# Red Hat Streams for Apache Kafka 2.7

## OpenShift 上的 Apache Kafka 2.7 发行注记

OpenShift Container Platform 中 Apache Kafka 发行版本的主要新功能及变化信息

Last Updated: 2024-06-26

# Red Hat Streams for Apache Kafka 2.7 OpenShift 上的 Apache Kafka 2.7 发行注记

OpenShift Container Platform 中 Apache Kafka 发行版本的主要新功能及变化信息

## 法律通告



## 摘要

本发行注记总结了 Apache Kafka 2.7 发行版本流中引入的新功能、功能增强和修复。

# 目录

# 第 1 章 APACHE KAFKA 的名称更改为 STREAMS 的通知

AMQ Streams 被重命名为 **Apache Kafka 的流**，作为品牌工作的一部分。这个变化旨在提高 Red Hat Enterprise Linux Kafka 产品的意识。在这个转换期间，您可能会遇到到旧名称 AMQ Streams 的引用。我们正在积极更新文档、资源和媒体，以反映新的名称。

# 第 2 章 功能

Apache Kafka 2.7 的流引入了本节中介绍的功能。

OpenShift 上的 Apache Kafka 2.7 流基于 Apache Kafka 3.7.0 和 Strimzi 0.40.x。

**注意**

要查看本版本中解决的所有增强和错误，请参阅 Apache Kafka JIRA 项目的流。

## 2.1. OPENSHIFT CONTAINER PLATFORM 支持

OpenShift Container Platform 4.12 到 4.15 支持 Apache Kafka 2.7。

如需更多信息，请参阅 第 10 章 *支持的配置*。

## 2.2. KAFKA 3.7.0 支持

Apache Kafka 的流现在支持并使用 Apache Kafka 版本 3.7.0。仅支持由红帽构建的 Kafka 发行版本。

您必须将 Cluster Operator 升级到 Apache Kafka 版本 2.7 的 Streams，然后才能将代理和客户端应用程序升级到 Kafka 3.7.0。有关升级说明，请参阅升级 Apache Kafka 的流。

如需更多信息，请参阅 Kafka 3.7.0 发行注记。

Kafka 3.6.x 仅支持升级到 Apache Kafka 2.7 的流。

**注意**

Kafka 3.7.0 提供对 KRaft 模式的访问，因此 Kafka 在使用 Raft 协议的情况下运行没有 ZooKeeper。

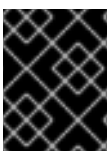
## 2.3. 支持 V1BETA2 API 版本

所有自定义资源的 **v1beta2** API 版本都引进了 Apache Kafka 1.7 的 Streams。对于 Apache Kafka 1.8 的 Streams，**v1alpha1** 和 **v1beta1** API 版本已从所有 Apache Kafka 自定义资源中删除，除了 **KafkaTopic** 和 **KafkaUser** 之外。

将自定义资源升级到 **v1beta2** 为 Apache Kafka 准备流，以迁移到 Kubernetes CRD **v1**，这是 Kubernetes 1.22 所需的。

如果您要从版本 1.7 之前的 Apache Kafka 版本升级：

1. 升级到 Apache Kafka 1.7 的流
2. 将自定义资源转换为 **v1beta2**
3. 升级到 Apache Kafka 1.8 的流



**重要**

在升级到 Apache Kafka 版本 2.7 的 Streams 之前，您必须将自定义资源升级到使用 API 版本 **v1beta2**。

### 2.3.1. 将自定义资源升级到 v1beta2

为了支持将自定义资源升级到 **v1beta2**，Apache Kafka 的 Streams 提供了一个 *API 转换工具*，您可以从 Apache Kafka 1.8 软件下载页面 。

您可以在两个步骤中执行自定义资源升级。

#### 步骤一：转换自定义资源的格式

使用 API 转换工具，您可以将自定义资源的格式转换为适用于 **v1beta2** 的格式：

- 转换描述 Apache Kafka 自定义资源配置的 YAML 文件
- 在集群中直接转换 Apache Kafka 自定义资源的流

另外，您可以手动将每个自定义资源转换为适用于 **v1beta2** 的格式。文档中提供了手动转换自定义资源的说明。

#### 步骤 2：将 CRD 升级到 v1beta2

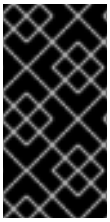
接下来，在 **crd-upgrade** 命令中使用 API 转换工具，您必须将 **v1beta2** 设置为 CRD 中的 *storage* API 版本。您不能手动执行此步骤。

如需更多信息，请参阅从 1.7 之前的 Apache Kafka 版本升级 。

## 2.4. STABLECONNECTIDENTITIES 功能门永久启用

**StableConnectIdentities** 功能门变为 GA （正式发布），现已永久启用。

该功能使用 **StrimziPodSet** 资源来管理 Kafka Connect 和 Kafka MirrorMaker 2 pod，而不是使用 **Deployment** 资源。这有助于最小化连接器任务的重新平衡数量。



**重要**

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

请参阅 **StableConnectIdentities** 功能门。

## 2.5. KAFKANODEPOOLS 功能门现在默认启用

这个 **KafkaNodePools** 功能门进入 beta 版本的成熟度，现在默认启用。功能门通过 **KafkaNodePool** 自定义资源启用不同 Apache Kafka 节点池的配置。

节点池指的是 Kafka 集群中不同的 Kafka 节点组。**KafkaNodePool** 自定义资源仅代表节点池中的节点的配置。每个池都有自己的唯一的配置，其中包括强制设置，如副本数、存储配置和分配的角色列表。由于您可以为节点池中的节点分配角色，您可以尝试使用 ZooKeeper 进行集群管理或 KRaft 模式的 Kafka 集群的功能。您可以分配控制器角色、代理角色或这两个角色。当与基于 ZooKeeper 的 Apache Kafka 集群一起使用时，该角色必须设置为 broker。

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

**禁用 KafkaNodePools 功能门**

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: -KafkaNodePools
```

请参阅配置节点池。

## 2.6. UNIDIRECTIONALTOPICOPERATOR 功能门现在默认启用

**UnidirectionalTopicOperator** 功能门进入 beta 版本的成熟度，现在默认启用。功能门引入了单向主题管理模式。在单向模式中，您可以使用 **KafkaTopic** 资源创建 Kafka 主题，然后由 Topic Operator 管理。

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

**禁用 UnidirectionalTopicOperator 功能门**

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: -UnidirectionalTopicOperator
```

请参阅使用主题 Operator。

## 2.7. KRAFT : 使用KRAFT 功能门现在默认启用

KRaft 模式 Apache Kafka 是一个技术预览，有一些限制，但本发行版本引入了一些支持 KRaft 的新功能。

**UseKRaft** 功能门进入 beta 版本的成熟度，现在默认启用。启用 **UseKRaft** 功能门后，Kafka 集群会在没有 ZooKeeper 的情况下以 KRaft (Kafka Raft metadata)模式部署。要在 KRaft 模式中使用 Kafka，Kafka 自定义资源还必须具有注解 **strimzi.io/kraft="enabled"**。

要使用 KRaft 模式，还必须使用 **KafkaNodePool** 资源来管理节点组的配置。

要禁用 **UseKRaft** 功能门，请指定 **-UseKRaft,-KafkaNodePools** 作为 Cluster Operator 配置中的 **STRIMZI_FEATURE_GATES** 环境变量的值。

**禁用 UseKRaft 功能门**

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: +UseKRaft,+KafkaNodePools
```

请参阅 **UseKRaft** 功能门 和功能增强发行版本。

## 2.8. KRAFT：从基于 ZOOKEEPER 迁移到基于 KRAFT 的 KAFKA 集群的支持

如果您使用 ZooKeeper 在 Kafka 集群中进行元数据管理，您现在可以在 KRaft 模式中迁移到使用 Kafka。

在迁移过程中，您可以执行以下操作：

1. 将控制器节点的仲裁安装为节点池，该池取代了 ZooKeeper 来管理集群。
2. 通过应用 **strimzi.io/kraft="migration"** 注解在集群配置中启用 KRaft 迁移。
3. 使用 **strimzi.io/kraft="enabled"** 注解将代理切换为使用 KRaft 和控制器的迁移模式。

请参阅 迁移到 KRaft 模式。

## 2.9. KRAFT: 对 KRAFT 角色转换的支持

Apache Kafka 的流支持节点转换为不同的 KRaft 角色。通过节点池配置，现在可以执行以下操作：

**合并 KRaft 角色**

使用仅代理和仅控制器角色从单独的节点池过渡到使用双角色节点池。

**分割 KRaft 角色**

将节点池与控制器和代理角色相结合，以使用具有独立角色的两个节点池。

如果在删除节点池配置中的 **代理** 角色时仍分配了分区，则阻止更改。

请参阅 过渡到双角色节点，并过渡到单独的代理和控制器角色。

**注意**

目前，扩展操作只能用于包含作为专用代理运行的节点的代理节点池。

## 2.10. KRAFT：基于 KRAFT 的集群的 KAFKA 升级

现在支持 KRaft 到 KRaft 升级。将基于 KRaft 的 Kafka 集群升级到更新支持的 Kafka 版本和 KRaft 元数据版本。

您可以使用 Kafka 资源中的新 **metadataVersion** 属性指定 Kafka 版本，如 before 和 KRaft **元数据版本**：

**KRaft 元数据版本配置**

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3
    metadataVersion: 3.7-IV2
    version: 3.7.0
    # ...
```

如果没有配置 **metadataVersion**，则 Apache Kafka 的 Streams 会在升级到 Kafka 版本后自动更新到当前默认值。滚动更新可确保每个 pod 在新版本的 Kafka 中使用代理二进制文件。

请参阅升级基于 KRaft 的 Kafka 集群和客户端应用程序。

## 2.11. KAFKA 代理的分层存储

分层存储是一个早期访问 Kafka 功能，它以 开发者预览 的形式在 Apache Kafka 中提供。

## 2.12. RHEL 7 不再被支持

RHEL 7 不再被支持。决策是由于 已知不兼容问题 而造成的。

# 第 3 章 功能增强

Apache Kafka 2.7 的流添加了很多改进。

## 3.1. KAFKA 3.7.0 的改进

有关 Kafka 3.7.0 中引入的增强功能概述，请参阅 Kafka 3.7.0 发行注记。

## 3.2. 改进了代理缩减处理

默认情况下，Apache Kafka 的 Streams 会执行检查，以确保在 Kafka 集群上启动缩减操作前，代理中没有分区副本。在以前的版本中，如果发现代理仍在使用，则缩减操作会被阻止，协调会失败。现在，会恢复缩减更改，但协调可以正常进行。

然而，在有些情况下，您可能想要绕过此阻塞机制。例如，在忙碌的集群中禁用检查可能很有用。要做到这一点，您可以通过将 **strimzi.io/skip-broker-scaledown-check** 设置为 **true** 来注解 **Kafka** 资源。

请参阅 对缩减操作进行跳过检查。

## 3.3. KAFKA EXPORTER：在断开连接的环境中防止指标集合

现在，您可以停止在不再连接的消费者上收集指标。要停止在断开连接的消费者上收集指标，请在 Kafka Exporter 配置中将 **showAllOffsets** 属性设置为 **false**。

请参阅 KafkaExporterSpec 模式参考。

## 3.4. 使 CRD 标签和注解一致

我们的 CRD 中的标签和注解定义（如 **template** 部分中的）不再接受整数值，且必须始终使用字符串值。

例如，这个配置：

**带有注解的模板配置示例**

```
template:
  apiService:
    metadata:
      annotations:
        discovery.myapigateway.io/port: 8080
```

必须如下所示：

**带有注解的模板配置示例**

```
template:
  apiService:
    metadata:
      annotations:
        discovery.myapigateway.io/port: "8080"
```

这是为了避免接收无效值错误的可能性：

**无效的值错误示例**

```
spec.template.apiService.metadata.annotations.discovery.myapigateway.io/port:
Invalid value: "integer":
spec.template.apiService.metadata.annotations.discovery.myapigateway.io/port in body must be of
type string: "integer"
```

## 3.5. DRAIN CLEANER：更改为处理驱除请求

Apache Kafka Drain Cleaner 的流处理 Kubernetes pod 驱除请求的方式已更改。默认情况下，Drain Cleaner 现在拒绝（阻止）Kubernetes 驱除请求，以防止 Kubernetes 驱除 pod，而是使用 Cluster Operator 来移动 pod。

您仍然可以使用前面的方法，其中 Drain Cleaner 允许 Kubernetes 驱除请求，同时还指示 Cluster Operator 移动 pod。要使这个传统模式正常工作，您必须执行以下操作：

1. 将 **STRIMZI_DENY_EVICTION** Drain Cleaner 环境变量窃取为 **false**。
2. 通过将 **maxUnavailable** 选项设置为 0，将 **PodDisruptionBudget** 配置为不允许任何 pod 驱除。

**请参阅 使用 Strimzi Drain Cleaner 驱除 pod。**

### 3.6. 增强的 KAFKA CONNECT 指标和仪表板示例

**为 Kafka Connect 收集的指标已被改进，使其包含以下内容：**

- **协调器指标的集合**
- **简化的指标正则表达式**

**Kafka Connect 的 Grafana 仪表板文件示例已更新，以添加显示更多连接器信息和重新平衡指标的面板。**

### 3.7. 增强的 MIRRORMAKER 2 仪表板示例

**MirrorMaker 2 的 Grafana 仪表板文件示例已更新，如下所示：**

- **用于组织面板的新行**

- 更新了显示更多连接器信息和重新平衡指标的面板

### 3.8. KAFKA BRIDGE 文本格式

在执行制作者操作时，POST 请求必须提供 Content-Type 标头来指定 *生成的消息的嵌入式数据格式*。在以前的版本中，JSON 和二进制是记录和键值支持的格式。现在也可以使用 文本格式。

表 3.1. 支持的内容类型格式

| 嵌入式数据格式 | Content-Type 标头 |
| --- | --- |
| JSON | **content-Type: application/vnd.kafka.json.v2+json** |
| 二进制 | **content-Type: application/vnd.kafka.binary.v2+json** |
| 文本 | **content-Type: application/vnd.kafka.text.v2+json** |

# 第 4 章 技术预览

Apache Kafka 2.7 的 Streams 中包含的技术预览功能。

重要

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。

## 4.1. KRAFT 模式

KRaft 模式作为技术预览提供。

目前，Apache Kafka Streams 中的 KRaft 模式有以下限制：

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

## 4.2. APACHE KAFKA 控制台流

Apache Kafka 的流控制台（用户界面）现在作为技术预览提供。Apache Kafka 控制台的流旨在与您的 Streams for Apache Kafka 部署无缝集成，为监控和管理 Kafka 集群提供了一个集中 hub。部署控制台并将其连接到由 Streams for Apache Kafka 管理的 Kafka 集群。

通过专用页面，深入了解每个连接的集群，包括代理、主题和消费者组。在查看代理、主题或连接的消费者组的具体信息前，查看 Kafka 集群的状态，如 Kafka 集群的状态。

请参阅 Apache Kafka 控制台的 Streams 指南。

### 4.3. APACHE KAFKA 代理流

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

作为技术预览的一部分，您可以尝试 Apache Kafka Proxy Record Encryption 过滤器的流。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。

请参阅 Apache Kafka 代理的流指南。

### 4.4. KAFKA 静态配额插件配置

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

**Kafka 静态配额插件配置示例**

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    config:
      client.quota.callback.class: io.strimzi.kafka.quotas.StaticQuotaCallback
      client.quota.callback.static.produce: 1000000
      client.quota.callback.static.fetch: 1000000
      client.quota.callback.static.storage.soft: 400000000000
      client.quota.callback.static.storage.hard: 500000000000
      client.quota.callback.static.storage.check-interval: 5
```

请参阅有关使用 Kafka 静态配额插件的代理设置限制。

# 第 5 章 开发者预览

Apache Kafka 2.7 的 Streams 中包含的开发人员预览功能。

作为 Kafka 集群管理员，您可以使用 Cluster Operator 部署配置中的功能门切换和关闭功能子集。作为开发人员预览的功能门处于 alpha 程度，默认是禁用的。

重要

开发者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。此开发者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，请参阅开发者预览支持范围。

## 5.1. KAFKA 代理的分层存储

Apache Kafka 的流现在支持 Kafka 代理的分层存储作为开发者预览，允许您引入自定义远程存储解决方案和本地存储。由于其 当前限制，不建议在生产环境中使用。

远程存储配置使用 Kafka 资源中的 kafka.layerStorage 属性来指定。您可以指定一个自定义远程存储管理器来管理分层存储。

自定义分层存储配置示例

```
kafka:
  tieredStorage:
    type: custom
    remoteStorageManager:
      className: com.example.kafka.tiered.storage.s3.S3RemoteStorageManager
      classPath: /opt/kafka/plugins/tiered-storage-s3/*
      config:
        # remote storage manager configuration (1)
        storage.bucket.name: my-bucket
  config:
    ...
    rlmm.config.remote.log.metadata.topic.replication.factor: 1 (2)
```

1

使用必要的设置配置自定义远程存储管理器。密钥自动作为 rsm.config 前缀，并附加到 Kafka 代理配置中。

2

Apache Kafka 的 Streams 使用 TopicBasedRemoteLogMetadataManager for Remote Log Metadata Management (RLMM)。使用 rlmm.config. 前缀添加 RLMM 配置。

注意

如果要使用自定义分层存储，您必须首先通过构建自定义容器镜像将分层存储插件添加到 Apache Kafka 镜像的 Streams 中。

请参阅分层存储（早期访问）。

# 第 6 章 KAFKA 中断更改

本节论述了对 Kafka 的任何需要更改 Apache Kafka 的 Streams 的更改才能继续工作。

## 6.1. 使用 KAFKA 的示例文件连接器

Kafka 不再在其 CLASSPATH 和 plugin.path 中包含示例文件连接器 FileStreamSourceConnector 和 FileStreamSinkConnector。Apache Kafka 的流已更新，您仍然可以使用这些示例连接器。现在，必须将示例添加到插件路径中，如任何连接器。

Apache Kafka 的流提供了一个示例连接器配置文件，其中包含将文件连接器部署为 KafkaConnector 资源所需的配置：

- examples/connect/source-connector.yaml

请参阅部署示例 KafkaConnector 资源，并使用连接器插件扩展 Kafka 连接。

# 第 7 章 已弃用的功能

之前的 Apache Kafka Streams 版本支持的已弃用的功能。

## 7.1. 模式属性弃用

| 模式 | 弃用的属性 | 替换属性 |
| --- | --- | --- |
| AclRule | operation | operation |
| CruiseControlSpec | tlsSidecar | - |
| CruiseControlTemplate | tlsSidecarContainer | - |
| CruiseControlSpec.BrokerCapacity | disk | - |
| CruiseControlSpec.BrokerCapacity | cpuUtilization | - |
| EntityUserOperator | zookeeperSessionTimeoutSeconds | - |
| JaegerTracing | type | - |
| KafkaConnectorSpec | pause | state |
| KafkaConnectTemplate | 部署 | 由 StrimziPodSet 资源替代 |
| KafkaClusterTemplate | statefulset | 由 StrimziPodSet 资源替代 |
| KafkaExporterTemplate | service | - |
| KafkaMirrorMaker | 所有属性 | - |
| KafkaMirrorMaker2ConnectorSpec | pause | state |
| KafkaMirrorMaker2MirrorSpec | topicsBlacklistPattern | topicsExcludePattern |
| KafkaMirrorMaker2MirrorSpec | groupsBlacklistPattern | groupsExcludePattern |
| ListenerStatus | type | name |

| 模式 | 弃用的属性 | 替换属性 |
| --- | --- | --- |
| ZookeeperClusterTemplate | statefulset | 由 StrimziPodSet 资源替代 |

请参阅 Apache Kafka 自定义资源 API 参考 的流。

## 7.2. JAVA 11 在 APACHE KAFKA 2.7.0 的流中被弃用

Kafka 3.7.0 和 Streams for Apache Kafka 2.7.0 中已弃用对 Java 11 的支持。对于 Apache Kafka 组件（包括客户端）的所有流（包括客户端）将不支持 Java 11。

Apache Kafka 的流支持 Java 17。在开发新应用程序时使用 Java 17。计划将当前使用 Java 11 的任何应用程序迁移到 17。

注意

在 Apache Kafka 2.4.0 的 Streams 中删除了对 Java 8 的支持。如果您当前正在使用 Java 8，计划以同样的方式迁移到 Java 17。

## 7.3. 环境变量配置供应商已弃用

您可以使用配置供应商为所有 Kafka 组件从外部来源加载配置数据，包括生成者和消费者。

在以前的版本中，您可以使用组件的 spec 配置中的 config.providers 属性启用 secrets.io.strimzi.kafka.EnvVarConfigProvider 环境变量配置供应商。但是，这个供应商现已弃用，并将在以后的版本中删除。因此，建议您更新您的实现以使用 Kafka 自己的环境变量配置供应商 (org.apache.kafka.common.config.provider.EnvVarConfigProvider)提供配置属性作为环境变量。

启用环境变量配置供应商的配置示例

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

```
config:
  # ...
  config.providers: env
  config.providers.env.class:
org.apache.kafka.common.config.provider.EnvVarConfigProvider
  # ...
```

## 7.4. KAFKA MIRRORMAKER 2 身份复制策略

身份复制策略是 MirrorMaker 2 的一个功能，用于覆盖远程主题的自动重命名。主题不会使用源集群的名称添加名称，而是保留其原始名称。此设置对主动/被动备份和数据迁移场景特别有用。

要实现身份复制策略，您必须在 MirrorMaker 2 配置中指定复制策略类(replication.policy.class)。在以前的版本中，您可以指定位于 Apache Kafka mirror-maker-2-extensions 组件的 Streams 中的 io.strimzi.kafka.connect.mirror.IdentityReplicationPolicy 类。但是，这个组件现已弃用，并将在以后的版本中删除。因此，建议您更新您的实现以使用 Kafka 自己的复制策略类(org.apache.kafka.connect.mirror.IdentityReplicationPolicy)。

请参阅配置 Kafka MirrorMaker 2。

## 7.5. KAFKA MIRRORMAKER 1

Kafka MirrorMaker 在两个或更多活跃 Kafka 集群之间复制数据，并在数据中心之间复制数据。Kafka MirrorMaker 1 在 Kafka 3.0.0 中已弃用，并将在 Kafka 4.0.0 中删除。MirrorMaker 2 将是唯一可用的版本。MirrorMaker 2 基于 Kafka Connect 框架，连接器管理集群之间的数据传输。

因此，用于部署 Kafka MirrorMaker 1 的 Apache Kafka KafkaMirrorMaker 自定义资源的 Streams 已被弃用。当使用 Kafka 4.0.0 时，KafkaMirrorMaker 资源将从 Apache Kafka 的 Streams 中删除。

如果您使用 MirrorMaker 1 （称为 Apache Kafka 文档中的 Streams 中的 *MirrorMaker*），使用带有 IdentityReplicationPolicy 类的 KafkaMirrorMaker2 自定义资源。MirrorMaker 2 将复制的主题重命名为目标集群。IdentityReplicationPolicy 配置会覆盖自动重命名。使用它生成与 MirrorMaker 1 相同的主动/被动单向复制。

请参阅配置 Kafka MirrorMaker 2。

### 7.6. KAFKA BRIDGE SPAN 属性

以下 Kafka Bridge span 属性已弃用，在适用的情况下显示替换：

- http.method 被 http.request.method替代
- http.url 被 url.scheme,url.path, 和 url.query替代
- messaging.destination 替换为 messaging.destination.name
- http.status_code 替换为 http.response.status_code
- messaging.destination.kind=topic 没有替换

Kafka Bridge 使用 OpenTelemetry 进行分布式追踪。更改是内联的，并带有 OpenTelemetry 语义惯例的更改。这些属性将在以后的 Kafka Bridge 发行版本中删除

# 第 8 章 修复的问题

**OpenShift 中 Apache Kafka 2.7 的 Streams 中修复的问题。**

**有关 Kafka 3.7.0 中修复的问题的详情，请参考 Kafka 3.7.0 发行注记。**

**表 8.1. 修复的问题**

| 问题号 | 描述 |
| --- | --- |
| ENTMQST-5839 | OAuth 问题修复： **fallbackUsernamePrefix** 没有效果 |
| ENTMQST-5820 | 由于生成的 **sasl.jaas.config** 值中不需要的引号，所以启用了 Oauth 的 MM2 连接器任务会失败。 |
| ENTMQST-5754 | 在资源被删除时避免不必要的补丁 |
| ENTMQST-5753 | 使用多个 HTTP 请求中的不同嵌入式格式生成不会导致 |
| ENTMQST-5656 | 当 API secret 发生变化时，Cruise Control 不会重启 |
| ENTMQST-5603 | **UseKRaft** 功能门提升到 beta |
| ENTMQST-5583 | 从 BTO 升级时，单向主题 Operator 似乎会导致中断 |
| ENTMQST-5582 | Cruise 控制主题应该使用正确的配置来平稳进行滚动更新并确保可用性 |
| ENTMQST-5581 | 单向主题 Operator 需要以更好的方式使用日志级别 |
| ENTMQST-5546 | **KafkaRoller** 被计算，将仅控制器节点转换到混合节点 |
| ENTMQST-5540 | 修复 MirrorMaker 2 连接器的连接器状态处理 |
| ENTMQST-5511 | KRaft 节点滚动、存活度和就绪度 |
| ENTMQST-5504 | 在启用 KRaft 时添加对 Kafka 和 Strimzi 升级的支持 |
| ENTMQST-5492 | 修复控制器节点的公告监听程序的处理 |
| ENTMQST-5387 | 将 **StableConnectIdentities** 功能门提升到 GA |
| ENTMQST-5383 | 支持带有自定义"bring-own"插件的分层存储 |
| ENTMQST-5360 | 其他用于统一主题 Operator 的任务(2.7.0 Edition) |

| 问题号 | 描述 |
| --- | --- |
| ENTMQST-5292 | 在从现有 PV / PVC 恢复时处理集群 ID 验证的问题 |
| ENTMQST-4194 | 主题 Operator 允许用户设置禁止的设置 |
| ENTMQST-4164 | 定期协调内部主题的持久性错误 |
| ENTMQST-4087 | 在进行批量主题删除时，主题 Operator 会失败 |
| ENTMQST-3970 | 内部 Kafka Connect 主题会使用无效的配置重新创建 |
| ENTMQST-3994 | zookeeper 到 KRaft 迁移 |
| ENTMQST-3974 | 主题 Operator 修复 |
| ENTMQST-3886 | 状态存储( topic-store)可能已迁移到另一个实例 |

**表 8.2. 修复的常见漏洞和风险(CVE)**

| 问题号 | 描述 |
| --- | --- |
| ENTMQST-5886 | CVE-2023-43642 安全漏洞在 snappy-java 中的 SnappyInputStream 中找到 |
| ENTMQST-5885 | CVE-2023-52428 Nimbus JOSE+JWT before 9.37.2 |
| ENTMQST-5884 | CVE-2022-4899 安全漏洞在 zstd v1.4.10 中找到 |
| ENTMQST-5883 | CVE-2021-24032 安全漏洞包括在 zstd 中 |
| ENTMQST-5882 | CVE-2024-23944 Apache ZooKeeper：持久监视器处理中的信息披露 |
| ENTMQST-5881 | CVE-2021-3520 在 lz4 中存在一个缺陷 |
| ENTMQST-5835 | CVE-2024-29025 netty-codec-http: 资源分配没有限制或 Throttling |
| ENTMQST-5646 | CVE-2024-1023 vert.x: io.vertx/vertx-core: 内存泄漏，因为在 Vertx 中使用 Netty FastThreadLocal 数据结构 |
| ENTMQST-5667 | 当 TCP 服务器配置了 TLS 和 SNI 支持时，CVE-2024-1300 vertx-core: io.vertx:vertx-core: leak |

# 第 9 章 已知问题

本节列出了 OpenShift 中 Apache Kafka 2.7 的 Streams 的已知问题。

## 9.1. 与 RHEL 7 不兼容

在 Kafka 3.7 中使用 RHEL 7 时已知的不兼容问题。因此，RHEL 7 不再被支持。造成这个问题的原因是，在 RHEL 7 上有一个过时的 GCC (GNU Compiler Collection)版本，它与 RocksDB JNI 库的版本不兼容(org.rocksdb:rocksdb:rocksdbjni:7.9.2)。

RocksDB JNI 版本 7.9.2 需要较新版本的 GCC 和相关的 libstdc++ 库，而不是 RHEL 7 中的可用内容。由于这些过时的库，使用 snappy 压缩和 Kafka Streams （依赖于 RocksDB）在 RHEL 7 上无法正常工作。

建议

- 将运行在 RHEL 7 上的客户端升级到 RHEL 8，以确保与 Kafka 3.7 和最新的 Apache Kafka 功能兼容。
- 如果要继续使用 RHEL 7，请考虑使用 Streams for Apache Kafka 2.5 LTS 或 2.6。

## 9.2. MIRRORMAKER 2 连接器的自动重启

MirrorMaker 2 连接器的自动重新启动无法正常工作，因为状态没有传递给连接器 Operator。以后的 Apache Kafka Streams 发行版本中会解决这个问题。

## 9.3. IPV6 集群上的 APACHE KAFKA CLUSTER OPERATOR 流

Apache Kafka Cluster Operator 的 Streams 不会在互联网协议版本 6 (IPv6)集群中启动。

临时解决方案

这个问题有两个临时解决方案。

临时解决方案：设置 KUBERNETES_MASTER 环境变量

1. 显示 OpenShift Container Platform 集群的 Kubernetes master 节点地址：

```
oc cluster-info
Kubernetes master is running at <master_address>
# ...
```

复制 master 节点的地址。

2. 列出所有 Operator 订阅：

```
oc get subs -n <operator_namespace>
```

3. 编辑 Apache Kafka 的 Streams 的订阅资源：

```
oc edit sub amq-streams -n <operator_namespace>
```

4. 在 spec.config.env 中，添加 KUBERNETES_MASTER 环境变量，设置为 Kubernetes master 节点的地址。例如：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amq-streams
  namespace: <operator_namespace>
spec:
  channel: amq-streams-1.8.x
  installPlanApproval: Automatic
  name: amq-streams
  source: mirror-amq-streams
  sourceNamespace: openshift-marketplace
  config:
    env:
    - name: KUBERNETES_MASTER
      value: MASTER-ADDRESS
```

5. 保存并退出编辑器。

6. 检查 订阅 是否已更新：

```
oc get sub amq-streams -n <operator_namespace>
```

7. 检查 Cluster Operator Deployment 是否已更新为使用新环境变量：

```
oc get deployment <cluster_operator_deployment_name>
```

临时解决方案：禁用主机名验证

1. 列出所有 Operator 订阅：

```
oc get subs -n <operator_namespace>
```

2. 编辑 Apache Kafka 的 Streams 的订阅资源：

```
oc edit sub amq-streams -n <operator_namespace>
```

3. 在 spec.config.env 中，添加 KUBERNETES_DISABLE_HOSTNAME_VERIFICATION 环境变量，设为 true。例如：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amq-streams
  namespace: <operator_namespace>
spec:
  channel: amq-streams-1.8.x
  installPlanApproval: Automatic
  name: amq-streams
  source: mirror-amq-streams
  sourceNamespace: openshift-marketplace
  config:
    env:
    - name: KUBERNETES_DISABLE_HOSTNAME_VERIFICATION
      value: "true"
```

4. 保存并退出编辑器。

5. 检查 订阅 是否已更新：

```
oc get sub amq-streams -n <operator_namespace>
```

6. 检查 Cluster Operator Deployment 是否已更新为使用新环境变量：

```
oc get deployment <cluster_operator_deployment_name>
```

### 9.4. 崩溃控制 CPU 利用率估算

Cruise Control for Apache Kafka 存在一个已知问题，与 CPU 使用率估算的计算相关。CPU 使用率计算为代理 pod 定义容量的百分比。在运行具有不同 CPU 内核的节点中的 Kafka 代理时，会出现这个问题。例如，node1 可能有 2 个 CPU 内核，node2 有 4 个 CPU 内核。在这种情况下，Cruise Control 可能会低估或高估了 CPU 的负载。这个问题可能会在 pod 负载过重时导致集群无法重新平衡。

这个问题有两个临时解决方案。

临时解决方案： Equal CPU 请求和限值

您可以在 Kafka.spec.kafka.resources 中设置与 CPU 限制相等的 CPU 请求。这样，所有 CPU 资源都会保留前期，并且始终可用。此配置允许 Cruise Control 在准备基于 CPU 目标的重新平衡建议时，正确评估 CPU 利用率。

临时解决方案：排除 CPU 目标

您可以从 Cruise Control 配置中指定的硬和默认目标中排除 CPU 目标。

没有 CPU 目标的 Cruise Control 配置示例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    topicOperator: {}
    userOperator: {}
  cruiseControl:
    brokerCapacity:
      inboundNetwork: 10000KB/s
      outboundNetwork: 10000KB/s
    config:
      hard.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.MinTopicLeadersPerBrokerGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal,
```

```
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundCapacityGoal
      default.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.MinTopicLeadersPerBrokerGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaDistributionGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.PotentialNwOutGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskUsageDistributionGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundUsageDistributionGoal,

    com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundUsageDistributionGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.TopicReplicaDistributionGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderReplicaDistributionGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderBytesInDistributionGoal
```

如需更多信息，请参阅 Insufficient CPU 容量。

### 9.5. 以 FIPS 模式运行时的 JMX 身份验证

当在启用了 JMX 身份验证的 FIPS 模式下运行 Apache Kafka 时，客户端可能会失败。要临时解决这个问题，在以 FIPS 模式运行时不要启用 JMX 身份验证。我们正在调查此问题，并在以后进行解决。

# 第 10 章 支持的配置

**Apache Kafka 2.7 发行版本流支持的配置。**

## 10.1. 支持的平台

**以下平台为在 OpenShift 提供的版本中使用 Kafka 运行的 Apache Kafka 2.7 测试了流。**

| 平台 | Version | 架构 |
|---|---|---|
| Red Hat OpenShift Container Platform | 4.12 到 4.15 | x86_64, ppc64le (IBM Power), s390x (IBM Z 和 IBM® LinuxONE), aarch64 (64-bit ARM) |
| Red Hat OpenShift Container Platform 断开连接的环境 | Latest | x86_64, ppc64le (IBM Power), s390x (IBM Z 和 IBM® LinuxONE), aarch64 (64-bit ARM) |
| Red Hat OpenShift Dedicated | Latest | x86_64 |
| Microsoft Azure Red Hat OpenShift (ARO) | Latest | x86_64 |
| Red Hat OpenShift Service on AWS (ROSA)<br>包含带有托管的 control plane (HCP)的 ROSA | Latest | x86_64 |
| Red Hat MicroShift | Latest | x86_64 |
| Red Hat OpenShift Local | 2.13-2.19 (OCP 4.12), 2.20-2.28 (OCP 4.13), 2.29-2.33 (OCP 4.14), 2.34 及更新版本 (OCP 4.15) | x86_64 |

**OpenShift Local 是 Red Hat OpenShift Container Platform (OCP)的一个有限版本。仅用于开发和评估某些功能可能不可用。**

**不支持的功能**

- **Red Hat MicroShift 不支持 Kafka Connect 的构建配置，用于使用连接器构建容器镜像。**

- IBM Z 和 IBM® LinuxONE s390x 架构不支持 Apache Kafka OPA 集成。

### FIPS 合规性

Apache Kafka 2.7.0 的流是为 FIPS 设计的。Apache Kafka 容器镜像流基于 RHEL 9.2，它已提交到 NIST 进行批准。

要检查在美国国家标准与技术研究院(NIST)批准的 RHEL 版本，请查看 NIST 网站上 的加密模块验证计划。

Red Hat OpenShift Container Platform 为 FIPS 设计。当在 RHEL 或 RHEL CoreOS 中以 FIPS 模式运行时，OpenShift Container Platform 核心组件仅在 x86_64、ppc64le (IBM Power)、s390x (IBM Z)和 aarch64 (64 位 ARM)架构中使用提交到 NIST 的 RHEL 加密库。有关 NIST 验证程序的更多信息，请参阅加密模块验证程序。有关为验证提交的 RHEL 加密库的单独版本的最新 NIST 状态，请参阅 Compliance Activities 和 Government Standards。

OpenShift Container Platform 4.12 是支持 FIPS 140-2 的最后一个版本。鉴于 NIST 未来 OpenShift 版本的验证时间表的不确定性，OpenShift 4.12 上支持 Apache Kafka 的流，直到进一步通知为止。

## 10.2. 支持的客户端

对于 Apache Kafka，只支持由红帽构建的客户端库。目前，Apache Kafka 的流仅提供 Java 客户端库。在以下操作系统和构架中，支持客户端与 Apache Kafka 2.7 的 Streams 搭配使用：

| 操作系统 | 架构 | JVM |
| --- | --- | --- |
| RHEL 和 UBI 8 和 9 | x86, amd64, ppc64le (IBM Power), s390x (IBM Z 和 IBM® LinuxONE), aarch64 (64-bit ARM) | Java 11 （已弃用）和 Java 17 |

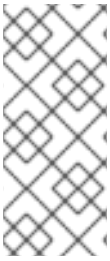
客户端使用 Open JDK 11 和 17 测试，但 Java 11 在 Apache Kafka 2.7.0 的 Streams 中弃用。IBM JDK 被支持，但不在每个发行版本中定期测试。不支持 Oracle JDK 11。

对 Red Hat Universal Base Image (UBI)版本的支持与相同的 RHEL 版本对应。

## 10.3. 支持的 APACHE KAFKA 生态系统

在 Apache Kafka 的 Streams 中，只支持从 Apache Software Foundation 直接发布的以下组件：

- Apache Kafka Broker
- Apache Kafka Connect
- Apache MirrorMaker
- Apache MirrorMaker 2
- Apache Kafka Java Producer, Consumer, Management client, 和 Kafka Streams
- Apache ZooKeeper



注意

Apache ZooKeeper 仅作为 Apache Kafka 的实现详情支持，不应出于其他目的进行修改。另外，分配给 ZooKeeper 节点的内核数或 vCPU 不包含在订阅合规计算中。换句话说，ZooKee 节点不会计算客户的订阅。

## 10.4. 其他支持的功能

- Kafka Bridge
- Drain Cleaner
- Sything Control
- 分布式追踪

- Apache Kafka 控制台流（技术预览）
- Apache Kafka 代理的流（技术预览）

**注意**

Apache Kafka 代理的流和 Apache Kafka 代理的流不是生产环境就绪的。对于技术预览，它们仅在 x86 和 amd64 上进行测试。

另请参阅 *第 12 章 支持的与红帽产品集成*。

## 10.5. 存储要求

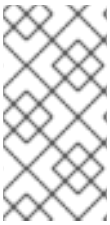
Apache Kafka 的流已使用块存储测试，并与 XFS 和 ext4 文件系统兼容，它们都通常与 Kafka 一起使用。文件存储选项（如 NFS）不兼容。

### 其他资源

有关 Apache Kafka 2.5 LTS 发行版本的流支持的配置的详情，请参考 Apache Kafka 2.5 发行注记。

## 第 11 章 组件详情

下表显示了 Apache Kafka 的每个流的组件版本。



**注意**

Operator、控制台和代理等组件仅适用于在 OpenShift 中使用 Streams for Apache Kafka。

| Apache Kafka Streams | Apache Kafka | Strimzi Operators | Kafka Bridge | Oauth | Sything Control | 控制台（Console） | Proxy |
|---|---|---|---|---|---|---|---|
| 2.7.0 | 3.7.0 | 0.40.0 | 0.28 | 0.15.0 | 2.5.128 | 0.1 | 0.5.1 |
| 2.6.0 | 3.6.0 | 0.38.0 | 0.27 | 0.14.0 | 2.5.128 | - | - |
| 2.5.1 | 3.5.0 | 0.36.0 | 0.26 | 0.13.0 | 2.5.123 | - | - |
| 2.5.0 | 3.5.0 | 0.36.0 | 0.26 | 0.13.0 | 2.5.123 | - | - |
| 2.4.0 | 3.4.0 | 0.34.0 | 0.25.0 | 0.12.0 | 2.5.112 | - | - |
| 2.3.0 | 3.3.1 | 0.32.0 | 0.22.3 | 0.11.0 | 2.5.103 | - | - |
| 2.2.2 | 3.2.3 | 0.29.0 | 0.21.5 | 0.10.0 | 2.5.103 | - | - |
| 2.2.1 | 3.2.3 | 0.29.0 | 0.21.5 | 0.10.0 | 2.5.103 | - | - |
| 2.2.0 | 3.2.3 | 0.29.0 | 0.21.5 | 0.10.0 | 2.5.89 | - | - |
| 2.1.0 | 3.1.0 | 0.28.0 | 0.21.4 | 0.10.0 | 2.5.82 | - | - |
| 2.0.1 | 3.0.0 | 0.26.0 | 0.20.3 | 0.9.0 | 2.5.73 | - | - |
| 2.0.0 | 3.0.0 | 0.26.0 | 0.20.3 | 0.9.0 | 2.5.73 | - | - |
| 1.8.4 | 2.8.0 | 0.24.0 | 0.20.1 | 0.8.1 | 2.5.59 | - | - |
| 1.8.0 | 2.8.0 | 0.24.0 | 0.20.1 | 0.8.1 | 2.5.59 | - | - |
| 1.7.0 | 2.7.0 | 0.22.1 | 0.19.0 | 0.7.1 | 2.5.37 | - | - |
| 1.6.7 | 2.6.3 | 0.20.1 | 0.19.0 | 0.6.1 | 2.5.11 | - | - |

| Apache Kafka Streams | Apache Kafka | Strimzi Operators | Kafka Bridge | Oauth | Sything Control | 控制台（Console） | Proxy |
|---|---|---|---|---|---|---|---|
| 1.6.6 | 2.6.3 | 0.20.1 | 0.19.0 | 0.6.1 | 2.5.11 | - | - |
| 1.6.5 | 2.6.2 | 0.20.1 | 0.19.0 | 0.6.1 | 2.5.11 | - | - |
| 1.6.4 | 2.6.2 | 0.20.1 | 0.19.0 | 0.6.1 | 2.5.11 | - | - |
| 1.6.0 | 2.6.0 | 0.20.0 | 0.19.0 | 0.6.1 | 2.5.11 | - | - |
| 1.5.0 | 2.5.0 | 0.18.0 | 0.16.0 | 0.5.0 | - | - | - |
| 1.4.1 | 2.4.0 | 0.17.0 | 0.15.2 | 0.3.0 | - | - | - |
| 1.4.0 | 2.4.0 | 0.17.0 | 0.15.2 | 0.3.0 | - | - | - |
| 1.3.0 | 2.3.0 | 0.14.0 | 0.14.0 | 0.1.0 | - | - | - |
| 1.2.0 | 2.2.1 | 0.12.1 | 0.12.2 | - | - | - | - |
| 1.1.1 | 2.1.1 | 0.11.4 | - | - | - | - | - |
| 1.1.0 | 2.1.1 | 0.11.1 | - | - | - | - | - |
| 1.0 | 2.0.0 | 0.8.1 | - | - | - | - | - |

**注意**

**Strimzi 0.26.0 包括一个 Log4j 漏洞。产品中包含的版本已更新，以取决于不包含漏洞的版本。**

# 第 12 章 支持的与红帽产品集成

Apache Kafka 2.7 的流支持与以下红帽产品集成：

红帽构建的 Keycloak

提供 OAuth 2.0 身份验证和 OAuth 2.0 授权。

Red Hat 3scale API Management

保护 Kafka Bridge 并提供额外的 API 管理功能。

红帽构建的 Debezium

监控数据库并创建事件流。

红帽构建的 Apicurio Registry

为数据流提供服务架构的集中存储。

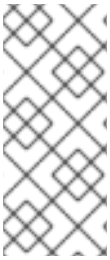
红帽构建的 Apache Camel K

提供轻量级集成框架。

有关这些产品可在 Apache Kafka 部署的 Streams 中引入的功能信息，请参阅产品文档。

## 12.1. RED HAT BUILD OF KEYCLOAK （以前称为 RED HAT SINGLE SIGN-ON）

Apache Kafka 的流通过红帽构建的 Keycloak 授权服务 支持基于 OAuth 2.0 令牌的授权，从而提供对安全策略和权限的集中管理。



注意

Red Hat build of Keycloak 替换了 Red Hat Single Sign-On，它现在处于维护支持中。我们正努力更新文档、资源和媒体，以反映这种转换。在此期间，在 Apache Kafka 的 Streams 中使用 Single Sign-On 的内容也适用于使用 Keycloak 的红帽构建。

## 12.2. RED HAT 3SCALE API MANAGEMENT

如果在 OpenShift Container Platform 上部署了 Kafka Bridge，您可以在 3scale 中使用它。3scale API 管理可以使用 TLS 保护 Kafka Bridge，并提供身份验证和授权。与 3scale 集成还意味着可以使用

metrics、速率限制和计费等额外功能。

有关部署 3scale 的详情，请参考在 Apache Kafka Bridge 中使用 3scale API 管理。

## 12.3. 红帽构建的 DEBEZIUM 用于更改数据捕获

红帽构建的 Debezium 是一个分布式更改数据捕获平台。它捕获数据库中的行级更改，创建更改事件记录，并将记录流传输到 Kafka 主题。Debezium 基于 Apache Kafka 构建。您可以将红帽构建的 Debezium 与 Apache Kafka 的 Streams 一起部署并集成。在部署了 Apache Kafka 的 Streams 后，您可以通过 Kafka Connect 将 Debezium 部署为连接器配置。Debezium 将更改事件记录传递给 OpenShift 上的 Apache Kafka 的流。应用程序可以读取 *这些更改事件流*，并按发生更改事件的顺序访问更改事件。

有关使用 Apache Kafka 的流部署 Debezium 的更多信息，请参阅 红帽构建的 Debezium 产品文档。

## 12.4. 红帽构建的 APICURIO REGISTRY 用于 SCHEMA 验证

您可以使用红帽构建的 Apicurio Registry 作为数据流的服务架构的集中存储。对于 Kafka，您可以使用红帽构建的 Apicurio Registry 来存储 *Apache Avro* 或 JSON 模式。

Apicurio Registry 提供 REST API 和 Java REST 客户端，用于通过服务器端端点从客户端应用注册和查询架构。

使用 Apicurio Registry 将管理模式的过程与客户端应用程序配置分离。您可以通过在客户端代码中指定 URL 来启用应用程序从 registry 中使用 schema。

例如，消息序列化和反序列化的架构可以存储在注册表中，后者随后从使用它们的应用程序引用，以确保它们发送和接收的消息与这些模式兼容。

Kafka 客户端应用程序可以在运行时从 Apicurio Registry 中推送或拉取其模式。

有关使用带有 Apache Kafka 的 Streams 的 Apicurio Registry 的红帽构建的 Apicurio Registry 的更多信息，请参阅 红帽构建的 Apicurio Registry 的产品文档。

## 12.5. 红帽构建的 APACHE CAMEL K

红帽构建的 Apache Camel K 是一个轻量级集成框架，从 Apache Camel K 构建，它在 OpenShift 上的云中原生运行。Camel K 支持无服务器集成，允许开发和部署集成任务，而无需管理底层基础架构。您可以使用 Camel K 构建并将事件驱动的应用程序与 Apache Kafka 环境的 Streams 集成。对于在不同系统或数据库之间需要实时数据同步的情况，Camel K 可用于捕获和转换事件的变化，并将其发送到 Apache Kafka 的流，以便将 Apache Kafka 分发到其他系统。

有关使用带有 Apache Kafka 的 Camel K 的 Camel K 的更多信息，请参阅 红帽构建的 Apache Camel K 产品文档。

**其他资源**

- Red Hat build of Keycloak 支持的配置
- Red Hat Single Sign-On 支持的配置(maintenance)
- Red Hat 3scale API Management Platform 支持的配置
- Red Hat build of Debezium 支持的配置
- Red Hat build of Apicurio Registry 支持的配置
- Red Hat build of Apache Camel K 支持的配置

*更新于 2024-06-25*