Red Hat Streams for Apache Kafka 2.7

使用 Apache Kafka 代理的流

使用专用功能增强 Kafka

Last Updated: 2024-05-29

Red Hat Streams for Apache Kafka 2.7 使用 Apache Kafka 代理的流

使用专用功能增强 Kafka

法律通告

摘要

AMQ Streams 代理是一个 Apache Kafka 协议感知代理，旨在通过其过滤机制来增强基于 Kafka 的系统。在这个预览中，AMQ Streams 代理包含一个 Record Encryption 过滤器，它为存储在 Kafka 集群中的数据提供加密。

目录

前言

对红帽文档提供反馈

我们感谢您对我们文档的反馈。

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

前提条件

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

流程

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

感谢您花时间来提供反馈。

技术预览

Apache Kafka 代理的流是一个技术预览。

技术预览功能不被红帽产品服务级别协议(SLA)支持，且可能无法完成。因此，红帽不推荐在生产环境中实施任何技术预览功能。此技术预览功能为您提供对即将推出的产品创新的早期访问，允许您在开发过程中测试并提供反馈。如需有关支持范围的更多信息，请参阅 技术预览功能支持范围。

第 1 章 APACHE KAFKA 代理流概述

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

作为中间的，Apache Kafka Proxy 代理的流会促进 Kafka 集群及其客户端之间的通信。它负责接收、过滤和转发信息。

1.1. 记录加密过滤器

Apache Kafka Proxy 的 Record Encryption 过滤器的 Streams 增强了 Kafka 信息的安全性。过滤器使用行业标准的加密技术将加密应用到 Kafka 信息，确保 Kafka 集群中存储的数据的机密性。Apache Kafka 代理的流集中了主题级加密，以确保在 Kafka 集群间简化加密。

过滤器使用 envelope 加密，以使用对称加密密钥加密记录。

信封加密

envelope 加密是一种行业标准技术，可用于以高效的方式加密大量数据。数据使用数据加密密钥(DEK)进行加密。DEK 使用密钥加密密钥(KEK)进行加密。KEK 安全地存储在密钥管理系统(KMS)中。

对称加密密钥

AES (GCM) 256 位加密对称加密密钥用于加密和解密记录数据。

此过程如下：

1. 过滤器截获从生成应用程序生成请求并加密记录。
2. 生成请求转发到代理。
3. 过滤器截获从消耗应用程序获取响应并解密记录。
4. 获取响应转发到消耗应用程序。

过滤器仅加密记录值。记录键、标头和时间戳没有加密。

整个过程对于 Kafka 客户端和 Kafka 代理的视角是透明的。不明显是加密的记录，也没有对加密密钥的任何访问权限，或者对加密进程有任何影响来保护记录。

过滤器与密钥管理服务(KMS)集成，它负责安全存储关键资料。目前，如果进一步支持的 KMS 集成，过滤器会与 HashiCorp Vault 集成作为其 KMS。

1.1.1. 过滤器如何加密记录

过滤器加密来自生成请求的记录，如下所示：

1. 过滤器选择要应用的 KEK。
2. 请求 KMS 为 KEK 生成 DEK。
3. 使用加密的 DEK （使用 KEK 加密的DEK）加密记录。
4. 将原始记录替换为密码记录（加密记录、加密 DEK 和元数据）。

过滤器使用 DEK 重复利用策略。加密记录使用相同的 DEK 发送到同一主题，直到达到超时或加密限制为止。

1.1.2. 过滤器如何解密记录

过滤器从获取响应中解密记录，如下所示：

1. 过滤器从 Kafka 代理接收密码记录。
2. 反向构建密码记录的进程。
3. 使用 KMS 解密 DEK。
4. 使用解密的 DEK 来解密加密的记录。
5. 使用解密的记录替换密码记录。

过滤器使用 LRU （最近使用的）策略来解密记录。解密的 DEK 保存在内存中，以减少与 KMS 的交互。

1.1.3. 过滤器如何使用 KMS

为了支持过滤器，KMS 提供以下内容：

- 用于存储密钥加密密钥的安全存储库(KEK)
- 用于生成和解密数据加密密钥的服务(DEK)

KEK 保留在 KMS 中。KMS 为给定 KEK 生成 DEK （安全生成的随机数据），然后返回 DEK 和加密的 DEK。加密的 DEK 具有相同的数据，但使用 KEK 加密。KMS 不存储 DEK；它们作为密码记录的一部分存储在代理中。

警告

KMS 必须在运行时可用。如果 KMS 不可用，则通过过滤器进行生产和消耗将变得不可能，直到 KMS 服务被恢复为止。建议您在高可用性(HA)配置中使用 KMS。

1.1.4. 记录的哪个部分被加密？

record encryption 过滤器仅加密记录的值，保留记录键、标头和时间戳不动。null 记录值（可能代表压缩主题中的删除）被传送到代理未加密。此方法可确保压缩的主题可以正常工作。

1.1.5. 未加密的主题

您可以配置系统，以便一些主题加密，其他主题没有加密。这意味着带有机密信息的主题是加密的，带有非敏感信息的 Kafka 主题可以保留未加密的。

第 2 章 为 RECORD ENCRYPTION 过滤器准备 HASHICORP VAULT

要将 Vault 与 OpenShift 集群中的 Record Encryption 过滤器一起使用，请为您的 Vault 实例使用以下设置：

- 启用 Transit Engine，因为 Record Encryption 过滤器依赖于其 API。
- 为过滤器创建一个 Vault 策略，其具有生成和解密用于信封加密的数据加密密钥(DEK)的权限。
- 获取包含过滤器策略的 Vault 令牌。

代理的部署配置使用 Vault Transit Engine 服务的 URL。

Vault 可以部署为现有实例、云实例或 OpenShift。通过对代理的可访问性，它可以与 Apache Kafka 代理的 Streams 共存，也可以远程部署。

有关在 OpenShift 上安装 Vault 并设置访问权限的详情，请参考 HashiCorp Vault 产品文档。

此流程概述了准备 Vault 的两个选项：

- 使用带有 Apache Kafka 代理的 Streams 提供的临时部署配置的 Helm 将 Vault 部署到 OpenShift 集群。
- 更新现有的 Vault 实例。

准备 Vault 实例时，您必须为 Record Encryption 过滤器创建 Vault 策略和令牌。

警告

示例部署配置不适用于生产环境。

Apache Kafka 的 Streams 在 **examples/proxy/record-encryption/vault** 文件夹中包括示例安装工件，其中包含与代理和记录加密过滤器兼容的预先配置 Vault 部署文件。

- **AMQstreams_proxy_encryption_filter_policy.hcl** 为 Record Encryption 过滤器定义一个 Vault 策略
- **helm-dev-values.yaml** 指定 Vault 的 Helm 部署配置

这些安装文件为尝试代理提供了一个快速设置。

先决条件

- 安装需要具有 **cluster-admin** 角色的 OpenShift 用户，如 **system:admin**。
- **oc** 命令行工具已安装并配置为连接到具有 admin 访问权限的 OpenShift 集群。
- **helm** 命令行工具已安装并配置为连接到具有 admin 访问权限的 OpenShift 集群。
- 名为 **proxy** 的 OpenShift 项目命名空间，这是默认安装代理的同一命名空间。

有关此流程中使用的 **oc** 和 **helm** 命令行选项的信息，请查看 **--help**。

使用 Helm 部署配置示例部署 Vault

1. 下载并提取 Apache Kafka 代理安装工件的 Streams。
   代理可从 Apache Kafka 软件下载页面 的 Streams 获取 。

   文件包含部署 Vault 所需的部署配置。

2. 创建根令牌并记下它：

   ```
   cat /dev/urandom | LC_ALL=C tr -dc 'a-zA-Z0-9' | fold -w 32 | head -n 1 > vault.root.token
   export VAULT_TOKEN=$(cat vault.root.token)
   ```

3. 使用 Helm 安装 Vault：

   ```
   helm repo add hashicorp https://helm.releases.hashicorp.com
   helm install vault hashicorp/vault \
     --create-namespace --namespace=vault \
     --version <helm_version> \
     --values vault/helm-dev-values.yaml \
     --set server.dev.devRootToken=${VAULT_TOKEN} \
     --wait
   ```

   root 令牌用于 Vault 实例。

4. 检查部署的状态：

   ```
   oc get pods -n vault
   ```

   输出显示部署名称和就绪

   ```
   NAME                    READY  STATUS   RESTARTS
   vault-0                 1/1    Running  0
   ```

   标识创建的 pod 的 pod ID。

   使用默认部署，您可以安装单个代理 pod。

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

5. 创建 Vault 地址(**VAULT_ADDR**)环境变量以指向新的 Vault 实例：

   ```
   export VAULT_ADDR=$(oc get route -n vault vault --template='https://{{.spec.host}}')
   ```

6. 以管理员身份登录到 Vault，并启用 Vault Transit secret 引擎：

   ```
   vault secrets enable transit
   ```

   如果已经启用了 secret 引擎，请忽略错误。

7. 创建指向 Vault Transit 地址的环境变量：

   ```
   export VAULT_TRANSIT_URL=${VAULT_ADDR}/v1/transit
   ```

地址在代理部署配置中使用。

8. 创建 Vault 策略和令牌。

配置您自己的 Vault 实例

如果您已经安装了 Kafka 实例，您可以更新它以用于 Apache Kafka 代理的 Streams。

1. 创建一个 Vault 地址环境变量(**VAULT_ADDR** 和 **VAULT_NAMESPACE**，如果使用 Enterprise)以指向 Vault 实例：

```
export VAULT_ADDR=https://<vault server>:8200
export VAULT_NAMESPACE=<namespaces>
```

2. 以管理员身份登录到 Vault，并启用 Vault Transit secret 引擎：

```
vault secrets enable transit
```

如果已经启用了 secret 引擎，请忽略错误。

3. 创建指向 Vault Transit 地址的环境变量：

```
export VAULT_TRANSIT_URL=${VAULT_ADDR}/v1/${VAULT_NAMESPACE}/transit
```

地址在代理部署配置中使用。

4. 更新代理部署配置，以引用您的 Vault 实例：

```
sed -i "s/\(vaultTransitEngineUrl:\).*$/\1 ${VAULT_TRANSIT_URL}/" */proxy/proxy-config.yaml
```

5. 创建 Vault 策略和令牌。

创建 Vault 策略和令牌

设置 Vault 实例后，为 Record Encryption 过滤器创建一个 Vault 策略和令牌。

1. 创建 Vault 策略：

```
vault policy write amqstreams_proxy_encryption_filter_policy vault/amqstreams_proxy_encryption_filter_policy.hcl
```

使用由 Apache Kafka Proxy 的 Streams 提供的 HashiCorp 策略定义文件(**.hcl**)将策略写入 Vault。该策略名为 **amqstreams_proxy_encryption_filter_policy**。

2. 创建 Vault 令牌：

```
vault token create \
  -display-name "amqstreams-proxy encryption filter" \
  -policy=amqstreams_proxy_encryption_filter_policy \
  -no-default-policy \
  -orphan \
  -field=token > vault.encryption.token
```

该命令使用指定策略创建令牌，且没有关联的父令牌或默认策略。

3. 创建包含令牌的 secret：

```
oc create secret generic proxy-encryption-vault-token \
  -n proxy \
  --from-file=encryption-vault-token.txt=vault.encryption.token \
  --dry-run=client \
  -o yaml > base/proxy/proxy-encryption-vault-token-secret.yaml
```

命令将 Vault 令牌存储在机密中，并将 secret 创建为 **代理** 命名空间中的 YAML 文件。

当使用 Record Encryption 过滤器部署流 for Apache Kafka Proxy 时，**proxy-encryption-vault-token-secret.yaml** secret 会被应用到 OpenShift 集群。

提示

定期轮转密钥，以最大程度降低被入侵的密钥的影响。当使用密钥管理系统(KMS)时，如 HashiCorp Vault，轮转保存在 KMS 中的密钥加密密钥(KEK)。Apache Kafka 代理的流会自动管理 DEK 轮转。代理可能需要执行 occasional 重启才能获取新密钥。另外，加密的消息应包含密钥版本元数据来指示密钥轮转。

第 3 章 使用 RECORD ENCRYPTION 过滤器部署 APACHE KAFKA 代理的流

Apache Kafka 代理的流旨在与由 Streams for Apache Kafka 管理的 Kafka 集群无缝集成。另外，它还提供与所有类型的 Kafka 实例的兼容性，无论它们的发布或协议版本是什么。无论您的部署涉及公共云或私有云，还是要设置本地开发环境，本指南中的说明适用于所有情况。

在此过程中，Apache Kafka Proxy 的 Streams 使用 Record Encryption 过滤器部署，以用于 OpenShift 中由 Apache Kafka 管理的 Kafka 实例。

Apache Kafka 的 Streams 提供了示例安装工件，其中包含 Apache Kafka Proxy 的 Streams 所需的配置，以连接到 **examples/proxy/record-encryption** 文件夹中的 Kafka 集群。

使用示例配置文件，使用以下监听程序类型部署和公开代理：

- 使用每个代理的 **ClusterIP** 服务的 **cluster-ip** 类型监听程序在 OpenShift 集群中公开代理
- 使用 per-broker **loadbalancer 服务在** OpenShift 集群外公开代理的 LoadBalancer 类型监听程序

对于每个选项，提供了以下文件：

- **kustomization.yaml** 指定用于部署代理的 Kubernetes 自定义
- **proxy-config.yaml** 指定代理的 **ConfigMap** 资源配置
- **proxy-service.yaml** 指定代理服务的 **Service** 资源配置

ConfigMap 资源提供设置虚拟集群和过滤器的配置。虚拟主机代表您要与代理一起使用的 Kafka 集群。

先决条件

- 运行受支持的版本的 OpenShift 集群。
- 由 Streams for Apache Kafka 管理的 Kafka 集群在 OpenShift 集群中运行的。
- 本地安装的 Kafka 二进制文件，以验证通过 loadbalancer 进行外部访问的代理设置。Kafka 二进制文件包含在 RHEL for Apache Kafka 软件下载页面 的 Streams for Apache Kafka 的安装工件中。
- 包含用于创建虚拟设备和过滤器的配置的配置映射。
- **oc** 命令行工具已安装并配置为连接到具有 admin 访问权限的 OpenShift 集群。
- **helm** 命令行工具已安装并配置为连接到具有 admin 访问权限的 OpenShift 集群。
- HashiCorp Vault 为代理设置，可从 Streams for Apache Kafka Proxy 访问。
  确保为 Record Encryption 过滤器设置了 Vault 实例。

有关此流程中使用的 **oc** 和 **helm** 命令行选项的信息，请查看 **--help**。

除了为 Apache Kafka 代理安装流的文件外，Apache Kafka Proxy 的 Streams 还提供预配置的文件来安装 Kafka 集群。安装文件提供了设置和尝试代理的最快速方法，但您可以使用自己的由 Streams for Apache Kafka 和 Vault 管理的 Kafka 集群部署。

在此过程中，我们连接到名为 **my-cluster** 的 Kafka 集群，部署到 **kafka** 命名空间。要将代理部署到与

Apache Kafka 中由 Streams 管理的集群相同的命名空间，请更改 **kustomization.yaml** 文件中的 **namespace** 设置。代理默认部署到 **代理** 命名空间中。如果保留此设置，则必须在集群范围内安装 Apache Kafka Operator 的 Streams。

流程

1. 下载并提取 Apache Kafka 代理安装工件的 Streams。
   代理可从 Apache Kafka 软件下载页面 的 Streams 获取 。

   文件包含通过 **cluster-ip** 或 **loadbalancer** 类型监听程序连接所需的部署配置。

2. 在 Kafka 集群中创建主题：

```
oc run -n <my_proxy_namespace> -ti proxy-producer \
  --image=registry.redhat.io/amq-streams/kafka-37-rhel8:2.7.0 \
  --rm=true \
  --restart=Never \
  -- bin/kafka-topics.sh \
  --bootstrap-server proxy-service:9092 \
  --create -topic my-topic
```

   在本例中，我们通过交互式 pod 容器创建一个名为 **my-topic** 的主题。

3. 在 Vault 中为 **my-topic** 创建密钥：

```
vault write -f transit/keys/KEK_my-topic
```

4. 编辑为代理提供过滤器配置的 **ConfigMap**。
   Record Encryption **过滤器** 配置需要 HashiCorp Vault KMS 的凭证。

   记录加密过滤器配置示例

```
filters:
  - type: RecordEncryption
    config:
      kms: VaultKmsService (1)
      kmsConfig:
        vaultTransitEngineUrl: http://vault.vault.svc.cluster.local:8200/v1/transit (2)
        vaultToken:
          passwordFile: /opt/proxy/encryption/token.txt (3)
      selector: TemplateKekSelector (4)
      selectorConfig:
        template: "KEK_${topicName}" (5)
  # ...
```

   (1) 使用的 KMS （密钥管理系统）的类型。在本例中，HashiCorp Vault。

   (2) Vault Transit Engine 服务的 URL。

   (3) 包含访问 Vault 服务所需的令牌的文件。如果此位置发生变化，则代理部署配置中需要进行等同的更改。

   (4) 要使用的密钥加密密钥(KEK)选择器。**${topicName}** 是代理理解的字面值。

5 基于特定主题名称派生的 KEK 模板。

5. 使用 Record Encryption 过滤器和适当的监听程序为 OpenShift 集群部署 Apache Kafka 代理的流：

使用 cluster-ip 侦听器部署代理

```
cd /examples/proxy/record-encryption/
oc apply -k cluster-ip
```

使用 loadbalancer 侦听器部署代理

```
cd /examples/proxy/record-encryption/
oc apply -k loadbalancer
```

6. 如果您使用 **loadbalancer** 侦听器，请更新代理配置以使用所创建的 loadbalancer 服务地址。

   a. 获取代理服务的外部地址：

   ```
   LOAD_BALANCER_ADDRESS=$(oc get service -n <my_proxy_namespace> proxy-service --template='{{(index .status.loadBalancer.ingress 0).hostname}}')
   ```

   b. 更新代理服务配置中的 **brokerAddressPattern** 属性以使用代理地址：

   ```
   sed -i "s/\(brokerAddressPattern:\).*$/\1 ${LOAD_BALANCER_ADDRESS}/" load-balancer/proxy/proxy-config.yaml
   ```

   c. 将更改应用到代理配置并重启代理 pod。

   ```
   oc apply -k load-balancer && oc delete pod -n <my_proxy_namespace> --all
   ```

7. 检查部署的状态：

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

输出显示部署名称和就绪

```
NAME                  READY  STATUS   RESTARTS
my-cluster-kafka-0      1/1    Running  0
my-cluster-kafka-1      1/1    Running  0
my-cluster-kafka-2      1/1    Running  0
my-cluster-proxy-<pod_id> 1/1    Running  0
```

my-cluster-proxy 是代理的名称。

标识创建的 pod 的 pod ID。

使用默认部署，您可以安装单个代理 pod。

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

8. 通过代理生成消息，然后直接从 Kafka 集群使用和间接使用，验证加密是否已应用到指定的主题。

3.1. 在使用 CLUSTER-IP 类型监听程序时验证代理

在使用 **cluster-ip** 类型监听程序时，为 OpenShift 集群中的 Kafka producer 和消费者运行交互式 pod 容器，以验证代理是否正常工作。

1. 从代理生成消息：

   通过代理生成消息

   ```
   oc run -n <my_proxy_namespace> -ti proxy-producer \
     --image=registry.redhat.io/amq-streams/kafka-37-rhel8:2.7.0 \
     --rm=true \
     --restart=Never \
     -- bin/kafka-console-producer.sh \
     --bootstrap-server proxy-service:9092 \
     --topic my-topic
   ```

2. 直接从 Kafka 集群使用消息来显示它们已被加密：

   直接从 Kafka 集群使用消息

   ```
   oc run -n my-cluster -ti cluster-consumer \
     --image=registry.redhat.io/amq-streams/kafka-37-rhel8:2.7.0 \
     --rm=true \
     --restart=Never
     -- ./bin/kafka-console-consumer.sh  \
     --bootstrap-server my-cluster-kafka-bootstrap:9092 \
     --topic my-topic \
     --from-beginning \
     --timeout-ms 10000
   ```

3. 使用代理的消息来自动解密它们：

   直接从 Kafka 集群使用消息

   ```
   oc run -n <my_proxy_namespace> -ti proxy-consumer \
     --image=registry.redhat.io/amq-streams/kafka-37-rhel8:2.7.0 \
     --rm=true \
     --restart=Never \
     -- ./bin/kafka-console-consumer.sh \
     --bootstrap-server proxy-service:9092 \
     --topic my-topic --from-beginning --timeout-ms 10000
   ```

3.2. 在使用 LOADBALANCER 类型监听程序时验证代理

在通过本地代理运行 Kafka producer 和消费者时，验证代理是否正常工作。

1. 使用 loadbalancer 地址从代理生成消息：

通过代理生成消息

```
kafka-console-producer \
--bootstrap-server <load_balancer_address>:9092 \
--topic my-topic
```

2. 使用交互式 pod 容器直接从 Kafka 集群使用消息来显示它们已被加密：

直接从 Kafka 集群使用消息

```
oc run -n my-cluster -ti cluster-consumer \
  --image=registry.redhat.io/amq-streams/kafka-37-rhel8:2.7.0 \
  --rm=true \
  --restart=Never
  -- ./bin/kafka-console-consumer.sh  \
  --bootstrap-server my-cluster-kafka-bootstrap:9092 \
  --topic my-topic \
  --from-beginning \
  --timeout-ms 10000
```

3. 使用代理的消息来自动解密它们：

直接从 Kafka 集群使用消息

```
kafka-console-consumer \
--bootstrap-server <load_balancer_address>:9092 \
--topic my-topic \
--from-beginning \
--timeout-ms 10000
```

第 4 章 为 APACHE KAFKA 代理配置流

通过为 Apache Kafka Proxy 资源配置流来微调部署，以根据您的特定要求包含额外功能。

4.1. APACHE KAFKA PROXY 配置示例

Apache Kafka Proxy 配置的流在 ConfigMap 资源中定义。使用 ConfigMap 资源的 data 属性来配置以下内容：

- 代表 Kafka 集群的虚拟集群
- Kafka 集群中代理通信的网络地址
- 过滤器为 Kafka 部署引入了额外的功能

在本例中，显示了 Record Encryption 过滤器的配置。

Apache Kafka Proxy 配置示例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: proxy-config
data:
  config.yaml: |
    adminHttp: 1
      endpoints:
        prometheus: {}
    virtualClusters: 2
      my-cluster-proxy: 3
        targetCluster:
          bootstrap_servers: my-cluster-kafka-bootstrap.kafka.svc.cluster.local:9093 4
          tls: 5
            trust:
              storeFile: /opt/proxy/trust/ca.p12
              storePassword:
                passwordFile: /opt/proxy/trust/ca.password
        clusterNetworkAddressConfigProvider: 6
          type: SniRoutingClusterNetworkAddressConfigProvider
```

```
        Config:
          bootstrapAddress: mycluster-proxy.kafka:9092
          brokerAddressPattern: broker$(nodeId).mycluster-proxy.kafka
        logNetwork: false 7
        logFrames: false
        tls: 8
          key:
            storeFile: /opt/proxy/server/key-material/keystore.p12
            storePassword:
              passwordFile: /opt/proxy/server/keystore-password/storePassword
filters: 9
 - type: EnvelopeEncryption 10
   config: 11
     kms: VaultKmsService
     kmsConfig:
       vaultTransitEngineUrl: https://vault.vault.svc.cluster.local:8200/v1/transit
       vaultToken:
         passwordFile: /opt/proxy/server/token.txt
       tls: 12
         key:
           storeFile: /opt/cert/server.p12
           storePassword:
             passwordFile: /opt/cert/store.password
           keyPassword:
             passwordFile: /opt/cert/key.password
           storeType: PKCS12
     selector: TemplateKekSelector
     selectorConfig:
       template: "${topicName}"
```

1

为代理启用指标。

2

虚拟集群配置。

3

虚拟机的名称。

4

要代理的目标物理 Kafka 集群的 bootstrap 地址。

5

用于连接到目标集群的 TLS 配置。

6

用于控制如何将虚拟集群的集群网络地址配置供应商提供给网络。

7

默认禁用日志记录。通过将日志记录属性设置为 true 来启用与网络活动相关的日志记录(logNetwork)和消息(logFrames)。

8

用于保护与客户端的连接的 TLS 加密。

9

过滤配置。

10

过滤器类型，这是本例中的 Record Encryption 过滤器。

11

特定于过滤器类型的配置。

12

Record Encryption 过滤器需要连接到 Vault。如果需要，您还可以使用 Vault 指定 TLS 身份验证的凭证，其键名称存储在其中。

4.2. 配置虚拟集群

Kafka 集群由代理表示，作为虚拟集群。客户端连接到虚拟集群，而不是实际的集群。当部署了 Apache Kafka 代理的流时，它包括用于创建虚拟集群的配置。

虚拟接口只有一个目标集群，但很多虚拟集群可以针对同一集群为目标。每个目标集群都以目标集群上的一个监听程序为目标，因此 Kafka 端的多个监听程序由代理表示为多个虚拟集群。客户端使用 bootstrap_servers 地址连接到虚拟集群。现有集群有一个 bootstrap 地址，它映射到目标集群中的每个代理。当客户端连接到代理时，通过重写地址将通信代理到目标代理。返回客户端的响应会被重写，以反映虚拟集群的相应网络地址。

您可以保护来自客户端和目标集群的虚拟集群连接。

Apache Kafka 代理的流接受 PEM (Privacy Enhanced Mail)、PKCS TOTP (Public-Key Cryptography Standards)或 JKS (Java KeyStore)密钥存储格式的密钥和证书。

4.2.1. 保护来自客户端的连接

要保护到虚拟集群的客户端连接，请执行以下操作在虚拟集群中配置 TLS：

从证书颁发机构（证书授权机构）获取虚拟集群的 CA （证书授权机构）证书。在请求证书时，请确保它与虚拟磁盘的 bootstrap 和代理地址的名称匹配。这可能需要通配符证书和主题备用名称(SAN)。

使用 tls 属性在虚拟集群配置中指定 TLS 凭证。

PKCSkamelet 配置示例

```
virtualClusters:
  my-cluster-proxy:
    tls:
      key:
        storeFile: <path>/server.p12 1
        storePassword:
          passwordFile: <path>/store.password 2
        keyPassword:
          passwordFile: <path>/key.password 3
        storeType: PKCS12 4
      # ...
```

1 虚拟主机的公共 CA 证书的 PKCSRG 存储。

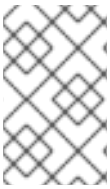
2 保护 PKCS TOTP 存储的密码。

3

（可选）密钥的密码。如果没有指定密码，则使用密钥存储的密码来解密密钥。

4

（可选） Keystore 类型。如果没有指定密钥存储类型，则使用默认的 JKS (Java Keystore)类型。



注意

在 Kafka 客户端和用于生产环境配置的虚拟集群中推荐 TLS。

PEM 配置示例

```
virtualClusters:
  my-cluster-proxy:
    tls:
      key:
        privateKeyFile: <path>/server.key 1
        certificateFile: <path>/server.crt 2
        keyPassword:
          passwordFile: <path>/key.password 3
# ...
```

1

虚拟集群的私钥。

2

虚拟集群的公共 CA 证书。

3

（可选）密钥的密码。

如果需要，将 insecure 属性配置为禁用信任并与任何 Kafka 集群建立不安全连接，无论证书的有效

性。但是，不建议在生产环境中使用这个选项。

启用不安全的 TLS 的示例

```
virtualClusters:
  demo:
    targetCluster:
      bootstrap_servers: myprivatecluster:9092
      tls:
        trust:
          insecure: true (1)
      #...
# ...
```

(1) 启用不安全的 TLS。

4.2.2. 保护到目标集群的连接

要保护到目标集群的虚拟集群连接，请在虚拟集群中配置 TLS。目标集群必须已配置为使用 TLS。

使用 targetCluster.tls 属性为虚拟集群配置指定 TLS

使用空对象({})来继承 OpenShift 平台的信任。如果目标集群使用由公共 CA 签名的 TLS 证书，则此选项适合。

TLS 的目标集群配置示例

```
virtualClusters:
  my-cluster-proxy:
    targetCluster:
      bootstrap_servers: my-cluster-kafka-bootstrap.kafka.svc.cluster.local:9093
      tls: {}
      #...
```

如果使用由私有 CA 签名的 TLS 证书，您必须为目标集群添加信任存储配置。

目标集群的 truststore 配置示例

```
virtualClusters:
  my-cluster-proxy:
    targetCluster:
      bootstrap_servers: my-cluster-kafka-bootstrap.kafka.svc.cluster.local:9093
      tls:
        trust:
          storeFile: <path>/trust.p12 (1)
          storePassword:
            passwordFile: <path>/store.password (2)
          storeType: PKCS12 (3)
      #...
```

(1) Kafka 集群的公共 CA 证书的 PKCSRG 存储。

(2) 访问公共 Kafka 集群 CA 证书的密码。

(3) （可选） Keystore 类型。如果没有指定密钥存储类型，则使用默认的 JKS (Java Keystore)类型。

对于 mTLS，您也可以为虚拟集群添加密钥存储配置。

mTLS 的 keystore 和 truststore 配置示例

```
virtualClusters:
```

```
  my-cluster-proxy:
    targetCluster:
      bootstrap_servers: my-cluster-kafka-bootstrap.kafka.svc.cluster.local:9093:9092
      tls:
        key:
          privateKeyFile: <path>/client.key 1
          certificateFile: <path>/client.crt 2
        trust:
          storeFile: <path>/server.crt
          storeType: PEM
  # ...
```

1 虚拟集群的私钥。

2 虚拟集群的公共 CA 证书。

对于生产环境外测试的目的，您可以将 insecure 属性设置为 true 以关闭 TLS，以便 Apache Kafka 代理的 Streams 可以连接到任何 Kafka 集群。

关闭 TLS 的配置示例

```
virtualClusters:
  my-cluster-proxy:
    targetCluster:
      bootstrap_servers: myprivatecluster:9092
      tls:
        trust:
          insecure: true
      #...
```

4.3. 配置网络地址

虚拟集群配置需要一个网络地址配置供应商，以管理网络通信并为客户端提供代理地址信息。

Apache Kafka 代理的流有两个内置供应商：

代理地址供应商(PortPerBrokerClusterNetworkAddressConfigProvider)

per-broker 网络地址配置供应商为虚拟集群的 bootstrap 地址打开一个端口，以及目标 Kafka 集群中每个代理的一个端口。端口是动态维护的。例如，如果从集群中删除代理，则分配给它的端口将关闭。

SNI 路由地址供应商(SniRoutingClusterNetworkAddressConfigProvider)

SNI 路由提供程序为所有虚拟集群或端口打开单一端口。对于 Kafka 集群，您可以为整个集群或每个代理打开端口。SNI 路由提供程序使用 SNI 信息来确定路由流量的位置。

代理地址供应商配置示例

```
clusterNetworkAddressConfigProvider:
  type: PortPerBrokerClusterNetworkAddressConfigProvider
  config:
    bootstrapAddress: mycluster.kafka.com:9192 (1)
    brokerAddressPattern: mybroker-$(nodeId).mycluster.kafka.com (2)
    brokerStartPort: 9193 (3)
    numberOfBrokerPorts: 3 (4)
    bindAddress: 192.168.0.1 (5)
```

(1) Kafka 客户端使用的 bootstrap 地址的主机名和端口。

(2) （可选）用于组成代理地址的代理地址模式。如果没有定义，则默认为 bootstrap 地址的主机名部分以及分配给代理的端口号。$(nodeId)令牌被代理的 node.id （如果未设置 node.id ）替代。

(3) （可选）代理端口范围的开始号。默认为 bootstrap 地址的端口加上 1。

(4) （可选）允许的最大代理端口数。默认值为 3。

5

（可选）绑定端口时使用的绑定地址。如果未定义，则绑定所有网络接口。

代理地址配置示例会创建以下代理地址：

```
mybroker-0.mycluster.kafka.com:9193
mybroker-1.mycluster.kafka.com:9194
mybroker-2.mycluster.kafka.com:9194
```

注意

对于具有多个物理集群的配置，请确保将 numberOfBrokerPorts 设置为（代理数量，每个代理的监听程序数）+ 所有集群中的 bootstrap 侦听程序数。例如，如果每个物理集群都有 3 个节点，并且每个代理都有一个监听器，则配置需要一个 8 值（每个集群中为代理监听程序 + 1 端口组成 3 个端口）。

SNI 路由地址供应商配置示例

```
clusterNetworkAddressConfigProvider:
  type: SniRoutingClusterNetworkAddressConfigProvider
  config:
    bootstrapAddress: mycluster.kafka.com:9192 1
    brokerAddressPattern: mybroker-$(nodeId).mycluster.kafka.com
    bindAddress: 192.168.0.1
```

1

所有流量的单一地址，包括 bootstrap 地址和代理。

在 SNI 路由地址配置中，brokerAddressPattern 规格是必需的，因为需要为每个代理生成路由。

第 5 章 指标简介

如果要向 Apache Kafka Proxy 部署的流引入指标，您可以配置不安全的 HTTP 和 Prometheus 端点 (at /metrics)。

在为 Apache Kafka Proxy 配置定义流的 ConfigMap 资源中添加以下内容：

最小指标配置

```
adminHttp:
  endpoints:
    prometheus: {}
```

默认情况下，HTTP 端点侦听 0.0.0.0:9190。您可以按照以下方式更改主机名和端口：

使用主机名和端口的指标配置示例

```
adminHttp:
  host: localhost
  port: 9999
  endpoints:
    prometheus: {}
```

代理提供的示例文件包括 PodMonitor 资源。如果您在 OpenShift 中为用户定义的项目启用了监控，您可以使用 PodMonitor 资源模拟代理指标。

PodMonitor 资源配置示例

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
```

```
  name: proxy
  labels:
    app: proxy
spec:
  selector:
    matchLabels:
      app: proxy
  namespaceSelector:
    matchNames:
      - proxy
  podMetricsEndpoints:
  - path: /metrics
    port: metrics
```

附录 A. 使用您的订阅

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

访问您的帐户

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

激活订阅

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

下载 Zip 和 Tar 文件

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

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

4. 单击组件的 **Download** 链接。

使用 DNF 安装软件包

要安装软件包以及所有软件包的依赖软件包，请使用：

```
dnf install <package_name>
```

要从本地目录中安装之前下载的软件包，请使用：

```
dnf install <path_to_download_package>
```

更新于 2024-04-30