   ```
   mv ca.crt ca-$(date -u -d$(openssl x509 -enddate -noout -in ca.crt | sed 's/.*=//') +'%Y-%m-%dT%H-%M-%SZ').
   crt
   ```

5. 将新的CA证书复制到目录中，命名为： `ca.crt`

   ```
   cp PATH-TO-NEW-CERTIFICATE ca.crt
   ```

6. 将您的CA证书放在相应的中。 `Secret`

   a. 删除现有机密：

      ```
      kubectl delete secret CA-CERTIFICATE-SECRET
      ```

      CA-CERTIFICATE-SECRET 是第一步中返回的的名称。 `Secret`

      忽略任何“不存在”错误。
   b. 重新创建秘密：

      ```
      kubectl create secret generic CA-CERTIFICATE-SECRET --from-file=.
      ```

7. 删除您创建的目录：

   ```
   cd ..
   rm -r new-ca-cert-secret
   ```

8. 将您的CA密钥放入相应的中。 `Secret`

   a. 删除现有机密：

      ```
      kubectl delete secret CA-KEY-SECRET
      ```

      CA-KEY-SECRET 是CA密钥的名称，用于群集CA密钥和客户端CA密钥。 *KAFKA-CLUSTER-NAME*-cluster-ca *KAFKA-CLUSTER-NAME*-clients-ca
   b. 使用新的CA密钥重新创建密钥：

      ```
      kubectl create secret generic CA-KEY-SECRET --from-file=ca.key=CA-KEY-SECRET-FILENAME
      ```

9. 用标签和标记秘密： `strimzi.io/kind=Kafka` `strimzi.io/cluster=KAFKA-CLUSTER-NAME`

   ```
   kubectl label secret CA-CERTIFICATE-SECRET strimzi.io/kind=Kafka strimzi.io/cluster=KAFKA-CLUSTER-NAME
   kubectl label secret CA-KEY-SECRET strimzi.io/kind=Kafka strimzi.io/cluster=KAFKA-CLUSTER-NAME
   ```

9.4。更换私钥

您可以替换群集CA和客户端CA证书使用的私钥。替换私钥后，群集 Operator 将为新私钥生成一个新的CA证书。
先决条件

- 群集 Operator 正在运行。
- 安装有CA证书和私钥的Kafka群集。

程序

- 将注释应用到包含要更新的私钥的。 `strimzi.io/force-replace` `Secret`

表13.替换私钥的命令

的私钥	秘密	注释命令
群集CA	*〈群集名称〉* -cluster-ca	`kubectl annotate secret <cluster-name>-cluster-ca strimzi.io/force-replace=true`
客户CA	*〈群集名称〉* -clients-ca	`kubectl annotate secret <cluster-name>-clients-ca strimzi.io/force-replace=true`

在下一次对帐中，集群 Operator 将：

- 为您注释的生成新的私钥 `Secret`
- 生成新的CA证书

如果配置了维护时间窗口，则群集 Operator 将在下一个维护时间窗口内的第一次对帐中生成新的私钥和CA证书。

客户端应用程序必须重新加载群集和由群集 Operator 更新的客户端CA证书。
额外资源

- 机密
- 维护时间窗口，用于滚动更新

9.5。TLS连接

9.5.1。ZooKeeper通讯

所有端口上的ZooKeeper节点之间以及客户端和ZooKeeper之间的通信都是加密的。

9.5.2。Kafka经纪人之间的沟通

Kafka代理之间的通信是通过端口9091上的内部侦听器完成的，该端口默认情况下是加密的，Kafka客户端无法访问。

Kafka代理与ZooKeeper节点之间的通信也已加密。

9.5.3。 Topic 和用户 operator

所有 Operator 均使用加密技术与Kafka和ZooKeeper进行通信。在 Topic 和用户 Operator 中，与ZooKeeper通信时使用TLS辅助工具。

9.5.4。巡航控制

Cruise Control使用加密与Kafka和ZooKeeper进行通信。与ZooKeeper通信时，使用TLS边车。

9.5.5。Kafka客户端连接

可通过配置侦听器来提供Kafka代理与在同一Kubernetes集群中运行的客户端之间的加密通信，该侦听器在端口9093上进行侦听。 `spec.kafka.listeners.tls`

Kafka代理与在同一Kubernetes集群之外运行的客户端之间的加密通信可以通过配置侦听器（侦听器的端口取决于其类型）来提供。 `spec.kafka.listeners.external` `external`

注意	可以通过配置与代理的未加密客户端通信，该代理在端口9092上侦听。 `spec.kafka.listeners.plain`

9.6。配置内部客户端以信任群集CA

此过程描述了如何配置位于Kubernetes群集中的Kafka客户端（连接到端口9093上的侦听器）以信任群集CA证书。 `tls`

对于内部客户端，最简单的方法是使用卷装入来访问包含必需的证书和密钥。 `Secrets`

请按照以下步骤配置由群集CA为基于Java的Kafka Producer，Consumer和Streams API签名的信任证书。

根据群集CA的证书格式选择要遵循的步骤：PKCS＃12（`.p12`）或PEM（`.crt`）。

这些步骤描述了如何将验证Kafka群集身份的群集机密安装到客户端吊舱。
先决条件

- 群集 Operator 必须正在运行。
- Kubernetes集群中需要有一个资源。 `Kafka`
- 您需要在Kubernetes集群中使用一个Kafka客户端应用程序，该应用程序将使用TLS连接，并且需要信任集群CA证书。
- 客户端应用程序必须在与资源相同的名称空间中运行。 `Kafka`

使用PKCS＃12格式（.p12）

1. 定义客户端吊舱时，将群集密钥安装为卷。

   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-cert
   ```

   在这里，我们正在安装：

   - 将PKCS＃12文件转换为精确路径，可以对其进行配置
   - 密码放入环境变量中，可在其中用于Java配置

2. 使用以下属性配置Kafka客户端：

   - 安全协议选项：
     - `security.protocol: SSL` 使用TLS进行加密时（使用或不使用TLS身份验证）。
     - `security.protocol: SASL_SSL` 通过TLS使用SCRAM-SHA身份验证时。
   - `ssl.truststore.location` 导入证书的信任库位置。
   - `ssl.truststore.password` 和用于访问信任库的密码。
   - `ssl.truststore.type=PKCS12` 识别信任库类型。

使用PEM格式（.crt）

1. 定义客户端吊舱时，将群集密钥安装为卷。

   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-cert
   ```

2. 将证书与使用X.509格式证书的客户端一起使用。

9.7。配置外部客户端以信任群集CA

此过程描述了如何配置位于Kubernetes群集外部的Kafka客户端（连接到端口9094上的侦听器）以信任群集CA证书。设置客户端时以及在续订期间（替换旧客户端CA证书），请遵循以下过程。 `external`

请按照以下步骤配置由群集CA为基于Java的Kafka Producer，Consumer和Streams API签名的信任证书。

根据群集CA的证书格式选择要遵循的步骤：PKCS＃12（`.p12`）或PEM（`.crt`）。

这些步骤描述了如何从“群集密钥”获得证书，以验证Kafka群集的身份。

重要	在将包含在CA证书续展期超过一个CA证书。客户必须将它们*全部*添加到其信任库中。 `<cluster-name>-cluster-ca-cert` `Secret`

先决条件

- 群集 Operator 必须正在运行。

- Kubernetes集群中需要有一个资源。 `Kafka`
- 您需要在Kubernetes群集之外的Kafka客户端应用程序，该应用程序将使用TLS连接，并且需要信任群集CA证书。

使用PKCS＃12格式（.p12）

1. 从生成的密钥中提取群集CA证书和密码。 `<cluster-name>-cluster-ca-cert`

   ```
   kubectl get secret <cluster-name>-cluster-ca-cert -o jsonpath='{.data.ca\.p12}' | base64 -d > ca.p12

   kubectl get secret <cluster-name>-cluster-ca-cert -o jsonpath='{.data.ca\.password}' | base64 -d > ca.
   password
   ```

2. 使用以下属性配置Kafka客户端：

   - 安全协议选项：
     - `security.protocol: SSL` 使用TLS进行加密时（使用或不使用TLS身份验证）。
     - `security.protocol: SASL_SSL` 通过TLS使用SCRAM-SHA身份验证时。
   - `ssl.truststore.location` 导入证书的信任库位置。
   - `ssl.truststore.password` 和用于访问信任库的密码。如果信任库不需要此属性，则可以省略。
   - `ssl.truststore.type=PKCS12` 识别信任库类型。

使用PEM格式（.crt）

1. 从生成的密钥中提取群集CA证书。 `<cluster-name>-cluster-ca-cert`

   ```
   kubectl get secret <cluster-name>-cluster-ca-cert -o jsonpath='{.data.ca\.crt}' | base64 -d > ca.crt
   ```

2. 将证书与使用X.509格式证书的客户端一起使用。

9.8。Kafka侦听器证书

您可以为以下类型的侦听器提供自己的服务器证书和私钥：

- TLS侦听器，用于集群间通信
- 外部侦听器（`route`，，，和类型），它具有TLS加密启用，Kafka客户和Kafka经纪人之间的沟通 `loadbalancer` `ingress` `nodeport`

这些用户提供的证书称为*Kafka侦听器证书*。

通过为外部侦听器提供Kafka侦听器证书，您可以利用现有的安全基础结构，例如组织的私有CA或公共CA。Kafka客户端将使用Kafka侦听器证书而非群集CA或客户端CA签名的证书连接到Kafka代理。

您必须在需要时手动续订Kafka侦听器证书。

9.8.1。提供您自己的Kafka侦听器证书

此过程说明如何配置侦听器以使用自己的私有密钥和服务器证书，称为Kafka侦听器证书。

您的客户端应用程序应使用CA公钥作为受信任的证书，以验证Kafka代理的身份。
先决条件

- Kubernetes集群
- 群集 Operator 正在运行。
- 对于每个侦听器，由外部CA签名的兼容服务器证书。
  - 提供PEM格式的X.509证书。
  - 为每个侦听器指定正确的 Topic 备用名称（SAN）。有关更多信息，请参阅服务器证书中针对Kafka侦听器的替代 Topic 。
  - 您可以在证书文件中提供包含整个CA链的证书。

程序

1. 创建一个包含您的私钥和服务器证书的文件： `Secret`

   ```
   kubectl create secret generic my-secret --from-file=my-listener-key.key --from-file=my-listener-certificate.
   crt
   ```

2. 编辑群集的资源。在属性中将侦听器配置为使用您的，证书文件和私钥文件。启用TLS加密的外部侦听器的示例配置 `Kafka` `Secret` `config uration.brokerCertChainAndKey` `loadbalancer`

   ```
   # ...
   listeners:
     plain: {}
     external:
       type: loadbalancer
       configuration:
         brokerCertChainAndKey:
           secretName: my-secret
           certificate: my-listener-certificate.crt
   ```

```
        key: my-listener-key.key
    tls: true
    authentication:
      type: tls
# ...
```

TLS侦听器的示例配置

```
# ...
listeners:
  plain: {}
  tls:
    configuration:
      brokerCertChainAndKey:
        secretName: my-secret
        certificate: my-listener-certificate.pem
        key: my-listener-key.key
    authentication:
      type: tls
# ...
```

3. 应用新配置以创建或更新资源：

   ```
   kubectl apply -f kafka.yaml
   ```

   集群 Operator 开始滚动更新Kafka集群，该更新将更新侦听器的配置。

注意	如果您在TLS或外部侦听器已使用的中更新Kafka侦听器证书，则也将开始滚动更新。 `Secret`

额外资源

- 服务器证书中Kafka侦听器的替代 Topic
- `KafkaListeners` 模式参考
- Kafka侦听器证书

9.8.2。服务器证书中Kafka侦听器的替代 Topic

为了对自己的Kafka侦听器证书使用TLS主机名验证，必须为每个侦听器使用正确的使用者备用名称（SAN）。证书SAN必须为以下内容指定主机名：

- 集群中的所有Kafka经纪人
- Kafka集群引导服务

如果您的CA支持通配符证书，则可以使用它们。

TLS侦听器SAN示例

使用以下示例可以帮助您在TLS侦听器的证书中指定SAN的主机名。
通配符示例

```
//Kafka brokers
*.<cluster-name>-kafka-brokers
*.<cluster-name>-kafka-brokers.<namespace>.svc

// Bootstrap service
<cluster-name>-kafka-bootstrap
<cluster-name>-kafka-bootstrap.<namespace>.svc
```

非通配符示例

```
// Kafka brokers
<cluster-name>-kafka-0.<cluster-name>-kafka-brokers
<cluster-name>-kafka-0.<cluster-name>-kafka-brokers.<namespace>.svc
<cluster-name>-kafka-1.<cluster-name>-kafka-brokers
<cluster-name>-kafka-1.<cluster-name>-kafka-brokers.<namespace>.svc
# ...

// Bootstrap service
<cluster-name>-kafka-bootstrap
<cluster-name>-kafka-bootstrap.<namespace>.svc
```

外部侦听器SAN示例

对于启用了TLS加密的外部侦听器，您需要在证书中指定的主机名取决于外部侦听器。 `type`

表14.每种类型的外部侦听器的SAN

外部监听器类型	在SAN中，指定...

Route	所有Kafka经纪人的地址和引导地址。 Routes Route 您可以使用匹配的通配符名称。
loadbalancer	所有Kafka经纪人的地址和引导地址。 loadbalancers loadbalancer 您可以使用匹配的通配符名称。
NodePort	Kafka代理程序pod可能调度到的所有Kubernetes工作者节点的地址。 您可以使用匹配的通配符名称。

额外资源

- 提供您自己的Kafka侦听器证书

10. 管理Strimzi

本章介绍维护Strimzi部署的任务。

10.1。使用自定义资源

您可以使用命令来检索信息并在Strimzi自定义资源上执行其他操作。 kubectl

使用与自定义资源的子资源可以让你获得有关资源的信息。 kubectl status

10.1.1。在自定义资源上执行操作 kubectl

使用命令，如，，，或，对资源类型进行操作。例如，检索所有Kafka Topic 的列表，并检索所有已部署的Kafka群集。 kubectl get describe edit delete kubectl get kafkatopics kubectl get kafkas

引用资源类型时，可以同时使用单数和复数名称：获得与相同的结果。 kubectl get kafkas kubectl get kafka

您也可以使用资源的*简称*。学习短名称可以节省您管理Strimzi的时间。的简称是，因此您也可以运行以列出所有Kafka群集。 Kafka k kubectl get k

```
kubectl get k

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

表15.每个Strimzi资源的简称

Strimzi资源	长名	简称
Kafka	Kafka	
Kafka Topic	kafkatopic	千吨
Kafka用户	kafkauser	区
Kafka Connect	kafkaconnect	c
Kafka Connect S2I	kafkaconnects2i	kcs2i
Kafka连接器	kafkaconnector	kctr
Kafka镜子制造商	kafkamirrormaker	公里
Kafka镜子制造商2	kafkamirrormaker2	平方千米
Kafka桥	Kafka布里奇	千位
Kafka再平衡	kafkarebalance	r

资源类别

定制资源的类别也可以在命令中使用。 kubectl

所有Strimzi自定义资源都属于该类别，因此您可以使用一个命令来获取所有Strimzi资源。 strimzi strimzi

例如，运行列出了给定名称空间中的所有Strimzi自定义资源。 kubectl get strimzi

```
kubectl 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
```

该命令返回所有资源类型和资源名称。该选项以*类型/名称*格式获取输出 `kubectl get strimzi -o name` `-o name`

```
kubectl get strimzi -o name
```

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

您可以将此命令与其他命令结合使用。例如，您可以将其传递给命令以在单个命令中删除所有资源。 `strimzi` `kubectl delete`

```
kubectl delete $(kubectl 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
```

例如，在测试新的Strimzi功能时，一次操作中删除所有资源可能会很有用。

查询子资源状态

您还可以将其他值传递给该选项。例如，通过使用获得YAML格式的输出。Usng 将其作为JSON返回。 `-o` `-o yaml` `-o json`

您可以在中看到所有选项。 `kubectl get --help`

JSONPath支持是最有用的选项之一，它使您可以传递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表达式从Kafka自定义资源的状态获取引导程序地址，并在您的Kafka客户端中使用它。 `{.status.listeners[?(@.type=="tls")].bootstrapServers}`

在此，该命令查找侦听器的值。 `bootstrapServers` `tls`

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

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

通过将类型条件更改为或，您还可以获得其他Kafka侦听器的地址。 `@.type=="external"` `@.type=="plain"`

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

```
192.168.1.247:9094
```

您可以用来从任何自定义资源中提取任何其他属性或一组属性。 `jsonpath`

10.1.2。Strimzi自定义资源状态信息

几个资源具有一个属性，如下表所述。 `status`

表16.定制资源状态属性

Strimzi资源	模式参考	在...上发布状态信息
`Kafka`	`KafkaStatus` 模式参考	Kafka集群。
`KafkaConnect`	`KafkaConnectStatus` 模式参考	Kafka Connect集群（如果已部署）。
`KafkaConnectS2I`	`KafkaConnectS2IStatus` 模式参考	带有Source-to-Image支持的Kafka Connect集群（如果已部署）。
`KafkaConnector`	`KafkaConnectorStatus` 模式参考	`KafkaConnector` 资源（如果已部署）。
`KafkaMirrorMaker`	`KafkaMirrorMakerStatus` 模式参考	Kafka MirrorMaker工具（如果已部署）。
`KafkaTopic`	`KafkaTopicStatus` 模式参考	Kafka集群中的Kafka Topic 。
`KafkaUser`	`KafkaUserStatus` 模式参考	您的Kafka集群中的Kafka用户。
`KafkaBridge`	`KafkaBridgeStatus` 模式参考	Strimzi Kafka桥（如果已部署）。

资源的属性提供有关资源的信息： `status`

- *目前的状态*，在财产 `status.conditions`
- *最后观察到的一代*，在财产 `status.observedGeneration`

该属性还提供特定于资源的信息。例如： `status`

- `KafkaConnectStatus` 提供Kafka Connect连接器的REST API端点。
- `KafkaUserStatus` 提供Kafka用户的用户名及其凭据存储在其中。 `Secret`
- `KafkaBridgeStatus` 提供外部客户端应用程序可以访问Bridge服务的HTTP地址。

资源的*当前状态*可用于跟踪与资源达到其*所需状态*（由资源定义）相关的进度。状态条件提供了资源状态更改的时间和原因，以及阻止或延迟 Operator 实现资源所需状态的事件详细信息。 `spec`

*观察到的最后一代*是集群 Operator 最后协调的资源的一代。如果的值与的值不同，则 Operator 尚未处理对资源的最新更新。如果这些值相同，则状态信息将反映资源的最新更改。 `observedGeneration` `metadata.generation`

Strimzi创建并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。例如，使用来对自定义资源执行更新时，其不可编辑。此外，更改此设置不会影响Kafka群集的配置。 `kubectl edit status` `status`

在这里，我们看到为Kafka自定义资源指定的属性。Kafka自定义资源的状态 `status`

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
spec:
  # ...
status:
  conditions: (1)
  - lastTransitionTime: 2019-07-23T23:46:57+0000
    status: "True"
    type: Ready (2)
  observedGeneration: 4 (3)
  listeners: (4)
  - addresses:
    - host: my-cluster-kafka-bootstrap.myproject.svc
      port: 9092
    type: plain
  - addresses:
    - host: my-cluster-kafka-bootstrap.myproject.svc
      port: 9093
    certificates:
    - |
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
    type: tls
  - addresses:
    - host: 172.29.49.180
      port: 9094
    certificates:
    - |
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
    type: external
    # ...
```

1. 状态描述了与状态相关的标准，这些标准不能从现有资源信息中推导出，或者特定于资源实例。 `conditions`
2. 该条件表明集群 Operator 当前是否认为Kafka集群能够处理流量。 `Ready`
3. 该指示的产生是最后由群集 Operator 不甘心自定义资源。 `observedGeneration` `Kafka`
4. 在按类型描述当前的Kafka引导地址。 `listeners`

重要	当前不支持类型为外部侦听器的自定义资源状态中的地址。 `nodeport`

注意	状态中列出的Kafka引导程序地址并不表示这些端点或Kafka群集处于就绪状态。

访问状态信息

您可以从命令行访问资源的状态信息。有关更多信息，请参见查找自定义资源的状态。

10.1.3。查找自定义资源的状态

此过程描述了如何查找自定义资源的状态。
先决条件

- Kubernetes集群
- 群集 Operator 正在运行。

程序

- 指定自定义资源，并使用选项应用标准的JSONPath表达式来选择属性： `-o jsonpath` `status`

  ```
  kubectl get kafka <kafka_resource_name> -o jsonpath='{.status}'
  ```

该表达式返回指定自定义资源的所有状态信息。您可以使用点符号（例如或）来微调您希望看到的状态信息。 `status.listeners` `status.observedGeneration`

额外资源

- Strimzi自定义资源状态信息
- 有关使用JSONPath的更多信息，请参见JSONPath支持。

10.2。使用标签和注释发现服务

服务发现使与Strimzi在同一Kubernetes集群中运行的客户端应用程序更容易与Kafka集群进行交互。

一个*服务发现*标签和注释用于访问Kafka集群服务产生：

- 内部Kafka引导程序服务
- HTTP桥服务

标签有助于使服务可发现，并且注释提供了客户端应用程序可以用来建立连接的连接详细信息。

该服务发现标签，被设置为为资源。服务发现批注具有相同的密钥，以JSON格式提供每个服务的连接详细信息。 `strimzi.io/discovery` `true` `Service`

内部Kafka引导服务示例

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    strimzi.io/discovery: |-
      [ {
        "port" : 9092,
        "tls" : false,
        "protocol" : "kafka",
        "auth" : "scram-sha-512"
      }, {
        "port" : 9093,
        "tls" : true,
        "protocol" : "kafka",
        "auth" : "tls"
      } ]
  labels:
    strimzi.io/cluster: my-cluster
    strimzi.io/discovery: "true"
    strimzi.io/kind: Kafka
    strimzi.io/name: my-cluster-kafka-bootstrap
  name: my-cluster-kafka-bootstrap
spec:
  #...
```

HTTP网桥服务示例

```
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
```

10.2.1。返回服务的连接详细信息

从命令行或相应的API调用中获取服务时，可以通过指定发现标签来找到服务。

```
kubectl get service -l strimzi.io/discovery=true
```

检索服务发现标签时，将返回连接详细信息。

10.3。从永久卷中恢复集群

您可以从持久卷（PV）中恢复Kafka群集（如果它们仍然存在）。

您可能想要这样做，例如，在之后：

- 命名空间被无意删除
- 整个Kubernetes集群丢失了，但PV保留在基础架构中

10.3.1。从名称空间删除中恢复

由于永久卷和名称空间之间的关系，可以从名称空间删除中恢复。一（PV）是住一个命名空间之外的存储资源。PV使用（PVC）安装在Kafka吊舱中，该PVC位于命名空间中。 `PersistentVolume` `PersistentVolumeClaim`

PV的回收策略告诉集群删除命名空间时如何操作。如果将回收策略设置为：

- *删除* （默认），当删除命名空间中的PVC时，PV被删除
- *保留*，删除命名空间时不会删除PV

为了确保在无意中删除了命名空间的情况下可以从PV中恢复，必须使用以下属性在PV规范中将策略从“ *删除*”重置为“ *保留* ” ： `persistentVolumeReclaimPolicy`

```
apiVersion: v1
kind: PersistentVolume
# ...
spec:
  # ...
  persistentVolumeReclaimPolicy: Retain
```

或者，PV可以继承关联存储类的回收策略。存储类用于动态卷分配。

通过配置存储类的属性，可以使用适当的回收策略来创建使用该存储类的PV。使用属性为PV配置了存储类。 `reclaimPolicy` `storageClassName`

```
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*保留*作为回收策略，但是要删除整个集群，则需要手动删除PV。否则它们将不会被删除，并可能导致不必要的资源支出。

10.3.2。从Kubernetes集群丢失中恢复

当群集丢失时，如果磁盘/卷中的数据保留在基础架构中，则可以使用它们来恢复群集。恢复过程与删除命名空间的过程相同，假定PV可以恢复并且是手动创建的。

10.3.3。从永久卷中恢复已删除的群集

此过程描述了如何从持久卷（PV）中恢复已删除的群集。

在这种情况下， Topic operator 会确定 Topic 存在于Kafka中，但是资源不存在。 `KafkaTopic`

在执行重新创建集群的步骤时，有两个选择：

1. 当您可以恢复所有资源时，请使用*选项1*。 `KafkaTopic`

   该集群开始之前，使相应的 Topic 不是由 Topic 操作删除资源，因此必须恢复。 `KafkaTopic`
2. 当您无法恢复所有资源时，请使用*选项2*。 `KafkaTopic`

   这次，您在没有 Topic operator 的情况下部署群集，请在ZooKeeper中删除 Topic operator 数据，然后重新部署它，以便 Topic operator 可以从相应 Topic 重新创建资源。 `KafkaTopic`

注意	如果未部署 Topic operator ，则仅需要恢复（PVC）资源。 `PersistentVolumeClaim`

在你开始之前

在此过程中，必须将PV安装到正确的PVC中，以避免数据损坏。为PVC指定了A ，它必须与PV的名称匹配。 `volumeName`

有关更多信息，请参见：

- 持久卷声明命名
- JBOD和持久量声明

注意	该过程不包括资源恢复，必须手动重新创建。如果需要保留密码和证书，则在创建资源之前必须重新创建秘密。 `KafkaUser` `KafkaUser`

程序

1. 检查有关集群中PV的信息：

   ```
   kubectl get pv
   ```

   带有数据的PV将显示信息。

   输出示例显示此过程重要的列：

   ```
   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已*保留*。
   - *CLAIM* 显示了到原始PVC的链接。
2. 重新创建原始名称空间：

   ```
   kubectl create namespace myproject
   ```

3. 重新创建原始的PVC资源规范，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 的属性。 `claimRef`

   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. 部署集群 Operator 。

```
kubectl apply -f install/cluster-operator -n my-project
```

6. 重新创建集群。

请执行以下步骤，具体取决于您是否具有重新创建集群所需的所有资源。 KafkaTopic

选项1：如果你有**所有**的已存在你失去了你的群集之前，包括内部的 Topic ，如从提交的偏移资源： KafkaTopic __consumer_offsets

a. 重新创建所有资源。 KafkaTopic

在部署群集之前，必须重新创建资源，否则 Topic Operator 将删除 Topic 。

b. 部署Kafka集群。

例如：

```
kubectl apply -f kafka.yaml
```

选项2：如果您没有丢失群集之前存在的所有资源： KafkaTopic

a. 与第一个选项一样，通过在部署之前从Kafka资源中删除属性来部署Kafka集群，但不使用 Topic operator 。 topicOperator

如果在部署中包括 Topic operator ，则 Topic operator 将删除所有 Topic 。

b. 对Kafka代理pod之一运行命令以打开ZooKeeper shell脚本。 exec

例如，其中*my-cluster-kafka-0*是代理窗格的名称：

```
kubectl exec my-cluster-kafka-0 bin/zookeeper-shell.sh localhost:2181
```

c. 删除整个路径以删除 Topic operator 存储： /strimzi

```
deleteall /strimzi
```

d. 通过重新部署带有属性的Kafka群集来重新创建资源，以启用 Topic operator 。 topicOperator KafkaTopic

例如：

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  #...
  entityOperator:
    topicOperator: {} (1)
    #...
```

a. 在这里，我们显示默认配置，没有其他属性。您可以使用架构参考中描述的属性来指定所需的配置。 EntityTopicOperatorSpec

7. 通过列出资源来验证恢复： KafkaTopic

```
kubectl get KafkaTopic
```

10.4。卸载Strimzi

此过程描述了如何卸载Strimzi并删除与部署相关的资源。
先决条件

为了执行此过程，请标识专门为部署创建并从Strimzi资源引用的资源。

这些资源包括：

- 机密（自定义CA和证书，Kafka Connect机密以及其他Kafka机密）
- 记录（类型） `ConfigMaps` `external`

这些都是引用的资源，，，，或配置。程序 `Kafka` `KafkaConnect` `KafkaConnectS2I` `KafkaMirrorMaker` `KafkaBridge`

1. 删除Cluster Operator ，related 和资源： `Deployment` `CustomResourceDefinitions` `RBAC`

   ```
   kubectl-f install / cluster-operator
   ```

警告	删除结果相应的自定义资源（的垃圾收集，，，，或）和资源依赖于他们（部署，StatefulSets，以及其他相关资源）。 `CustomResourceDefinitions` `Kafka` `KafkaConnect` `KafkaConnectS2I` `KafkaMirrorMaker` `KafkaBridge`

2. 删除前提条件中标识的资源。