Red Hat OpenShift Serverless 1.30

Eventing

如何在 OpenShift Serverless 中使用事件驱动的架构

Last Updated: 2023-09-20

Red Hat OpenShift Serverless 1.30 Eventing

如何在 OpenShift Serverless 中使用事件驱动的架构

法律通告

摘要

本文档提供有关 Eventing 功能的信息，如事件源和接收器、代理、触发器、频道和订阅。

目录

第 1 章 KNATIVE EVENTING

OpenShift Container Platform 上的 Knative Eventing 可让开发人员使用 事件驱动的架构和无服务器应用程序。事件驱动的体系结构是基于事件和事件用户间分离关系的概念。

事件生成者创建事件，事件 *sink*、或消费者接收事件。Knative Eventing 使用标准 HTTP POST 请求来发送和接收事件制作者和 sink 之间的事件。这些事件符合 CloudEvents 规范，它允许在任何编程语言中创建、解析、发送和接收事件。

1.1. KNATIVE EVENTING 用例：

Knative Eventing 支持以下用例：

在不创建消费者的情况下发布事件

您可以将事件作为 HTTP POST 发送到代理，并使用绑定分离生成事件的应用程序的目标配置。

在不创建发布程序的情况下消费事件

您可以使用 Trigger 来根据事件属性消费来自代理的事件。应用程序以 HTTP POST 的形式接收事件。

要启用多种 sink 类型的交付，Knative Eventing 会定义以下通用接口，这些接口可由多个 Kubernetes 资源实现：

可寻址的资源

能够接收和确认通过 HTTP 发送的事件到 Event 的 **status.address.url** 字段中定义的地址。Kubernetes **Service** 资源也满足可寻址的接口。

可调用的资源

能够通过 HTTP 接收事件并转换它，并在 HTTP 响应有效负载中返回 **0** 或 **1** 新事件。这些返回的事件可能会象处理外部事件源中的事件一样进一步处理。

第 2 章 事件源

2.1. 事件源

Knative *事件源*可以是生成或导入云事件的任何 Kubernetes 对象，并将这些事件转发到另一个端点，称为*接收器（sink）*。事件源对于开发对事件做出反应的分布式系统至关重要。

您可以使用 OpenShift Container Platform Web 控制台、Knative (**kn**) CLI 或应用 YAML 文件的 **Developer** 视角创建和管理 Knative 事件源。

目前，OpenShift Serverless 支持以下事件源类型：

API 服务器源

将 Kubernetes API 服务器事件引入 Knative。每次创建、更新或删除 Kubernetes 资源时，API 服务器源会发送一个新事件。

Ping 源

根据指定的 cron 计划生成带有固定有效负载的事件。

Kafka 事件源

将 Apache Kafka 集群连接到接收器作为事件源。

您还可以创建自定义事件源。

2.2. ADMINISTRATOR 视角中的事件源

事件源对于开发对事件做出反应的分布式系统至关重要。

2.2.1. 使用 Administrator 视角创建事件源

Knative *事件源*可以是生成或导入云事件的任何 Kubernetes 对象，并将这些事件转发到另一个端点，称为*接收器（sink）*。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 您已登录到 Web 控制台，且处于 **Administrator** 视角。
- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Administrator** 视角中，导航到 **Serverless → Eventing**。
2. 在 **Create** 列表中，选择 **Event Source**。您将被定向到 **Event Sources** 页面。
3. 选择您要创建的事件源类型。

2.3. 创建 API 服务器源

API 服务器源是一个事件源，可用于将事件接收器（sink），如 Knative 服务，连接到 Kubernetes API 服务器。API 服务器源监视 Kubernetes 事件并将其转发到 Knative Eventing 代理。

2.3.1. 使用 Web 控制台创建 API 服务器源

在集群中安装 Knative Eventing 后，您可以使用 web 控制台创建 API 服务器源。使用 OpenShift Container Platform Web 控制台提供了一个简化且直观的用户界面来创建事件源。

先决条件

- 已登陆到 OpenShift Container Platform Web 控制台。
- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI(**oc**)。

流程

如果要重新使用现有服务帐户，您可以修改现有的 **ServiceAccount** 资源，使其包含所需的权限，而不是创建新资源。

1. 以 YAML 文件形式,为事件源创建服务帐户、角色和角色绑定：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: events-sa
  namespace: default 1

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: event-watcher
  namespace: default 2
rules:
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - get
      - list
      - watch

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: k8s-ra-event-watcher
  namespace: default 3
```

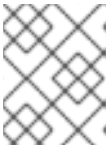
```
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: event-watcher
subjects:
  - kind: ServiceAccount
    name: events-sa
    namespace: default 4
```

1 2 3 4 将这个命名空间更改为已选择安装事件源的命名空间。

2. 应用 YAML 文件：

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

3. 在 **Developer** 视角中，导航到 **+Add → Event Source**。此时会显示 **Event Sources** 页面。
4. 可选：如果您的事件源有多个供应商，请从 **Providers** 列表中选择所需的供应商，以过滤供应商的可用事件源。
5. 选择 **ApiServerSource**，然后点 **Create Event Source**。此时会显示 **Create Event Source** 页面。
6. 使用 **Form view** 或 **YAML view** 配置 **ApiServerSource** 设置：



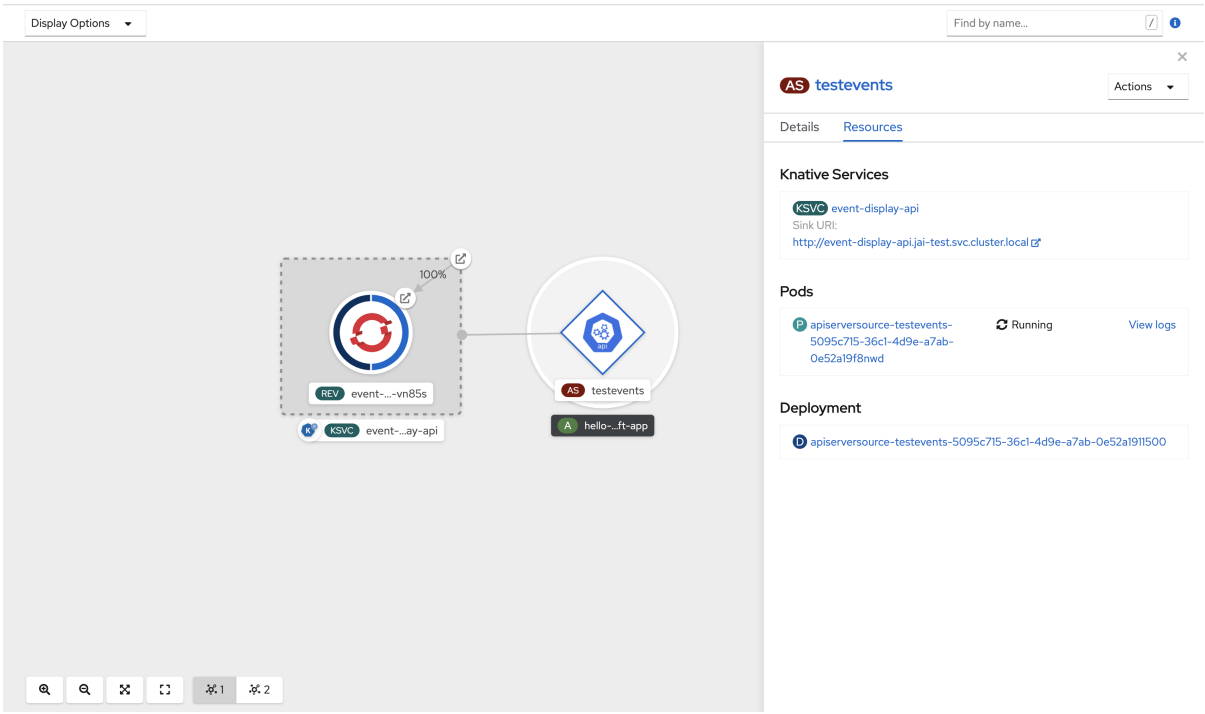
注意

您可以在 **Form view** 和 **YAML view** 间进行切换。在不同视图间切换时数据会被保留。

   a. 输入 **v1** 作为 **APIVERSION** 和 **Event** 作为 **KIND**。
   b. 为您创建的服务帐户选择 **Service Account Name**。
   c. 在 **Target** 部分中，选择您的事件 sink。这可以是 **Resource** 或一个 **URI** ：
      i. 选择 **Resource** 来使用频道、代理或服务作为事件源的事件 sink。
      ii. 选择 **URI** 来指定事件路由到的 Uniform Resource Identifier (URI)。
7. 点 **Create**。

验证

- 创建 API 服务器源后，通过在 **Topology** 视图中查看它来检查它是否已连接到事件 sink。

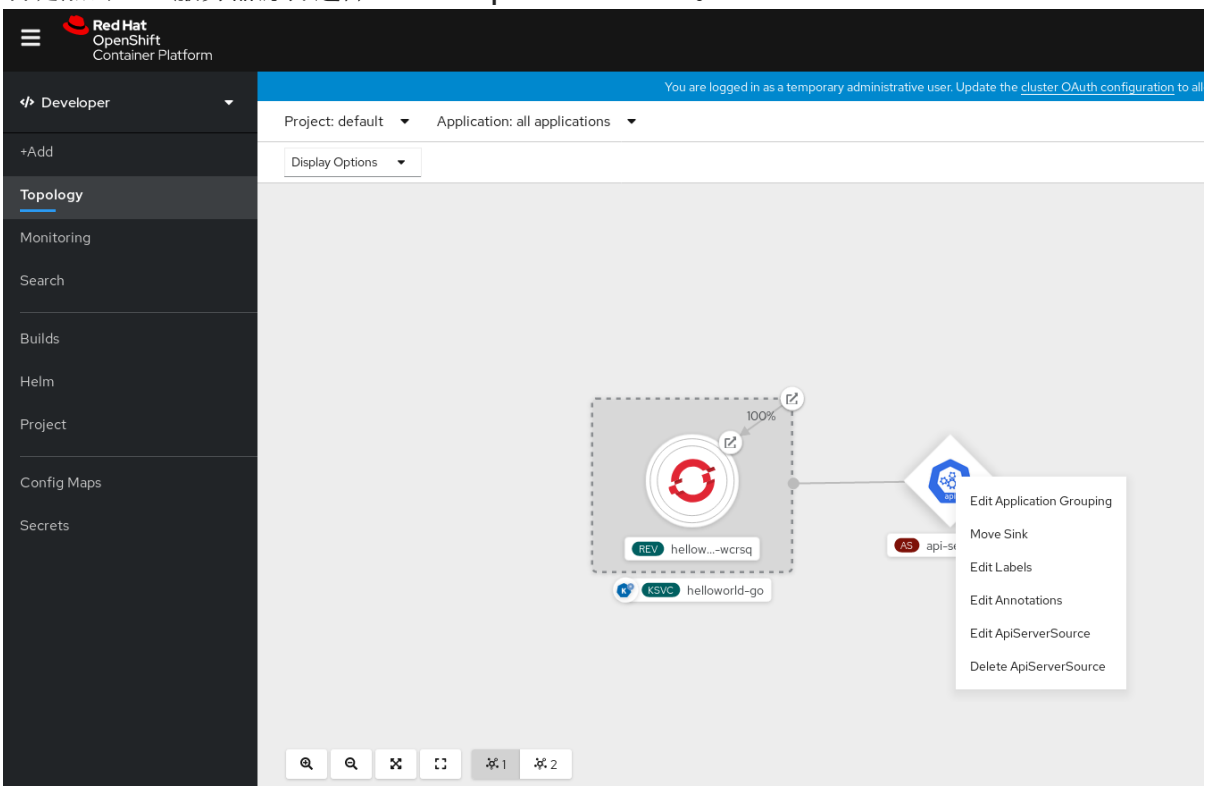


注意

如果使用 **URI sink**，您可以通过右键点击 URI sink → **Edit URI** 来修改 URI。

删除 API 服务器源

1. 导航到 **Topology** 视图。
2. 右键点击 API 服务器源并选择 **Delete ApiServerSource**。



2.3.2. 使用 Knative CLI 创建 API 服务器源

您可以使用 **kn source apiserver create** 命令，使用 **kn** CLI 创建 API 服务器源。使用 **kn** CLI 创建 API 服务器源可提供比直接修改 YAML 文件更精简且直观的用户界面。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI(**oc**)。
- 已安装 Knative (**kn**) CLI。

流程

如果要重新使用现有服务帐户，您可以修改现有的 **ServiceAccount** 资源，使其包含所需的权限，而不是创建新资源。

1. 以 YAML 文件形式,为事件源创建服务帐户、角色和角色绑定：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: events-sa
  namespace: default 1

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: event-watcher
  namespace: default 2
rules:
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - get
      - list
      - watch

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: k8s-ra-event-watcher
  namespace: default 3
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: event-watcher
subjects:
```

```
  - kind: ServiceAccount
    name: events-sa
    namespace: default 4
```

1 2 3 4 将这个命名空间更改为已选择安装事件源的命名空间。

2. 应用 YAML 文件：

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

3. 创建具有事件 sink 的 API 服务器源。在以下示例中，sink 是一个代理：

```
$ kn source apiserver create <event_source_name> --sink broker:<broker_name> --
resource "event:v1" --service-account <service_account_name> --mode Resource
```

4. 要检查 API 服务器源是否已正确设置，请创建一个 Knative 服务，在日志中转储传入的信息：

```
$ kn service create event-display --image quay.io/openshift-knative/showcase
```

5. 如果您使用代理作为事件 sink，请创建一个触发器将事件从 **default** 代理过滤到服务：

```
$ kn trigger create <trigger_name> --sink ksvc:event-display
```

6. 通过在 default 命名空间中启动 pod 来创建事件：

```
$ oc create deployment event-origin --image quay.io/openshift-knative/showcase
```

7. 通过检查以下命令生成的输出来检查是否正确映射了控制器：

```
$ kn source apiserver describe <source_name>
```

输出示例

```
Name:               mysource
Namespace:          default
Annotations:        sources.knative.dev/creator=developer,
sources.knative.dev/lastModifier=developer
Age:                3m
ServiceAccountName:  events-sa
Mode:               Resource
Sink:
  Name:       default
  Namespace:  default
  Kind:       Broker (eventing.knative.dev/v1)
Resources:
  Kind:       event (v1)
  Controller:  false
Conditions:
  OK TYPE                  AGE REASON
  ++ Ready                 3m
  ++ Deployed               3m
```

```
++ SinkProvided            3m
++ SufficientPermissions   3m
++ EventTypesProvided      3m
```

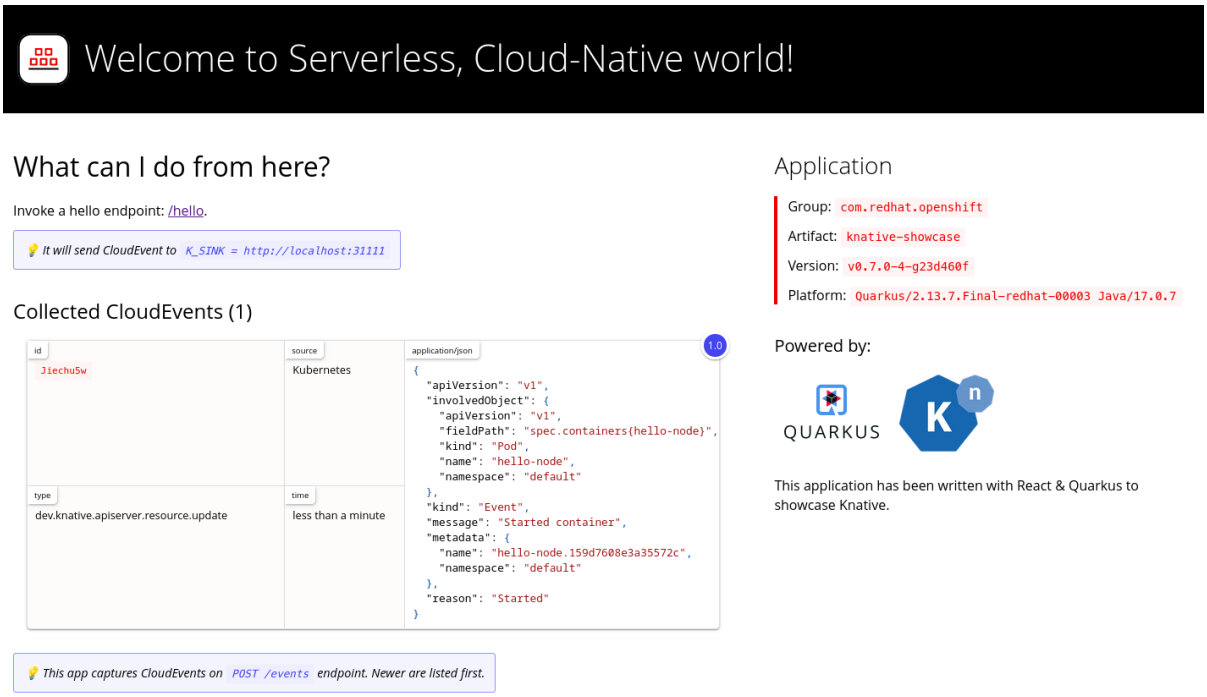
验证

要验证 Kubernetes 事件是否已发送到 Knative，请查看 event-display 日志或使用 Web 浏览器查看事件。

- 要在网页浏览器中查看事件，请使用以下命令返回的链接：

```
$ kn service describe event-display -o url
```

图 2.1. 浏览器页面示例



- 另外，要在终端中查看日志，请输入以下命令来查看 pod 的 event-display 日志：

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

输出示例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.apiserver.resource.update
  datacontenttype: application/json
  ...
Data,
  {
    "apiVersion": "v1",
    "involvedObject": {
      "apiVersion": "v1",
      "fieldPath": "spec.containers{event-origin}",
      "kind": "Pod",
```

```
    "name": "event-origin",
    "namespace": "default",
     .....
  },
  "kind": "Event",
  "message": "Started container",
  "metadata": {
    "name": "event-origin.159d7608e3a3572c",
    "namespace": "default",
    ....
  },
  "reason": "Started",
  ...
}
```

删除 API 服务器源

1. 删除触发器：

```
$ kn trigger delete <trigger_name>
```

2. 删除事件源：

```
$ kn source apiserver delete <source_name>
```

3. 删除服务帐户、集群角色和集群绑定：

```
$ oc delete -f authentication.yaml
```

2.3.2.1. Knative CLI sink 标记

当使用 Knative (**kn**) CLI 创建事件源时，您可以使用 **--sink** 标志指定事件从该资源发送到的接收器。sink 可以是任何可寻址或可调用的资源，可以从其他资源接收传入的事件。

以下示例创建使用服务 **http://event-display.svc.cluster.local** 的接收器绑定作为接收器：

使用 sink 标记的命令示例

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
  --subject "Job:batch/v1:app=heartbeat-cron" \
  --sink http://event-display.svc.cluster.local \ 1
  --ce-override "sink=bound"
```

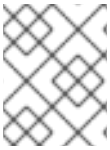
1 **http://event-display.svc.cluster.local** 中的 **svc** 确定接收器是一个 Knative 服务。其他默认的接收器前缀包括 **channel** 和 **broker**。

2.3.3. 使用 YAML 文件创建 API 服务器源

使用 YAML 文件创建 Knative 资源使用声明性 API，它允许您以可重复的方式描述事件源。要使用 YAML 创建 API 服务器源，您必须创建一个 YAML 文件来定义 **ApiServerSource** 对象，然后使用 **oc apply** 命令应用它。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您已在与 API 服务器源 YAML 文件中定义的相同的命名空间中创建 **default** 代理。
- 安装 OpenShift CLI (**oc**) 。



流程

如果要重新使用现有服务帐户，您可以修改现有的 **ServiceAccount** 资源，使其包含所需的权限，而不是创建新资源。

1. 以 YAML 文件形式,为事件源创建服务帐户、角色和角色绑定：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: events-sa
  namespace: default 1

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: event-watcher
  namespace: default 2
rules:
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - get
      - list
      - watch

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: k8s-ra-event-watcher
  namespace: default 3
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: event-watcher
subjects:
  - kind: ServiceAccount
    name: events-sa
    namespace: default 4
```

1 2 3 4 将这个命名空间更改为已选择安装事件源的命名空间。

2. 应用 YAML 文件：

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

3. 将 API 服务器源创建为 YAML 文件：

```
apiVersion: sources.knative.dev/v1alpha1
kind: ApiServerSource
metadata:
  name: testevents
spec:
  serviceAccountName: events-sa
  mode: Resource
  resources:
    - apiVersion: v1
      kind: Event
  sink:
    ref:
      apiVersion: eventing.knative.dev/v1
      kind: Broker
      name: default
```

4. 应用 **ApiServerSource** YAML 文件：

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

5. 要检查 API 服务器源是否已正确设置，请创建一个 Knative 服务作为 YAML 文件，在日志中转储传入的信息：

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
  namespace: default
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/showcase
```

6. 应用 **Service** YAML 文件：

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

7. 创建一个 **Trigger** 对象作为一个 YAML 文件，该文件将事件从 **default** 代理过滤到上一步中创建的服务：

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: event-display-trigger
```

```
  namespace: default
spec:
  broker: default
  subscriber:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

8. 应用 **Trigger** YAML 文件：

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

9. 通过在 default 命名空间中启动 pod 来创建事件：

```
$ oc create deployment event-origin --image=quay.io/openshift-knative/showcase
```

10. 输入以下命令并检查输出，检查是否正确映射了控制器：

```
$ oc get apiserversource.sources.knative.dev testevents -o yaml
```

输出示例

```
apiVersion: sources.knative.dev/v1alpha1
kind: ApiServerSource
metadata:
  annotations:
  creationTimestamp: "2020-04-07T17:24:54Z"
  generation: 1
  name: testevents
  namespace: default
  resourceVersion: "62868"
  selfLink:
/apis/sources.knative.dev/v1alpha1/namespaces/default/apiserversources/testevents2
  uid: 1603d863-bb06-4d1c-b371-f580b4db99fa
spec:
  mode: Resource
  resources:
  - apiVersion: v1
    controller: false
    controllerSelector:
      apiVersion: ""
      kind: ""
      name: ""
      uid: ""
    kind: Event
    labelSelector: {}
  serviceAccountName: events-sa
  sink:
    ref:
      apiVersion: eventing.knative.dev/v1
      kind: Broker
      name: default
```

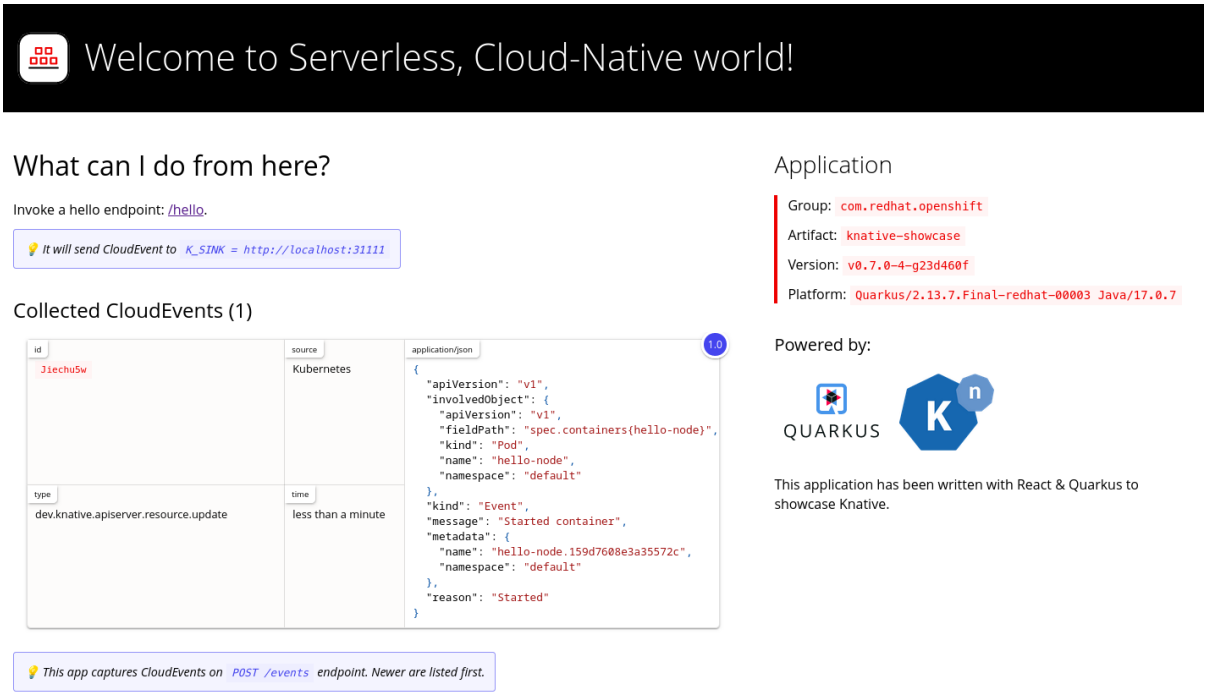
验证

要验证 Kubernetes 事件是否已发送到 Knative，您可以查看 event-display 日志或使用 Web 浏览器查看事件。

- 要在网页浏览器中查看事件，请使用以下命令返回的链接：

```
$ oc get ksvc event-display -o jsonpath='{.status.url}'
```

图 2.2. 浏览器页面示例



- 要在终端中查看日志，请输入以下命令来查看 pod 的 event-display 日志：

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

输出示例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.apiserver.resource.update
  datacontenttype: application/json
  ...
Data,
  {
    "apiVersion": "v1",
    "involvedObject": {
      "apiVersion": "v1",
      "fieldPath": "spec.containers{event-origin}",
      "kind": "Pod",
      "name": "event-origin",
      "namespace": "default",
       .....
    },
```

```
  "kind": "Event",
  "message": "Started container",
  "metadata": {
    "name": "event-origin.159d7608e3a3572c",
    "namespace": "default",
    ....
  },
  "reason": "Started",
  ...
}
```

删除 API 服务器源

1. 删除触发器：

```
$ oc delete -f trigger.yaml
```

2. 删除事件源：

```
$ oc delete -f k8s-events.yaml
```

3. 删除服务帐户、集群角色和集群绑定：

```
$ oc delete -f authentication.yaml
```

2.4. 创建 PING 源

ping 源是一个事件源，可用于定期向事件消费者发送带有恒定有效负载的 ping 事件。ping 源可以用来调度发送事件，类似于计时器。

2.4.1. 使用 Web 控制台创建 ping 源

在集群中安装 Knative Eventing 后，您可以使用 web 控制台创建 ping 源。使用 OpenShift Container Platform Web 控制台提供了一个简化且直观的用户界面来创建事件源。

先决条件

- 已登陆到 OpenShift Container Platform Web 控制台。
- 在集群中安装了 OpenShift Serverless Operator、Knative Serving 和 Knative Eventing。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 要验证 ping 源是否可以工作，请创建一个简单的 Knative 服务，在服务日志中转储传入的信息。
   a. 在 **Developer** 视角中，导航到 **+Add → YAML**。
   b. 复制 YAML 示例：

```
apiVersion: serving.knative.dev/v1
```

```
kind: Service
metadata:
  name: event-display
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/showcase
```

c. 点 **Create**。

2. 在与上一步中创建的服务相同的命名空间中创建一个 ping 源，或您要将事件发送到的任何其他接收器。

a. 在 **Developer** 视角中，导航到 **+Add → Event Source**。此时会显示 **Event Sources** 页面。

b. 可选：如果您的事件源有多个供应商，请从 **Providers** 列表中选择所需的供应商，以过滤供应商的可用事件源。

c. 选择 **Ping Source**，然后点击 **Create Event Source**。此时会显示 **Create Event Source** 页面。

注意

您可以使用 **Form view** 或 **YAML view** 配置 **PingSource** 设置，并可以在两者间切换。在不同视图间切换时数据会被保留。

d. 为 **Schedule** 输入一个值。在本例中，值为 ***/2 * * ***，它会创建一个 PingSource，每两分钟发送一条消息。

e. 可选：您可以为 **Data** 输入一个值，它是消息的有效负载。

f. 在 **Target** 部分中，选择您的事件 sink。这可以是 **Resource** 或一个 **URI** ：

i. 选择 **Resource** 来使用频道、代理或服务作为事件源的事件 sink。在本例中，上一步中创建的 **event-display** 服务用作目标 **资源**。

ii. 选择 **URI** 来指定事件路由到的 Uniform Resource Identifier (URI)。

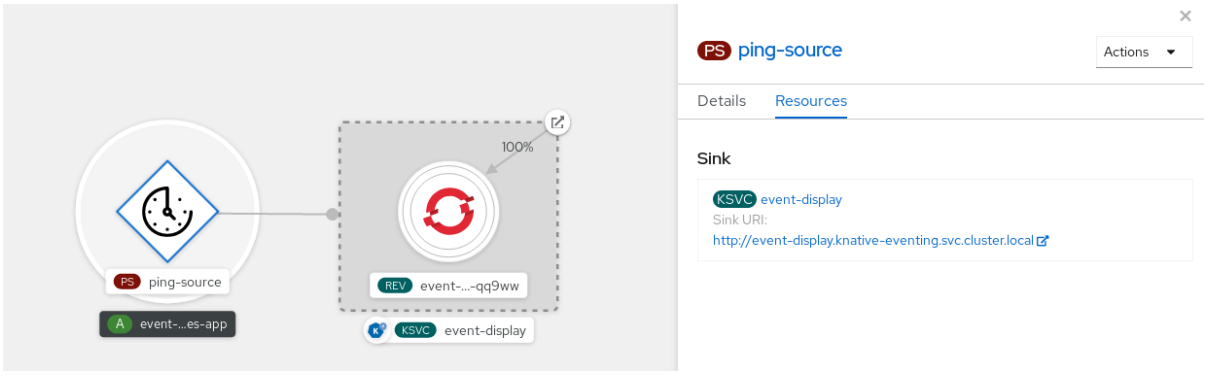
g. 点 **Create**。

验证

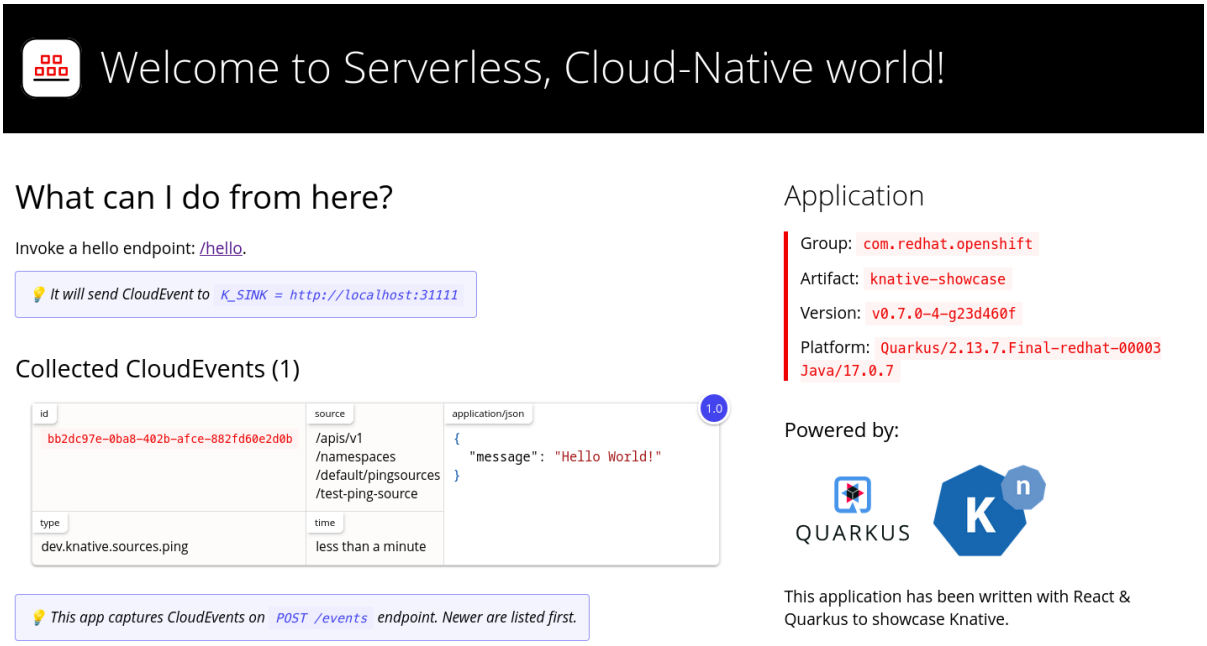
您可以通过查看 **Topology** 页面来验证 ping 源是否已创建并连接到接收器。

1. 在 **Developer** 视角中，导航到 **Topology**。

2. 查看 ping 源和接收器。



3. 在 Web 浏览器中查看 event-display 服务。您应该在 Web UI 中看到 ping 源事件。



删除 ping 源

1. 导航到 **Topology** 视图。
2. 右键单击 API 服务器源，再选择 **Delete Ping Source**。

2.4.2. 使用 Knative CLI 创建 ping 源

您可以使用 **kn source ping create** 命令，通过 Knative (**kn**) CLI 创建 ping 源。使用 Knative CLI 创建事件源提供了比直接修改 YAML 文件更精简且直观的用户界面。

先决条件

- 在集群中安装了 OpenShift Serverless Operator、Knative Serving 和 Knative Eventing。
- 已安装 Knative (**kn**) CLI。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 可选： 如果要使用此流程验证步骤，请安装 OpenShift CLI (**oc**) 。

流程

流程

1. 要验证 ping 源是否可以工作，请创建一个简单的 Knative 服务，在服务日志中转储传入的信息：

```
$ kn service create event-display \
    --image quay.io/openshift-knative/showcase
```

2. 对于您要请求的每一组 ping 事件，请在与事件消费者相同的命名空间中创建一个 ping 源：

```
$ kn source ping create test-ping-source \
    --schedule "*/2 * * * *" \
    --data '{"message": "Hello world!"}' \
    --sink ksvc:event-display
```

3. 输入以下命令并检查输出，检查是否正确映射了控制器：

```
$ kn source ping describe test-ping-source
```

输出示例

```
Name:         test-ping-source
Namespace:    default
Annotations:  sources.knative.dev/creator=developer,
sources.knative.dev/lastModifier=developer
Age:          15s
Schedule:     */2 * * * *
Data:         {"message": "Hello world!"}

Sink:
  Name:       event-display
  Namespace:  default
  Resource:   Service (serving.knative.dev/v1)

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                8s
  ++ Deployed             8s
  ++ SinkProvided         15s
  ++ ValidSchedule        15s
  ++ EventTypeProvided    15s
  ++ ResourcesCorrect     15s
```

验证

您可以通过查看 sink pod 的日志来验证 Kubernetes 事件是否已发送到 Knative 事件。

默认情况下，如果在 60 秒内都没有流量，Knative 服务会终止其 Pod。本指南中演示的示例创建了一个 ping 源，每 2 分钟发送一条消息，因此每个消息都应该在新创建的 pod 中观察到。

1. 查看新创建的 pod：

```
$ watch oc get pods
```

2. 使用 Ctrl+C 取消查看 pod，然后查看所创建 pod 的日志：

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

输出示例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.sources.ping
  source: /apis/v1/namespaces/default/pingsources/test-ping-source
  id: 99e4f4f6-08ff-4bff-acf1-47f61ded68c9
  time: 2020-04-07T16:16:00.000601161Z
  datacontenttype: application/json
Data,
  {
    "message": "Hello world!"
  }
```

删除 ping 源

- 删除 ping 源：

```
$ kn delete pingsources.sources.knative.dev <ping_source_name>
```

2.4.2.1. Knative CLI sink 标记

当使用 Knative (**kn**) CLI 创建事件源时，您可以使用 **--sink** 标志指定事件从该资源发送到的接收器。sink 可以是任何可寻址或可调用的资源，可以从其他资源接收传入的事件。

以下示例创建使用服务 **http://event-display.svc.cluster.local** 的接收器绑定作为接收器：

使用 sink 标记的命令示例

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
  --subject "Job:batch/v1:app=heartbeat-cron" \
  --sink http://event-display.svc.cluster.local \ 1
  --ce-override "sink=bound"
```

1 **http://event-display.svc.cluster.local** 中的 **svc** 确定接收器是一个 Knative 服务。其他默认的接收器前缀包括 **channel** 和 **broker**。

2.4.3. 使用 YAML 创建 ping 源

使用 YAML 文件创建 Knative 资源使用声明性 API，它允许您以可重复的方式描述事件源。要使用 YAML 创建无服务器 ping 源，您必须创建一个 YAML 文件来定义 **PingSource** 对象，然后使用 **oc apply** 来应用它。

PingSource 对象示例

```
apiVersion: sources.knative.dev/v1
kind: PingSource
```

```
metadata:
  name: test-ping-source
spec:
  schedule: "*/2 * * * *" 1
  data: '{"message": "Hello world!"}' 2
  sink: 3
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

1. 事件指定的调度使用 CRON 格式。
2. 事件消息正文以 JSON 编码的数据字符串表示。
3. 这些是事件消费者的详情。在这个示例中，我们使用名为 **event-display** 的 Knative 服务。

先决条件

- 在集群中安装了 OpenShift Serverless Operator、Knative Serving 和 Knative Eventing。
- 安装 OpenShift CLI (**oc**) 。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 要验证 ping 源是否可以工作，请创建一个简单的 Knative 服务，在服务日志中转储传入的信息。
   a. 创建服务 YAML 文件：

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/showcase
```

   b. 创建服务：

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

2. 对于您要请求的每一组 ping 事件，请在与事件消费者相同的命名空间中创建一个 ping 源。
   a. 为 ping 源创建 YAML 文件：

```
apiVersion: sources.knative.dev/v1
kind: PingSource
metadata:
  name: test-ping-source
```

```
  spec:
    schedule: "*/2 * * * *"
    data: '{"message": "Hello world!"}'
    sink:
      ref:
        apiVersion: serving.knative.dev/v1
        kind: Service
        name: event-display
```

b. 创建 ping 源：

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

3. 输入以下命令检查是否正确映射了控制器：

```
$ oc get pingsource.sources.knative.dev <ping_source_name> -oyaml
```

输出示例

```
apiVersion: sources.knative.dev/v1
kind: PingSource
metadata:
  annotations:
    sources.knative.dev/creator: developer
    sources.knative.dev/lastModifier: developer
  creationTimestamp: "2020-04-07T16:11:14Z"
  generation: 1
  name: test-ping-source
  namespace: default
  resourceVersion: "55257"
  selfLink: /apis/sources.knative.dev/v1/namespaces/default/pingsources/test-ping-source
  uid: 3d80d50b-f8c7-4c1b-99f7-3ec00e0a8164
spec:
  data: '{ value: "hello" }'
  schedule: '*/2 * * * *'
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
      namespace: default
```

验证

您可以通过查看 sink pod 的日志来验证 Kubernetes 事件是否已发送到 Knative 事件。

默认情况下，如果在 60 秒内都没有流量，Knative 服务会终止其 Pod。本指南中演示的示例创建了一个 PingSource，每 2 分钟发送一条消息，因此每个消息都应该在新创建的 pod 中观察到。

1. 查看新创建的 pod：

```
$ watch oc get pods
```

2. 使用 Ctrl+C 取消查看 pod，然后查看所创建 pod 的日志：

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

输出示例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.sources.ping
  source: /apis/v1/namespaces/default/pingsources/test-ping-source
  id: 042ff529-240e-45ee-b40c-3a908129853e
  time: 2020-04-07T16:22:00.000791674Z
  datacontenttype: application/json
Data,
  {
    "message": "Hello world!"
  }
```

删除 ping 源

- 删除 ping 源：

```
$ oc delete -f <filename>
```

示例命令

```
$ oc delete -f ping-source.yaml
```

2.5. APACHE KAFKA 的源

您可以创建一个 Apache Kafka 源，从 Apache Kafka 集群中读取事件，并将这些事件传递给接收器。您可以使用 OpenShift Container Platform web 控制台、Knative (**kn**) CLI 或直接创建 **KafkaSource** 对象并使用 OpenShift CLI (**oc**) 创建 Kafka 源来应用它。

注意

请参阅有关 为 Apache Kafka 安装 Knative 代理 的文档。

2.5.1. 使用 Web 控制台创建 Apache Kafka 事件源

在集群中安装 Apache Kafka 的 Knative 代理实现后，您可以使用 Web 控制台创建 Apache Kafka 源。使用 OpenShift Container Platform Web 控制台提供了一个简化的用户界面来创建 Kafka 源。

先决条件

- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** 自定义资源已安装在集群中。
- 已登陆到 web 控制台。
- 您可以访问 Red Hat AMQ Streams（Kafka）集群，该集群会生成您要导入的 Kafka 信息。

- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

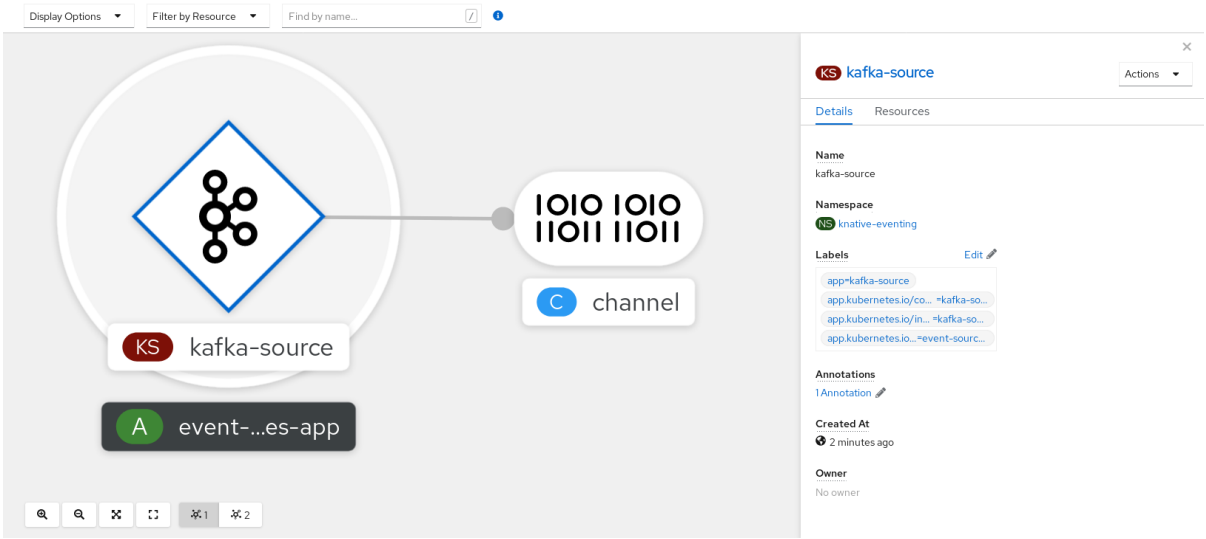
流程

1. 在 **Developer** 视角中，进入到 **+Add** 页面并选择 **Event Source**。
2. 在 **Event Sources** 页面中，在 **Type** 部分选择 **Kafka Source**。
3. 配置 **Kafka Source** 设置：
   a. 添加用逗号分开的 **Bootstrap 服务器**列表。
   b. 添加以逗号分隔的**标题**列表。
   c. 添加一个 **消费者组**。
   d. 为您创建的服务帐户选择 **Service Account Name**。
   e. 在 **Target** 部分中，选择您的事件 sink。这可以是 **Resource** 或一个 **URI** ：
      i. 选择 **Resource** 来使用频道、代理或服务作为事件源的事件 sink。
      ii. 选择 **URI** 来指定事件路由到的 Uniform Resource Identifier (URI)。
   f. 输入 Kafka 事件源的 **名称**。
4. 点 **Create**。

验证

您可以通过查看 **Topology** 页面来验证 Kafka 事件源是否已创建并连接到接收器。

1. 在 **Developer** 视角中，导航到 **Topology**。
2. 查看 Kafka 事件源和接收器。



2.5.2. 使用 Knative CLI 创建 Apache Kafka 事件源

您可以使用 **kn source kafka create** 命令，使用 Knative (**kn**) CLI 创建 Kafka 源。使用 Knative CLI 创建事件源提供了比直接修改 YAML 文件更精简且直观的用户界面。

先决条件

- OpenShift Serverless Operator、Knative Eventing、Knative Serving 和 **KnativeKafka** 自定义资源（CR）已安装在集群中。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您可以访问 Red Hat AMQ Streams（Kafka）集群，该集群会生成您要导入的 Kafka 信息。
- 已安装 Knative (**kn**) CLI。
- 可选：如果您想要使用此流程中的验证步骤，已安装 OpenShift CLI (**oc**)。

流程

1. 要验证 Kafka 事件源是否可以工作，请创建一个 Knative 服务，在服务日志中转储传入的事件：

```
$ kn service create event-display \
    --image quay.io/openshift-knative/showcase
```

2. 创建 **KafkaSource** CR：

```
$ kn source kafka create <kafka_source_name> \
    --servers <cluster_kafka_bootstrap>.kafka.svc:9092 \
    --topics <topic_name> --consumergroup my-consumer-group \
    --sink event-display
```

注意

将此命令中的占位符值替换为源名称、引导服务器和主题的值。

--servers、**--topics** 和 **--consumergroup** 选项指定到 Kafka 集群的连接参数。 **--consumergroup** 选项是可选的。

3. 可选：查看您创建的 **KafkaSource** CR 的详情：

```
$ kn source kafka describe <kafka_source_name>
```

输出示例

```
Name:              example-kafka-source
Namespace:         kafka
Age:               1h
BootstrapServers:  example-cluster-kafka-bootstrap.kafka.svc:9092
Topics:            example-topic
ConsumerGroup:     example-consumer-group

Sink:
  Name:       event-display
  Namespace:  default
  Resource:   Service (serving.knative.dev/v1)

Conditions:
```

```
OK TYPE           AGE REASON
++ Ready          1h
++ Deployed       1h
++ SinkProvided   1h
```

验证步骤

1. 触发 Kafka 实例将信息发送到主题：

```
$ oc -n kafka run kafka-producer \
  -ti --image=quay.io/strimzi/kafka:latest-kafka-2.7.0 --rm=true \
  --restart=Never -- bin/kafka-console-producer.sh \
  --broker-list <cluster_kafka_bootstrap>:9092 --topic my-topic
```

在提示符后输入信息。这个命令假设：

- Kafka 集群安装在 **kafka** 命名空间中。
- **KafkaSource** 对象已被配置为使用 **my-topic** 主题。

2. 通过查看日志来验证消息是否显示：

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

输出示例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.kafka.event
  source: /apis/v1/namespaces/default/kafkasources/example-kafka-source#example-topic
  subject: partition:46#0
  id: partition:46/offset:0
  time: 2021-03-10T11:21:49.4Z
Extensions,
  traceparent: 00-161ff3815727d8755848ec01c866d1cd-7ff3916c44334678-00
Data,
  Hello!
```

2.5.2.1. Knative CLI sink 标记

当使用 Knative (**kn**) CLI 创建事件源时，您可以使用 **--sink** 标志指定事件从该资源发送到的接收器。sink 可以是任何可寻址或可调用的资源，可以从其他资源接收传入的事件。

以下示例创建使用服务 **http://event-display.svc.cluster.local** 的接收器绑定作为接收器：

使用 sink 标记的命令示例

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
  --subject "Job:batch/v1:app=heartbeat-cron" \
  --sink http://event-display.svc.cluster.local \ 1
  --ce-override "sink=bound"
```

–

1. **http://event-display.svc.cluster.local** 中的 **svc** 确定接收器是一个 Knative 服务。其他默认的接收器前缀包括 **channel** 和 **broker**。

2.5.3. 使用 YAML 创建 Apache Kafka 事件源

使用 YAML 文件创建 Knative 资源使用声明性 API，它允许您以声明性的方式描述应用程序，并以可重复的方式描述应用程序。要使用 YAML 创建 Kafka 源，您必须创建一个 YAML 文件来定义 **KafkaSource** 对象，然后使用 **oc apply** 命令应用它。

先决条件

- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** 自定义资源已安装在集群中。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您可以访问 Red Hat AMQ Streams（Kafka）集群，该集群会生成您要导入的 Kafka 信息。
- 安装 OpenShift CLI (**oc**) 。

流程

1. 创建 **KafkaSource** 对象作为 YAML 文件：

```
apiVersion: sources.knative.dev/v1beta1
kind: KafkaSource
metadata:
  name: <source_name>
spec:
  consumerGroup: <group_name> 1
  bootstrapServers:
  - <list_of_bootstrap_servers>
  topics:
  - <list_of_topics> 2
  sink:
  - <list_of_sinks> 3
```

1. 用户组是一组使用相同组群 ID 的用户，并消耗一个标题中的数据。
2. 主题提供数据存储的目的地。每个主题都被分成一个或多个分区。
3. sink 指定事件从源发送到的位置。

重要

仅支持 OpenShift Serverless 上的 **KafkaSource** 对象的 **v1beta1** API 版本。不要使用这个 API 的 **v1alpha1** 版本，因为这个版本现已弃用。

KafkaSource 对象示例

```
apiVersion: sources.knative.dev/v1beta1
kind: KafkaSource
metadata:
  name: kafka-source
spec:
  consumerGroup: knative-group
  bootstrapServers:
  - my-cluster-kafka-bootstrap.kafka:9092
  topics:
  - knative-demo-topic
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

2. 应用 **KafkaSource** YAML 文件：

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

验证

- 输入以下命令验证 Kafka 事件源是否已创建：

```
$ oc get pods
```

输出示例

```
NAME                                READY   STATUS    RESTARTS   AGE
kafkasource-kafka-source-5ca0248f-...   1/1     Running   0          13m
```

2.5.4. 为 Apache Kafka 源配置 SASL 身份验证

Apache Kafka 使用 *简单身份验证和安全层* (SASL) 进行身份验证。如果在集群中使用 SASL 身份验证，用户则必须向 Knative 提供凭证才能与 Kafka 集群通信，否则无法生成或消耗事件。

先决条件

- 在 OpenShift Container Platform 上具有集群或专用管理员权限。
- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** CR 已安装在 OpenShift Container Platform 集群中。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您有一个 Kafka 集群的用户名和密码。
- 您已选择使用 SASL 机制，例如 **PLAIN**、**SCRAM-SHA-256** 或 **SCRAM-SHA-512**。
- 如果启用了 TLS，您还需要 Kafka 集群的 **ca.crt** 证书文件。
- 已安装 OpenShift (**oc**) CLI。

流程

1. 在所选命名空间中创建证书文件作为 secret：

```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
  --from-file=ca.crt=caroot.pem \
  --from-literal=password="SecretPassword" \
  --from-literal=saslType="SCRAM-SHA-512" \ 1
  --from-literal=user="my-sasl-user"
```

1 SASL 类型可以是 **PLAIN**、**SCRAM-SHA-256** 或 **SCRAM-SHA-512**。

2. 创建或修改 Kafka 源，使其包含以下 **spec** 配置：

```
apiVersion: sources.knative.dev/v1beta1
kind: KafkaSource
metadata:
  name: example-source
spec:
...
  net:
    sasl:
      enable: true
      user:
        secretKeyRef:
          name: <kafka_auth_secret>
          key: user
      password:
        secretKeyRef:
          name: <kafka_auth_secret>
          key: password
      type:
        secretKeyRef:
          name: <kafka_auth_secret>
          key: saslType
    tls:
      enable: true
      caCert: 1
        secretKeyRef:
          name: <kafka_auth_secret>
          key: ca.crt
...
```

1 如果您使用公共云 Kafka 服务，则不需要 **caCert** spec。

2.6. 自定义事件源

如果您需要从 Knative 中没有包含在 Knative 的事件制作者或发出没有 **CloudEvent** 格式的事件的制作者中入站事件，您可以通过创建自定义事件源来实现此目标。您可以使用以下方法之一创建自定义事件源：

- 通过创建接收器绑定，将 **PodSpecable** 对象用作事件源。
- 通过创建容器源，将容器用作事件源。

2.6.1. 接收器（sink）绑定

SinkBinding 对象支持将事件产品与交付寻址分离。接收器绑定用于将 *事件制作者* 连接到事件消费者（*sink*）。event producer 是一个 Kubernetes 资源，用于嵌入 **PodSpec** 模板并生成事件。sink 是一个可寻址的 Kubernetes 对象，可以接收事件。

SinkBinding 对象将环境变量注入到 sink 的 **PodTemplateSpec** 中，这意味着应用程序代码不需要直接与 Kubernetes API 交互来定位事件目的地。这些环境变量如下：

K_SINK

解析 sink 的 URL。

K_CE_OVERRIDES

指定出站事件覆盖的 JSON 对象。

注意

SinkBinding 对象目前不支持服务的自定义修订名称。

2.6.1.1. 使用 YAML 创建接收器绑定

使用 YAML 文件创建 Knative 资源使用声明性 API，它允许您以可重复的方式描述事件源。要使用 YAML 创建接收器绑定，您必须创建一个 YAML 文件来定义 **SinkBinding** 对象，然后使用 **oc apply** 命令应用它。

先决条件

- 在集群中安装了 OpenShift Serverless Operator、Knative Serving 和 Knative Eventing。
- 安装 OpenShift CLI (**oc**) 。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 要检查接收器绑定是否已正确设置，请创建一个 Knative 事件显示服务或事件接收器，在日志中转储传入的信息。

   a. 创建服务 YAML 文件：

   服务 YAML 文件示例

   ```
   apiVersion: serving.knative.dev/v1
   kind: Service
   metadata:
     name: event-display
   spec:
     template:
       spec:
         containers:
           - image: quay.io/openshift-knative/showcase
   ```

   b. 创建服务：

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

2. 创建将事件定向到该服务的接收器绑定实例。

   a. 创建接收器绑定 YAML 文件：

   服务 YAML 文件示例

```
apiVersion: sources.knative.dev/v1alpha1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  subject:
    apiVersion: batch/v1
    kind: Job 1
    selector:
      matchLabels:
        app: heartbeat-cron

  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

   1 在本例中，任何具有标签 **app: heartbeat-cron** 的作业都将被绑定到事件 sink。

   b. 创建接收器绑定：

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

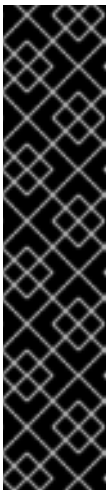
3. 创建 **CronJob** 对象。

   a. 创建 cron 任务 YAML 文件：

   Cron Job YAML 文件示例

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: heartbeat-cron
spec:
  # Run every minute
  schedule: "* * * * *"
  jobTemplate:
    metadata:
      labels:
        app: heartbeat-cron
        bindings.knative.dev/include: "true"
    spec:
      template:
        spec:
          restartPolicy: Never
```

```
      containers:
        - name: single-heartbeat
          image: quay.io/openshift-knative/heartbeats:latest
          args:
            - --period=1
          env:
            - name: ONE_SHOT
              value: "true"
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
```



重要

要使用接收器绑定，您必须手动在 Knative 资源中添加 **bindings.knative.dev/include=true** 标签。

例如，要将此标签添加到 **CronJob** 资源，请将以下行添加到 **Job** 资源 YAML 定义中：

```
jobTemplate:
  metadata:
    labels:
      app: heartbeat-cron
      bindings.knative.dev/include: "true"
```

b. 创建 cron job：

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

4. 输入以下命令并检查输出，检查是否正确映射了控制器：

```
$ oc get sinkbindings.sources.knative.dev bind-heartbeat -oyaml
```

输出示例

```
spec:
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
      namespace: default
  subject:
    apiVersion: batch/v1
    kind: Job
    namespace: default
```

```
    selector:
      matchLabels:
        app: heartbeat-cron
```

验证

您可以通过查看消息 dumper 功能日志，来验证 Kubernetes 事件是否已发送到 Knative 事件。

1. 输入命令：

```
$ oc get pods
```

2. 输入命令：

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

输出示例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.eventing.samples.heartbeat
  source: https://knative.dev/eventing-contrib/cmd/heartbeats/#event-test/mypod
  id: 2b72d7bf-c38f-4a98-a433-608fbcdd2596
  time: 2019-10-18T15:23:20.809775386Z
  contenttype: application/json
Extensions,
  beats: true
  heart: yes
  the: 42
Data,
  {
    "id": 1,
    "label": ""
  }
```

2.6.1.2. 使用 Knative CLI 创建接收器绑定

您可以使用 **kn source binding create** 命令通过 Knative (**kn**) CLI 创建接收器绑定。使用 Knative CLI 创建事件源提供了比直接修改 YAML 文件更精简且直观的用户界面。

先决条件

- 在集群中安装了 OpenShift Serverless Operator、Knative Serving 和 Knative Eventing。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 安装 Knative (**kn**) CLI。
- 安装 OpenShift CLI (**oc**) 。

注意

以下操作过程要求您创建 YAML 文件。

如果更改了示例中使用的 YAML 文件的名称，则需要更新对应的 CLI 命令。

流程

1. 要检查接收器绑定是否已正确设置，请创建一个 Knative 事件显示服务或事件 sink，在日志中转储传入的信息：

```
$ kn service create event-display --image quay.io/openshift-knative/showcase
```

2. 创建将事件定向到该服务的接收器绑定实例：

```
$ kn source binding create bind-heartbeat --subject Job:batch/v1:app=heartbeat-cron --sink ksvc:event-display
```

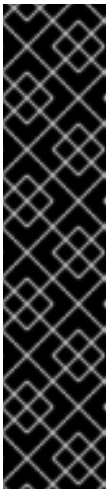
3. 创建 **CronJob** 对象。

   a. 创建 cron 任务 YAML 文件：

Cron Job YAML 文件示例

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: heartbeat-cron
spec:
  # Run every minute
  schedule: "* * * * *"
  jobTemplate:
    metadata:
      labels:
        app: heartbeat-cron
        bindings.knative.dev/include: "true"
    spec:
      template:
        spec:
          restartPolicy: Never
          containers:
            - name: single-heartbeat
              image: quay.io/openshift-knative/heartbeats:latest
              args:
                - --period=1
              env:
                - name: ONE_SHOT
                  value: "true"
                - name: POD_NAME
                  valueFrom:
                    fieldRef:
                      fieldPath: metadata.name
                - name: POD_NAMESPACE
```

```
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
```



重要

要使用接收器绑定，您必须手动在 Knative CR 中添加 **bindings.knative.dev/include=true** 标签。

例如，要将此标签添加到 **CronJob** CR，请将以下行添加到 **Job** CR YAML 定义中：

```
jobTemplate:
  metadata:
    labels:
      app: heartbeat-cron
      bindings.knative.dev/include: "true"
```

b. 创建 cron job：

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

4. 输入以下命令并检查输出，检查是否正确映射了控制器：

```
$ kn source binding describe bind-heartbeat
```

输出示例

```
Name:         bind-heartbeat
Namespace:    demo-2
Annotations:  sources.knative.dev/creator=minikube-user,
sources.knative.dev/lastModifier=minikub ...
Age:          2m
Subject:
  Resource:   job (batch/v1)
  Selector:
    app:      heartbeat-cron
Sink:
  Name:       event-display
  Resource:   Service (serving.knative.dev/v1)

Conditions:
  OK TYPE     AGE REASON
  ++ Ready    2m
```

验证

您可以通过查看消息 dumper 功能日志，来验证 Kubernetes 事件是否已发送到 Knative 事件。

- 您可以输入以下命令来查看消息转储程序功能日志：

```
$ oc get pods
```

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

输出示例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.eventing.samples.heartbeat
  source: https://knative.dev/eventing-contrib/cmd/heartbeats/#event-test/mypod
  id: 2b72d7bf-c38f-4a98-a433-608fbcdd2596
  time: 2019-10-18T15:23:20.809775386Z
  contenttype: application/json
Extensions,
  beats: true
  heart: yes
  the: 42
Data,
  {
    "id": 1,
    "label": ""
  }
```

2.6.1.2.1. Knative CLI sink 标记

当使用 Knative (**kn**) CLI 创建事件源时，您可以使用 **--sink** 标志指定事件从该资源发送到的接收器。sink 可以是任何可寻址或可调用的资源，可以从其他资源接收传入的事件。

以下示例创建使用服务 **http://event-display.svc.cluster.local** 的接收器绑定作为接收器：

使用 sink 标记的命令示例

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
  --subject "Job:batch/v1:app=heartbeat-cron" \
  --sink http://event-display.svc.cluster.local \ 1
  --ce-override "sink=bound"
```

1 **http://event-display.svc.cluster.local** 中的 **svc** 确定接收器是一个 Knative 服务。其他默认的接收器前缀包括 **channel** 和 **broker**。

2.6.1.3. 使用 Web 控制台创建接收器绑定

在集群中安装 Knative Eventing 后，您可以使用 web 控制台创建接收器绑定。使用 OpenShift Container Platform Web 控制台提供了一个简化且直观的用户界面来创建事件源。

先决条件

- 已登陆到 OpenShift Container Platform Web 控制台。
- OpenShift Serverless Operator、Knative Serving 和 Knative Eventing 已在 OpenShift Container Platform 集群中安装。

- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 创建 Knative 服务以用作接收器：
   a. 在 **Developer** 视角中，导航到 **+Add → YAML**。
   b. 复制 YAML 示例：

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/showcase
```

   c. 点 **Create**。
2. 创建用作事件源的 **CronJob** 资源，并每分钟发送一个事件。
   a. 在 **Developer** 视角中，导航到 **+Add → YAML**。
   b. 复制 YAML 示例：

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: heartbeat-cron
spec:
  # Run every minute
  schedule: "*/1 * * * *"
  jobTemplate:
    metadata:
      labels:
        app: heartbeat-cron
        bindings.knative.dev/include: true (1)
    spec:
      template:
        spec:
          restartPolicy: Never
          containers:
            - name: single-heartbeat
              image: quay.io/openshift-knative/heartbeats
              args:
              - --period=1
              env:
                - name: ONE_SHOT
                  value: "true"
                - name: POD_NAME
                  valueFrom:
                    fieldRef:
```

```
          fieldPath: metadata.name
      - name: POD_NAMESPACE
        valueFrom:
          fieldRef:
            fieldPath: metadata.namespace
```

1 确保包含 **bindings.knative.dev/include: true** 标签。OpenShift Serverless 的默认命名空间选择行为使用包含模式。

c. 点 **Create**。

3. 在与上一步中创建的服务相同的命名空间中创建接收器绑定，或您要将事件发送到的任何其他接收器。

a. 在 **Developer** 视角中，导航到 **+Add → Event Source**。此时会显示 **Event Sources** 页面。

b. 可选：如果您的事件源有多个供应商，请从 **Providers** 列表中选择所需的供应商，以过滤供应商的可用事件源。

c. 选择 **Sink Binding**，然后单击 **Create Event Source**。此时会显示 **Create Event Source** 页面。

注意

您可以使用 **Form view** 或 **YAML view** 配置 **Sink Binding** 设置，并可以在两者间切换。在不同视图间切换时数据会被保留。

d. 在 **apiVersion** 字段中，输入 **batch/v1**。

e. 在 **Kind** 字段中，输入 **Job**。

注意

OpenShift Serverless sink 绑定不支持 **CronJob** kind，因此 **Kind** 字段必须以 cron 任务创建的 **Job** 对象为目标，而不是 cron 作业对象本身。

f. 在 **Target** 部分中，选择您的事件 sink。这可以是 **Resource** 或一个 **URI** ：

i. 选择 **Resource** 来使用频道、代理或服务作为事件源的事件 sink。在本例中，上一步中创建的 **event-display** 服务用作目标 **资源**。

ii. 选择 **URI** 来指定事件路由到的 Uniform Resource Identifier (URI)。

g. 在 **Match labels** 部分：

i. 在 **Name** 字段中输入 **app**。

ii. 在 **Value** 字段中输入 **heartbeat-cron**。

注意

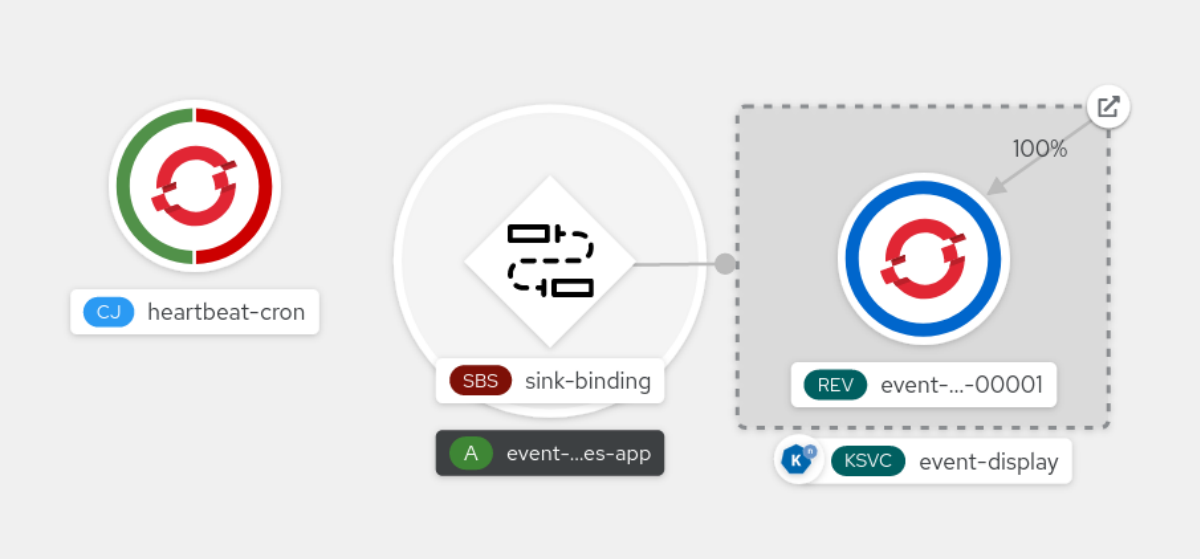
使用带有接收器绑定的 cron 任务时，需要标签选择器，而不是资源名称。这是因为，Cron Job 创建的作业没有可预测的名称，并在名称中包含随机生成的字符串。例如，**hearthbeat-cron-1cc23f**.

h. 点 **Create**。

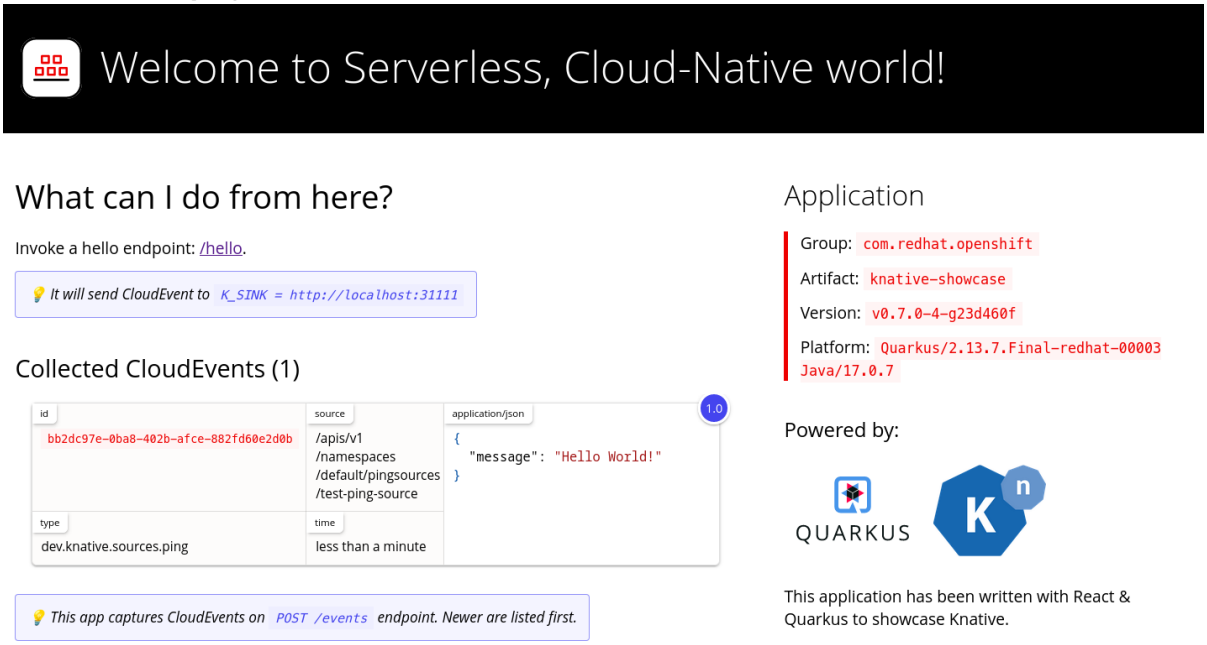
验证

您可以通过查看 **Topology** 页面和 pod 日志来验证接收器绑定、接收器和 cron 任务是否已创建并正常工作。

1. 在 **Developer** 视角中，导航到 **Topology**。
2. 查看接收器绑定、接收器和心跳 cron 任务。



3. 观察在添加了接收器绑定后 cron 任务正在注册成功的作业。这意味着接收器绑定成功重新配置由 cron 任务创建的作业。
4. 浏览 **event-display** 服务，以查看 heartbeats cron 作业生成的事件。



2.6.1.4. 接收器绑定引用

您可以通过创建接收器绑定，将 **PodSpecable** 对象用作事件源。您可以在创建 **SinkBinding** 对象时配置多个参数。

SinkBinding 对象支持以下参数：

字段	描述	必需或可选
apiVersion	指定 API 版本，如 **sources.knative.dev/v1**。	必需
kind	将此资源对象标识为 **SinkBinding** 对象。	必需
metadata	指定唯一标识 **SinkBinding** 对象的元数据。例如，**名称**。	必需
spec	指定此 **SinkBinding** 对象的配置信息。	必需
spec.sink	对解析为 URI 作为 sink 的对象的引用。	必需
spec.subject	提及通过绑定实施来增强运行时合同的资源。	必需
spec.ceOverrides	定义覆盖来控制发送到 sink 的事件的输出格式和修改。	选填

2.6.1.4.1. 主题参数

Subject 参数引用通过绑定实施来增强运行时合同的资源。您可以为 **Subject** 定义配置多个字段。

Subject 定义支持以下字段：

字段	描述	必需或可选
apiVersion	引用的 API 版本。	必需
kind	引用的类型。	必需
namespace	引用的命名空间。如果省略，则默认为对象的命名空间。	选填
name	引用的名称。	如果配置 **选择器**，请不要使用。
selector	引用的选择器。	如果配置 **名称**，请不要使用。
selector.matchExpressions	标签选择器要求列表。	仅使用 **matchExpressions** 或 **matchLabels** 中的一个。

字段	描述	必需或可选
selector.matchExpressions.key	选择器应用到的标签键。	使用 **matchExpressions** 时需要此项。
selector.matchExpressions.operator	代表键与一组值的关系。有效的运算符为 **In**、**NotIn**、**Exists** 和 **DoesNotExist**。	使用 **matchExpressions** 时需要此项。
selector.matchExpressions.values	字符串值数组。如果 **operator** 参数值是 **In** 或 **NotIn**，则值数组必须是非空的。如果 **operator** 参数值是 **Exists** 或 **DoesNotExist**，则值数组必须为空。这个数组会在策略性合并补丁中被替换。	使用 **matchExpressions** 时需要此项。
selector.matchLabels	键值对映射.**matchLabels** 映射中的每个键值对等同于 **matchExpressions** 元素，其中 key 字段是 **matchLabels.<key>**，**Operator** 为 **In**，**values** 数组仅包含 **matchLabels.<value>**。	仅使用 **matchExpressions** 或 **matchLabels** 中的一个。

主题参数示例

根据以下 YAML，选择 **default** 命名空间中名为 **mysubject** 的 **Deployment** 对象：

```
apiVersion: sources.knative.dev/v1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  subject:
    apiVersion: apps/v1
    kind: Deployment
    namespace: default
    name: mysubject
  ...
```

根据以下 YAML，可以选择在 **default** 命名空间中带有 **working=example** 标签的 **Job** 对象：

```
apiVersion: sources.knative.dev/v1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  subject:
    apiVersion: batch/v1
    kind: Job
    namespace: default
    selector:
```

```
      matchLabels:
        working: example
  ...
```

根据以下 YAML，可以选择在 **default** 命名空间中带有标签 **working=example** 或 **working=sample** 的 **Pod** 对象：

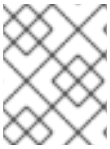
```
apiVersion: sources.knative.dev/v1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  subject:
    apiVersion: v1
    kind: Pod
    namespace: default
    selector:
      - matchExpression:
        key: working
        operator: In
        values:
          - example
          - sample
  ...
```

2.6.1.4.2. CloudEvent 覆盖

ceOverrides 定义提供覆盖控制发送到 sink 的 CloudEvent 输出格式和修改。您可以为 **ceOverrides** 定义配置多个字段。

ceOverrides 定义支持以下字段：

字段	描述	必需或可选
extensions	指定在出站事件中添加或覆盖哪些属性。每个 **extensions** 键值对在事件上作为属性扩展进行独立设置。	选填



注意

仅允许有效的 **CloudEvent** 属性名称作为扩展。您无法从扩展覆盖配置设置 spec 定义的属性。例如，您无法修改 **type** 属性。

CloudEvent Overrides 示例

```
apiVersion: sources.knative.dev/v1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  ...
```

```
  ceOverrides:
    extensions:
      extra: this is an extra attribute
      additional: 42
```

这会在 **主题** 上设置 **K_CE_OVERRIDES** 环境变量：

输出示例

```
{ "extensions": { "extra": "this is an extra attribute", "additional": "42" } }
```

2.6.1.4.3. include 标签

要使用接收器绑定，您需要为资源或包含资源的命名空间分配 **bindings.knative.dev/include: "true"** 标签。如果资源定义不包括该标签，集群管理员可以通过运行以下命令将它附加到命名空间：

```
$ oc label namespace <namespace> bindings.knative.dev/include=true
```

2.6.1.5. 将 Service Mesh 与 SinkBinding 集成

先决条件

- 您已将 Service Mesh 与 OpenShift Serverless 集成。

流程

1. 在作为 **Service MeshMemberRoll** 的成员的命名空间中创建一个 Service。

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
  namespace: <namespace> 1
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true" 2
        sidecar.istio.io/rewriteAppHTTPProbers: "true"
    spec:
      containers:
      - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

1. 作为 **ServiceMeshMemberRoll** 的成员的命名空间。
2. 将 Service Mesh sidecar 注入 Knative 服务 pod。

2. 应用 **Service** 资源。

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

3. 创建 **SinkBinding**。

```
apiVersion: sources.knative.dev/v1
kind: SinkBinding
metadata:
  name: bind-heartbeat
  namespace: <namespace> 1
spec:
  subject:
    apiVersion: batch/v1
    kind: Job 2
    selector:
      matchLabels:
        app: heartbeat-cron

  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

1 作为 **ServiceMeshMemberRoll** 的成员的命名空间。

2 在本例中，任何带有标签 **app: heartbeat-cron** 的作业都绑定到事件 sink。

4. 应用 **SinkBinding** 资源。

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

5. 创建 **CronJob** ：

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: heartbeat-cron
  namespace: <namespace> 1
spec:
  # Run every minute
  schedule: "* * * * *"
  jobTemplate:
    metadata:
      labels:
        app: heartbeat-cron
        bindings.knative.dev/include: "true"
    spec:
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "true" 2
            sidecar.istio.io/rewriteAppHTTPProbers: "true"
        spec:
          restartPolicy: Never
          containers:
            - name: single-heartbeat
              image: quay.io/openshift-knative/heartbeats:latest
              args:
```

```
        - --period=1
      env:
        - name: ONE_SHOT
          value: "true"
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
```

1. 作为 **ServiceMeshMemberRoll** 的成员的命名空间。
2. 将 Service Mesh sidecar 注入 **CronJob** pod。

6. 应用 **CronJob** 资源。

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

验证

要验证事件是否已发送到 Knative 事件 sink，请查看消息转储程序功能日志。

1. 输入以下命令：

```
$ oc get pods
```

2. 输入以下命令：

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

输出示例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.eventing.samples.heartbeat
  source: https://knative.dev/eventing/test/heartbeats/#event-test/mypod
  id: 2b72d7bf-c38f-4a98-a433-608fbcdd2596
  time: 2019-10-18T15:23:20.809775386Z
  contenttype: application/json
Extensions,
  beats: true
  heart: yes
  the: 42
Data,
  {
    "id": 1,
    "label": ""
  }
```

其他资源

- IntegratingService Mesh with OpenShift Serverless

2.6.2. 容器源

容器源创建容器镜像来生成事件并将事件发送到 sink。您可以通过创建容器镜像和使用您的镜像 URI 的 **ContainerSource** 对象，使用容器源创建自定义事件源。

2.6.2.1. 创建容器镜像的指南

两个环境变量由容器源控制器注入：**K_SINK** 和 **K_CE_OVERRIDES**。这些变量分别从 **sink** 和 **andceOverrides** spec 解析。事件发送到 **K_SINK** 环境变量中指定的 sink URI。该消息必须使用 **CloudEvent** HTTP 格式作为 **POST** 发送。

容器镜像示例

以下是心跳容器镜像的示例：

```
package main

import (
 "context"
 "encoding/json"
 "flag"
 "fmt"
 "log"
 "os"
 "strconv"
 "time"

 duckv1 "knative.dev/pkg/apis/duck/v1"

 cloudevents "github.com/cloudevents/sdk-go/v2"
 "github.com/kelseyhightower/envconfig"
)

type Heartbeat struct {
 Sequence int    `json:"id"`
 Label    string `json:"label"`
}

var (
 eventSource string
 eventType   string
 sink        string
 label       string
 periodStr   string
)

func init() {
 flag.StringVar(&eventSource, "eventSource", "", "the event-source (CloudEvents)")
 flag.StringVar(&eventType, "eventType", "dev.knative.eventing.samples.heartbeat", "the event-type
(CloudEvents)")
 flag.StringVar(&sink, "sink", "", "the host url to heartbeat to")
 flag.StringVar(&label, "label", "", "a special label")
```

```
 flag.StringVar(&periodStr, "period", "5", "the number of seconds between heartbeats")
}

type envConfig struct {
 // Sink URL where to send heartbeat cloud events
 Sink string `envconfig:"K_SINK"`

 // CEOverrides are the CloudEvents overrides to be applied to the outbound event.
 CEOverrides string `envconfig:"K_CE_OVERRIDES"`

 // Name of this pod.
 Name string `envconfig:"POD_NAME" required:"true"`

 // Namespace this pod exists in.
 Namespace string `envconfig:"POD_NAMESPACE" required:"true"`

 // Whether to run continuously or exit.
 OneShot bool `envconfig:"ONE_SHOT" default:"false"`
}

func main() {
 flag.Parse()

 var env envConfig
 if err := envconfig.Process("", &env); err != nil {
  log.Printf("[ERROR] Failed to process env var: %s", err)
  os.Exit(1)
 }

 if env.Sink != "" {
  sink = env.Sink
 }

 var ceOverrides *duckv1.CloudEventOverrides
 if len(env.CEOverrides) > 0 {
  overrides := duckv1.CloudEventOverrides{}
  err := json.Unmarshal([]byte(env.CEOverrides), &overrides)
  if err != nil {
   log.Printf("[ERROR] Unparseable CloudEvents overrides %s: %v", env.CEOverrides, err)
   os.Exit(1)
  }
  ceOverrides = &overrides
 }

 p, err := cloudevents.NewHTTP(cloudevents.WithTarget(sink))
 if err != nil {
  log.Fatalf("failed to create http protocol: %s", err.Error())
 }

 c, err := cloudevents.NewClient(p, cloudevents.WithUUIDs(), cloudevents.WithTimeNow())
 if err != nil {
  log.Fatalf("failed to create client: %s", err.Error())
 }

 var period time.Duration
 if p, err := strconv.Atoi(periodStr); err != nil {
```

```
  period = time.Duration(5) * time.Second
 } else {
  period = time.Duration(p) * time.Second
 }

 if eventSource == "" {
  eventSource = fmt.Sprintf("https://knative.dev/eventing-contrib/cmd/heartbeats/#%s/%s",
env.Namespace, env.Name)
  log.Printf("Heartbeats Source: %s", eventSource)
 }

 if len(label) > 0 && label[0] == '"' {
  label, _ = strconv.Unquote(label)
 }
 hb := &Heartbeat{
  Sequence: 0,
  Label:    label,
 }
 ticker := time.NewTicker(period)
 for {
  hb.Sequence++

  event := cloudevents.NewEvent("1.0")
  event.SetType(eventType)
  event.SetSource(eventSource)
  event.SetExtension("the", 42)
  event.SetExtension("heart", "yes")
  event.SetExtension("beats", true)

  if ceOverrides != nil && ceOverrides.Extensions != nil {
   for n, v := range ceOverrides.Extensions {
    event.SetExtension(n, v)
   }
  }

  if err := event.SetData(cloudevents.ApplicationJSON, hb); err != nil {
   log.Printf("failed to set cloudevents data: %s", err.Error())
  }

  log.Printf("sending cloudevent to %s", sink)
  if res := c.Send(context.Background(), event); !cloudevents.IsACK(res) {
   log.Printf("failed to send cloudevent: %v", res)
  }

  if env.OneShot {
   return
  }

  // Wait for next tick
  <-ticker.C
 }
}
```

以下是引用先前心跳容器镜像的容器源示例：

```
apiVersion: sources.knative.dev/v1
```

```
kind: ContainerSource
metadata:
  name: test-heartbeats
spec:
  template:
    spec:
      containers:
        # This corresponds to a heartbeats image URI that you have built and published
        - image: gcr.io/knative-releases/knative.dev/eventing/cmd/heartbeats
          name: heartbeats
          args:
            - --period=1
          env:
            - name: POD_NAME
              value: "example-pod"
            - name: POD_NAMESPACE
              value: "event-test"
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: showcase
...
```

2.6.2.2. 使用 Knative CLI 创建和管理容器源

您可以使用 **kn source container** 命令来使用 Knative (**kn**) 创建和管理容器源。使用 Knative CLI 创建事件源提供了比直接修改 YAML 文件更精简且直观的用户界面。

创建容器源

```
$ kn source container create <container_source_name> --image <image_uri> --sink <sink>
```

删除容器源

```
$ kn source container delete <container_source_name>
```

描述容器源

```
$ kn source container describe <container_source_name>
```

列出现有容器源

```
$ kn source container list
```

以 YAML 格式列出现有容器源

```
$ kn source container list -o yaml
```

更新容器源

此命令为现有容器源更新镜像 URI：

```
$ kn source container update <container_source_name> --image <image_uri>
```

2.6.2.3. 使用 Web 控制台创建容器源

在集群中安装 Knative Eventing 后，您可以使用 Web 控制台创建容器源。使用 OpenShift Container Platform Web 控制台提供了一个简化且直观的用户界面来创建事件源。

先决条件

- 已登陆到 OpenShift Container Platform Web 控制台。
- OpenShift Serverless Operator、Knative Serving 和 Knative Eventing 已在 OpenShift Container Platform 集群中安装。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 在 **Developer** 视角中，导航到 **+Add → Event Source**。此时会显示 **Event Sources** 页面。
2. 选择 **Container Source**，然后点 **Create Event Source**。此时会显示 **Create Event Source** 页面。
3. 使用 **Form view** 或 **YAML** 视图配置 **Container Source** 设置：

注意

您可以在 **Form view** 和 **YAML view** 间进行切换。在不同视图间切换时数据会被保留。

   a. 在 **Image** 字段中，输入您要在由容器源创建的容器中运行的镜像的 URI。
   b. 在 **Name** 字段中输入镜像的名称。
   c. 可选：在 **Arguments** 参数字段中，输入要传递给容器的任何参数。
   d. 可选：在 **Environment variables** 字段中，添加容器中要设置的任何环境变量。
   e. 在 **Target** 部分中，选择您的事件 sink。这可以是 **Resource** 或一个 **URI** ：
      i. 选择 **Resource** 来使用频道、代理或服务作为事件源的事件 sink。
      ii. 选择 **URI** 来指定事件路由到的 Uniform Resource Identifier (URI)。
4. 配置完容器源后，点 **Create**。

2.6.2.4. 容器源参考

您可以通过创建 **ContainerSource** 对象来使用容器作为事件源。您可以在创建 **ContainerSource** 对象时配置多个参数。

ContainerSource 对象支持以下字段：

字段	描述	必需或可选
apiVersion	指定 API 版本，如 **sources.knative.dev/v1**。	必需
kind	将此资源对象标识为 **ContainerSource** 对象。	必需
metadata	指定唯一标识 **ContainerSource** 对象的元数据。例如，**name**。	必需
spec	指定此 **ContainerSource** 对象的配置信息。	必需
spec.sink	对解析为 URI 作为 sink 的对象的引用。	必需
spec.template	**ContainerSource** 对象的 **template** 规格。	必需
spec.ceOverrides	定义覆盖来控制发送到 sink 的事件的输出格式和修改。	选填

模板参数示例

```
apiVersion: sources.knative.dev/v1
kind: ContainerSource
metadata:
  name: test-heartbeats
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/heartbeats:latest
          name: heartbeats
          args:
            - --period=1
          env:
            - name: POD_NAME
              value: "mypod"
            - name: POD_NAMESPACE
              value: "event-test"
  ...
```

2.6.2.4.1. CloudEvent 覆盖

ceOverrides 定义提供覆盖控制发送到 sink 的 CloudEvent 输出格式和修改。您可以为 **ceOverrides** 定义配置多个字段。

ceOverrides 定义支持以下字段：

字段	描述	必需或可选
extensions	指定在出站事件中添加或覆盖哪些属性。每个 **extensions** 键值对在事件上作为属性扩展进行独立设置。	选填

注意

仅允许有效的 **CloudEvent** 属性名称作为扩展。您无法从扩展覆盖配置设置 spec 定义的属性。例如，您无法修改 **type** 属性。

CloudEvent Overrides 示例

```
apiVersion: sources.knative.dev/v1
kind: ContainerSource
metadata:
  name: test-heartbeats
spec:
  ...
  ceOverrides:
    extensions:
      extra: this is an extra attribute
      additional: 42
```

这会在 **主题** 上设置 **K_CE_OVERRIDES** 环境变量：

输出示例

```
{ "extensions": { "extra": "this is an extra attribute", "additional": "42" } }
```

2.6.2.5. 将 Service Mesh 与 ContainerSource 集成

先决条件

- 您已将 Service Mesh 与 OpenShift Serverless 集成。

流程

1. 在作为 **Service MeshMemberRoll** 的成员的命名空间中创建一个 Service。

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
  namespace: <namespace> 1
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true" 2
```

```
      sidecar.istio.io/rewriteAppHTTPProbers: "true"
  spec:
    containers:
    - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

1. 作为 **ServiceMeshMemberRoll** 的成员的命名空间。
2. 将 Service Mesh sidecar 注入 Knative 服务 pod。

2. 应用 **Service** 资源。

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

3. 在作为 **ServiceMeshMemberRoll** 的成员的命名空间中创建一个 **ContainerSource**，将 sink 设置为 **event-display**。

```
apiVersion: sources.knative.dev/v1
kind: ContainerSource
metadata:
  name: test-heartbeats
  namespace: <namespace> 1
spec:
  template:
    metadata: 2
      annotations:
        sidecar.istio.io/inject: "true"
        sidecar.istio.io/rewriteAppHTTPProbers: "true"
    spec:
      containers:
        - image: quay.io/openshift-knative/heartbeats:latest
          name: heartbeats
          args:
            - --period=1s
          env:
            - name: POD_NAME
              value: "example-pod"
            - name: POD_NAMESPACE
              value: "event-test"
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

1. 命名空间是 **ServiceMeshMemberRoll** 的一部分。
2. 启用 Service Mesh 与 **ContainerSource**集成

4. 应用 **ContainerSource** 资源。

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

验证

要验证事件是否已发送到 Knative 事件 sink，请查看消息转储程序功能日志。

1. 输入以下命令：

```
$ oc get pods
```

2. 输入以下命令：

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

输出示例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.eventing.samples.heartbeat
  source: https://knative.dev/eventing/test/heartbeats/#event-test/mypod
  id: 2b72d7bf-c38f-4a98-a433-608fbcdd2596
  time: 2019-10-18T15:23:20.809775386Z
  contenttype: application/json
Extensions,
  beats: true
  heart: yes
  the: 42
Data,
  {
    "id": 1,
    "label": ""
  }
```

其他资源

- IntegratingService Mesh with OpenShift Serverless

2.7. 使用 DEVELOPER 视角将事件源连接到事件 SINK

当使用 OpenShift Container Platform Web 控制台创建事件源时，您可以指定事件从该源发送到的目标事件接收器。事件 sink 可以是任何可寻址或可调用的资源，可以从其他资源接收传入的事件。

2.7.1. 使用 Developer 视角将事件源连接到事件 sink

先决条件

- OpenShift Serverless Operator、Knative Serving 和 Knative Eventing 已在 OpenShift Container Platform 集群中安装。
- 已登陆到 web 控制台，且处于 **Developer** 视角。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您已创建了事件 sink，如 Knative 服务、频道或代理。

流程

1. 进入 **+Add → Event Source** 并选择您要创建的事件源类型，创建任何类型的事件源。
2. 在 **Create Event Source** 表单视图的 **Target** 部分，选择您的事件 sink。这可以是 **Resource** 或一个 **URI** ：
   a. 选择 **Resource** 来使用频道、代理或服务作为事件源的事件 sink。
   b. 选择 **URI** 来指定事件路由到的 Uniform Resource Identifier (URI)。
3. 点 **Create**。

验证

您可以通过查看 **Topology** 页面来验证事件源是否已创建并连接到 sink。

1. 在 **Developer** 视角中，导航到 **Topology**。
2. 查看事件源并点连接的事件 sink 查看右侧面板中的接收器详情。

第 3 章 事件 SINK

3.1. 事件 SINK

在创建事件源时，您可以指定事件从源发送到的事件接收器（sink）。事件接收器是一个可寻址或可调用的资源，可以从其他资源接收传入的事件。Knative 服务、频道和代理都是事件接收器示例。还有一个特定的 Apache Kafka 接收器类型。

可寻址的对象接收和确认通过 HTTP 发送的事件到其 **status.address.url** 字段中定义的地址。作为特殊情况，核心 Kubernetes **Service** 对象也履行可寻址的接口。

可调用的对象可以接收通过 HTTP 发送的事件并转换事件，并在 HTTP 响应中返回 **0** 或 **1** 新事件。这些返回的事件可能会象处理外部事件源中的事件一样进一步处理。

3.1.1. Knative CLI sink 标记

当使用 Knative (**kn**) CLI 创建事件源时，您可以使用 **--sink** 标志指定事件从该资源发送到的接收器。sink 可以是任何可寻址或可调用的资源，可以从其他资源接收传入的事件。

以下示例创建使用服务 **http://event-display.svc.cluster.local** 的接收器绑定作为接收器：

使用 sink 标记的命令示例

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
  --subject "Job:batch/v1:app=heartbeat-cron" \
  --sink http://event-display.svc.cluster.local \ 1
  --ce-override "sink=bound"
```

1 **http://event-display.svc.cluster.local** 中的 **svc** 确定接收器是一个 Knative 服务。其他默认的接收器前缀包括 **channel** 和 **broker**。

提示

您可以通过自定义 **kn**，配置哪些 CR 可在 Knative (**kn**) CLI 命令中使用 **--sink** 标记。

3.2. 创建事件接收器

在创建事件源时，您可以指定事件从源发送到的事件接收器（sink）。事件接收器是一个可寻址或可调用的资源，可以从其他资源接收传入的事件。Knative 服务、频道和代理都是事件接收器示例。还有一个特定的 Apache Kafka 接收器类型。

有关创建可用作事件接收器的资源的详情，请查看以下文档：

- 无服务器应用程序
- 创建代理
- 创建频道
- Kafka 接收器

3.3. APACHE KAFKA 的接收器

Apache Kafka sink 是集群中启用了 Apache Kafka 时可用的事件 sink 类型。您可以使用 Kafka sink 将事件从事件源发送到 Kafka 主题。

3.3.1. 使用 YAML 创建 Apache Kafka sink

您可以创建一个 Kafka 接收器将事件发送到 Kafka 主题。默认情况下，Kafka sink 使用二进制内容模式，其效率比结构化模式更高效。要使用 YAML 创建 Kafka sink，您必须创建一个 YAML 文件来定义 **KafkaSink** 对象，然后使用 **oc apply** 命令应用它。

先决条件

- 在集群中安装了 OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** 自定义资源 (CR)。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您可以访问 Red Hat AMQ Streams（Kafka）集群，该集群会生成您要导入的 Kafka 信息。
- 安装 OpenShift CLI (**oc**)。

流程

1. 创建一个 **KafkaSink** 对象定义作为一个 YAML 文件：

   Kafka sink YAML

   ```
   apiVersion: eventing.knative.dev/v1alpha1
   kind: KafkaSink
   metadata:
     name: <sink-name>
     namespace: <namespace>
   spec:
     topic: <topic-name>
     bootstrapServers:
      - <bootstrap-server>
   ```

2. 要创建 Kafka sink，请应用 **KafkaSink** YAML 文件：

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

3. 配置事件源，以便在其 spec 中指定 sink:

   连接到 API 服务器源的 Kafka sink 示例

   ```
   apiVersion: sources.knative.dev/v1alpha2
   kind: ApiServerSource
   metadata:
     name: <source-name> 1
     namespace: <namespace> 2
   spec:
     serviceAccountName: <service-account-name> 3
   ```

```
    mode: Resource
    resources:
    - apiVersion: v1
      kind: Event
    sink:
      ref:
        apiVersion: eventing.knative.dev/v1alpha1
        kind: KafkaSink
        name: <sink-name> 4
```

1. 事件源的名称。
2. 事件源的命名空间。
3. 事件源的服务帐户。
4. Kafka sink 名称。

3.3.2. 使用 OpenShift Container Platform Web 控制台为 Apache Kafka 创建事件 sink

您可以使用 OpenShift Container Platform Web 控制台中的 **Developer** 视角创建一个 Kafka 接收器将事件发送到 Kafka 主题。默认情况下，Kafka sink 使用二进制内容模式，其效率比结构化模式更高效。

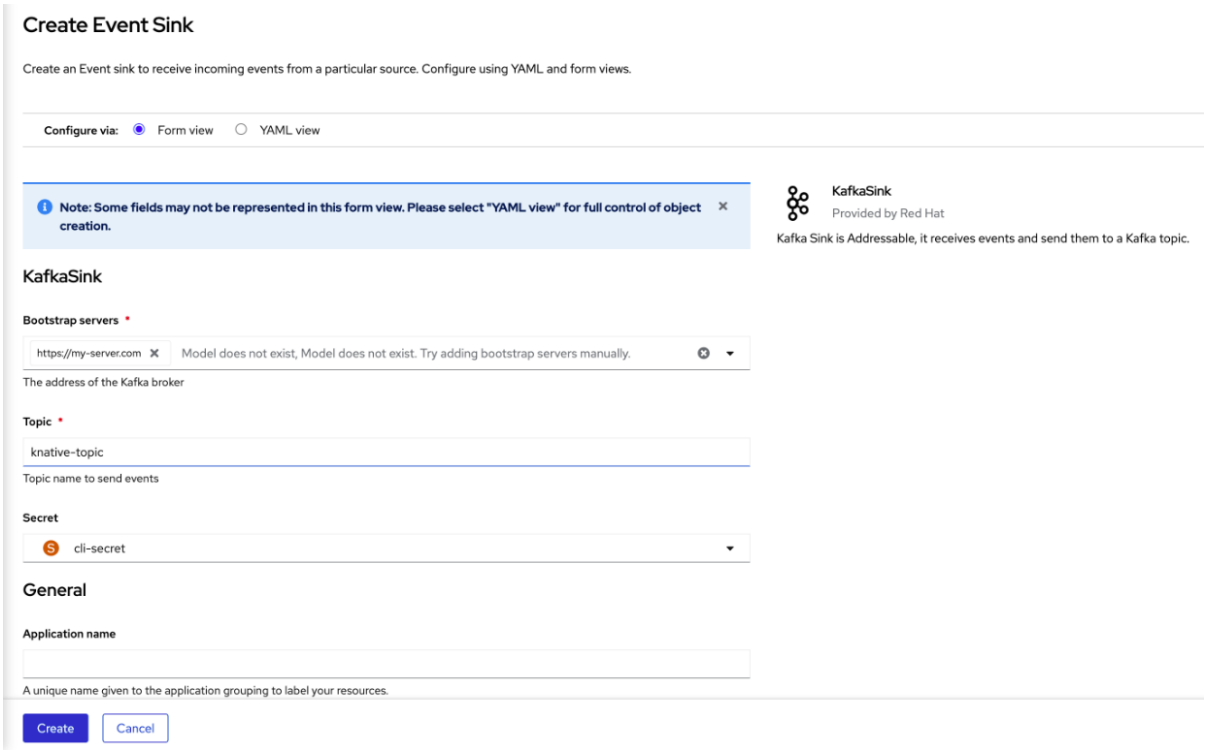
作为开发人员，您可以创建一个事件 sink 来从特定源接收事件并将其发送到 Kafka 主题。

先决条件

- 您已从 OperatorHub 安装了带有 Knative Serving、Knative Eventing 和 Knative 代理的 OpenShift Serverless Operator。
- 您已在 Kafka 环境中创建了一个 Kafka 主题。

流程

1. 在 **Developer** 视角中，进入到 **+Add** 视图。
2. 点 **Eventing** 目录中的 **Event Sink**。
3. 在目录项中搜索 **KafkaSink** 并点它。
4. 点 **Create Event Sink**。
5. 在表单视图中，键入 bootstrap 服务器的 URL，这是主机名和端口的组合。



6. 键入要发送事件数据的主题名称。
7. 键入事件 sink 的名称。
8. 点 **Create**。

验证

1. 在 **Developer** 视角中，进入 **Topology** 视图。
2. 点创建的事件 sink 在右侧面板中查看其详情。

3.3.3. 为 Apache Kafka sink 配置安全性

Apache Kafka 客户端和服务器使用 *传输层安全性* (TLS) 来加密 Knative 和 Kafka 之间的流量，以及用于身份验证。TLS 是 Apache Kafka 的 Knative 代理实现唯一支持的流量加密方法。

Apache Kafka 使用 *简单身份验证和安全层* (SASL) 进行身份验证。如果在集群中使用 SASL 身份验证，用户则必须向 Knative 提供凭证才能与 Kafka 集群通信，否则无法生成或消耗事件。

先决条件

- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** 自定义资源（CR）已安装在 OpenShift Container Platform 集群中。
- 在 **KnativeKafka** CR 中启用了 Kafka sink。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您有一个 Kafka 集群 CA 证书存储为一个 **.pem** 文件。
- 您有一个 Kafka 集群客户端证书，并存储为 **.pem** 文件的密钥。

- 已安装 OpenShift (**oc**) CLI。
- 您已选择使用 SASL 机制，例如 **PLAIN**、**SCRAM-SHA-256** 或 **SCRAM-SHA-512**。

流程

1. 在与 **KafkaSink** 对象相同的命名空间中创建一个 secret：

重要

证书和密钥必须采用 PEM 格式。

- 对于使用 SASL 时没有加密的身份验证：

```
$ oc create secret -n <namespace> generic <secret_name> \
  --from-literal=protocol=SASL_PLAINTEXT \
  --from-literal=sasl.mechanism=<sasl_mechanism> \
  --from-literal=user=<username> \
  --from-literal=password=<password>
```

- 对于使用 TLS 的 SASL 和加密进行身份验证：

```
$ oc create secret -n <namespace> generic <secret_name> \
  --from-literal=protocol=SASL_SSL \
  --from-literal=sasl.mechanism=<sasl_mechanism> \
  --from-file=ca.crt=<my_caroot.pem_file_path> \ 1
  --from-literal=user=<username> \
  --from-literal=password=<password>
```

1 如果您使用公共云管理的 Kafka 服务，可以省略 **ca.crt** 来使用系统的 root CA 设置。

- 使用 TLS 进行身份验证和加密：

```
$ oc create secret -n <namespace> generic <secret_name> \
  --from-literal=protocol=SSL \
  --from-file=ca.crt=<my_caroot.pem_file_path> \ 1
  --from-file=user.crt=<my_cert.pem_file_path> \
  --from-file=user.key=<my_key.pem_file_path>
```

1 如果您使用公共云管理的 Kafka 服务，可以省略 **ca.crt** 来使用系统的 root CA 设置。

2. 创建或修改 **KafkaSink** 对象，并在 **auth** spec 中添加对 secret 的引用：

```
apiVersion: eventing.knative.dev/v1alpha1
kind: KafkaSink
metadata:
  name: <sink_name>
  namespace: <namespace>
spec:
...
  auth:
    secret:
```

```
      ref:
        name: <secret_name>
...
```

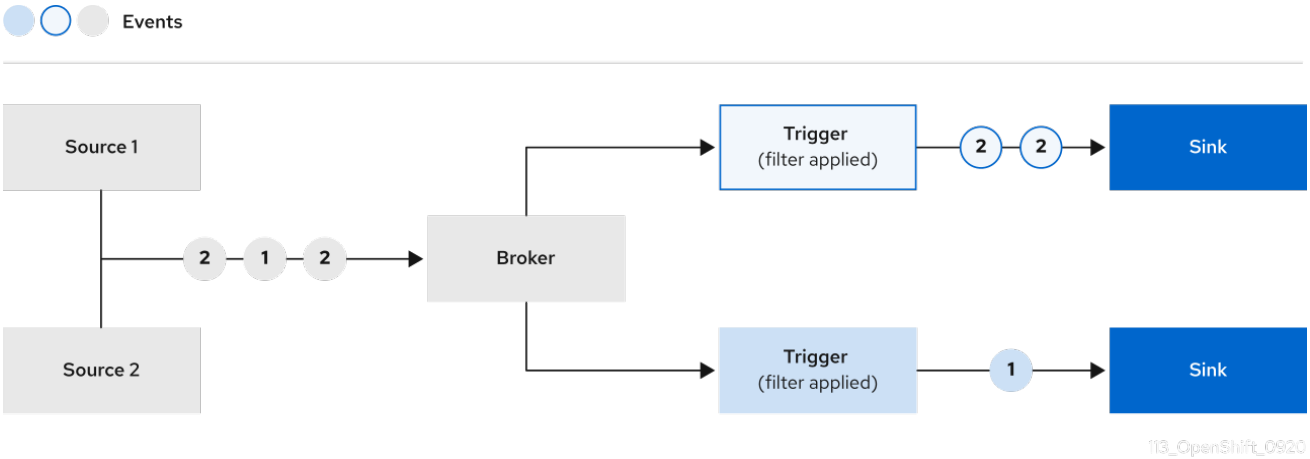
3. 应用 **KafkaSink** 对象：

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

第 4 章 代理（BROKER）

4.1. 代理（BROKER）

代理可与触发器结合使用，用于将事件源发送到事件 sink。事件从事件源发送到代理，作为 HTTP **POST** 请求。事件进入代理后，可使用触发器根据 CloudEvent 属性 进行过滤，并作为 HTTP **POST** 请求发送到事件 sink。



4.2. 代理类型

集群管理员可为集群设置 default 代理实施。创建代理时，会使用默认代理实现，除非在 **Broker** 对象中提供配置。

4.2.1. 用于开发的默认代理实现

Knative 提供基于频道的默认代理实现。这个基于频道的代理可用于开发和测试目的，但不为生产环境提供适当的事件交付保证。默认代理由 **InMemoryChannel** 频道实现支持。

如果要使用 Apache Kafka 减少网络跃点，请为 Apache Kafka 使用 Knative 代理实现。不要将基于频道的代理配置为由 **KafkaChannel** 频道实现支持。

4.2.2. Apache Kafka 的生产环境就绪的 Knative 代理实现

对于生产环境就绪的 Knative Eventing 部署，红帽建议为 Apache Kafka 使用 Knative 代理实现。代理是 Knative 代理的 Apache Kafka 原生实现，它将 CloudEvents 直接发送到 Kafka 实例。

Knative 代理与 Kafka 有一个原生集成，用于存储和路由事件。它可以更好地与 Kafka 集成用于代理，并在其他代理类型中触发模型，并减少网络跃点。Knative 代理实现的其他优点包括：

- 最少一次的交付保证
- 根据 CloudEvents 分区扩展排序事件交付
- 数据平面的高可用性
- 水平扩展数据平面

Apache Kafka 的 Knative 代理实现使用二进制内容模式将传入的 CloudEvents 存储为 Kafka 记录。这意味着，所有 CloudEvent 属性和扩展都会在 Kafka 记录上映射，而 CloudEvent 的 **data** 规格与 Kafka 记录的值对应。

4.3. 创建代理

Knative 提供基于频道的默认代理实现。这个基于频道的代理可用于开发和测试目的，但不为生产环境提供适当的事件交付保证。

如果集群管理员将 OpenShift Serverless 部署配置为使用 Apache Kafka 作为默认代理类型，使用默认设置创建代理会为 Apache Kafka 创建一个 Knative 代理。

如果您的 OpenShift Serverless 部署没有配置为使用 Apache Kafka 的 Knative 代理作为默认代理类型，则按照以下流程中的默认设置时会创建基于频道的代理。

4.3.1. 使用 Knative CLI 创建代理

代理可与触发器结合使用，用于将事件源发送到事件 sink。通过使用 Knative (**kn**) CLI 创建代理，通过直接修改 YAML 文件来提供更简化的、直观的用户界面。您可以使用 **kn broker create** 命令创建代理。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 已安装 Knative (**kn**) CLI。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

- 创建代理：

```
$ kn broker create <broker_name>
```

验证

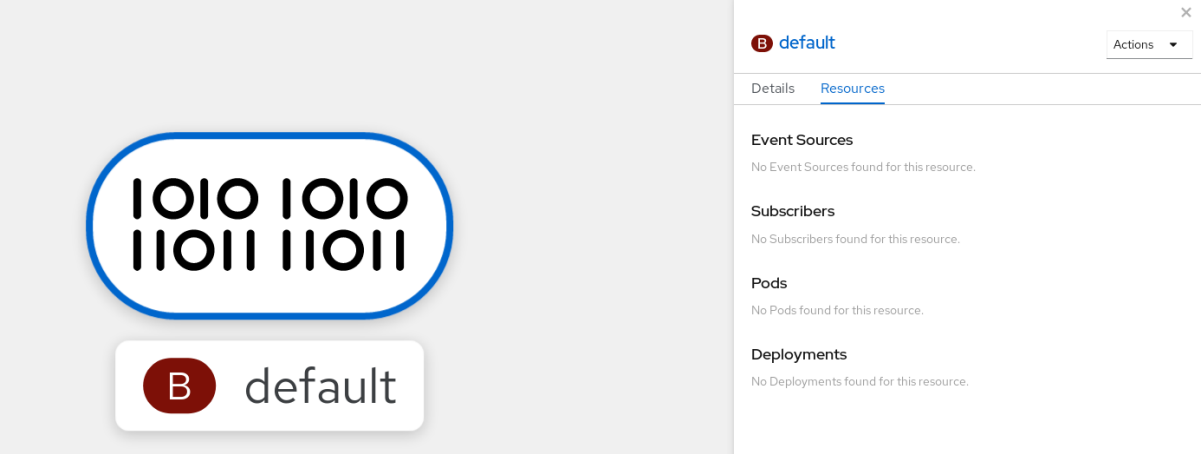
1. 使用 **kn** 命令列出所有现有代理：

```
$ kn broker list
```

输出示例

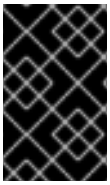
```
NAME      URL                                                    AGE   CONDITIONS   READY
REASON
default   http://broker-ingress.knative-eventing.svc.cluster.local/test/default   45s   5 OK / 5
True
```

2. 可选：如果使用 OpenShift Container Platform Web 控制台，在 **Developer** 视角中进入 **Topology** 视图来查看存在的代理：



4.3.2. 通过注解触发器来创建代理

代理可与触发器结合使用，用于将事件源发送到事件 sink。您可以通过将 **eventing.knative.dev/injection: enabled** 注解添加到 **Trigger** 对象来创建代理。



重要

如果您使用 **eventing.knative.dev/injection: enabled** 注解创建代理，则在没有集群管理员权限的情况下无法删除该代理。如果您在集群管理员还没有删除此注解前删除了代理，则代理会在删除后再次被创建。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 安装 OpenShift CLI (**oc**) 。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 创建一个 **Trigger** 对象作为 YAML 文件，该文件带有 **eventing.knative.dev/injection: enabled** 注解：

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  annotations:
    eventing.knative.dev/injection: enabled
  name: <trigger_name>
spec:
  broker: default
  subscriber: 1
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: <service_name>
```

1 指定触发器将事件发送到的事件 sink 或 *subscriber*。

2. 应用 **Trigger** YAML 文件：

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

验证

您可以使用 **oc** CLI，或使用 web 控制台中的 **Topology** 视图来验证代理是否已成功创建。

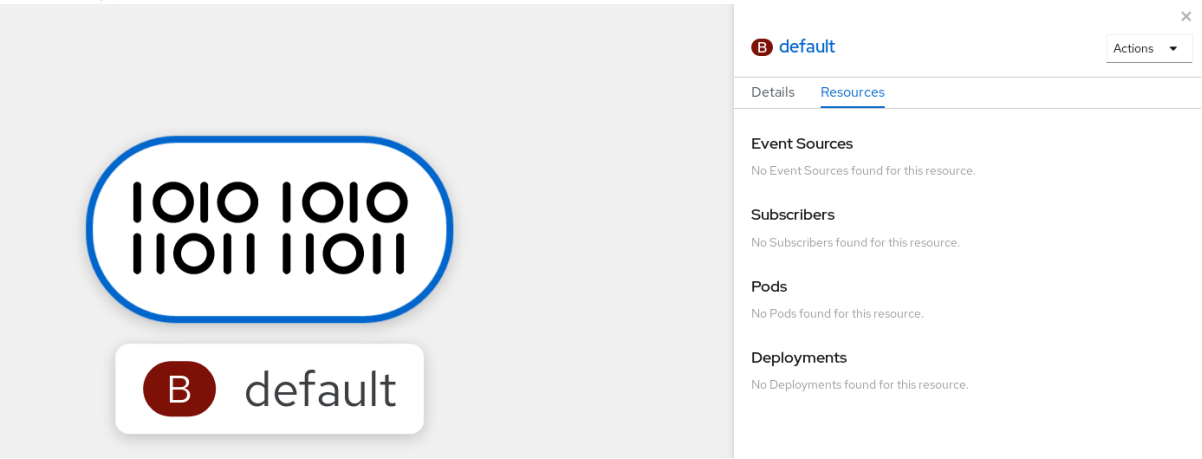
1. 输入以下 **oc** 命令来获取代理：

```
$ oc -n <namespace> get broker default
```

输出示例

```
NAME    READY    REASON    URL                                                                    AGE
default   True             http://broker-ingress.knative-eventing.svc.cluster.local/test/default
3m56s
```

2. 可选：如果使用 OpenShift Container Platform Web 控制台，在 **Developer** 视角中进入 **Topology** 视图来查看存在的代理：



4.3.3. 通过标记命名空间来创建代理

代理可与触发器结合使用，用于将事件源发送到事件 sink。您可以通过标记您拥有的命名空间或具有写入权限来自动创建 **default** 代理。

注意

如果您删除该标签，则不会删除使用这个方法创建的代理。您必须手动删除它们。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 安装 OpenShift CLI (**oc**) 。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

- 如果您使用 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated，则具有集群或专用管理员权限。

流程

- 使用 **eventing.knative.dev/injection=enabled** 标识一个命名空间：

```
$ oc label namespace <namespace> eventing.knative.dev/injection=enabled
```

验证

您可以使用 **oc** CLI，或使用 web 控制台中的 **Topology** 视图来验证代理是否已成功创建。

1. 使用 **oc** 命令获取代理：

```
$ oc -n <namespace> get broker <broker_name>
```

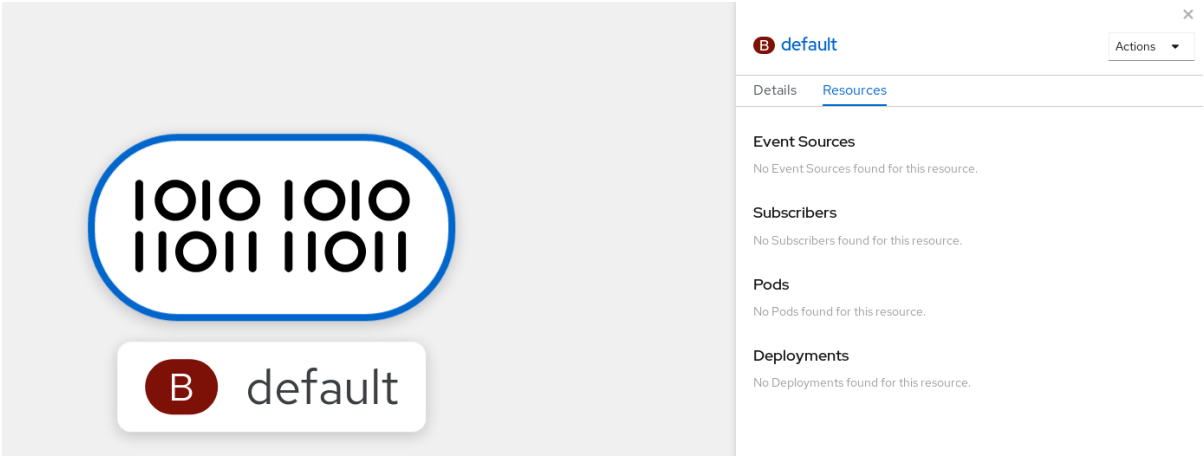
示例命令

```
$ oc -n default get broker default
```

输出示例

```
NAME      READY     REASON    URL                                                     AGE
default   True                http://broker-ingress.knative-eventing.svc.cluster.local/test/default   3m56s
```

2. 可选：如果使用 OpenShift Container Platform Web 控制台，在 **Developer** 视角中进入 **Topology** 视图来查看存在的代理：



4.3.4. 删除通过注入创建的代理

如果通过注入创建了一个代理并在以后需要删除它时，您必须手动删除它。如果删除了标签或注解，则使用命名空间标签或触发器注解创建的代理不会被永久删除。

先决条件

- 安装 OpenShift CLI (**oc**)。

流程

1. 从命名空间中删除 **eventing.knative.dev/injection=enabled** 标识：

```
$ oc label namespace <namespace> eventing.knative.dev/injection-
```

移除注解可防止 Knative 在删除代理后重新创建代理。

2. 从所选命名空间中删除代理：

```
$ oc -n <namespace> delete broker <broker_name>
```

验证

- 使用 **oc** 命令获取代理：

```
$ oc -n <namespace> get broker <broker_name>
```

示例命令

```
$ oc -n default get broker default
```

输出示例

```
No resources found.
Error from server (NotFound): brokers.eventing.knative.dev "default" not found
```

4.3.5. 使用 Web 控制台创建代理

在集群中安装 Knative Eventing 后，您可以使用 web 控制台创建代理。使用 OpenShift Container Platform Web 控制台提供了一个简化的用户界面来创建代理。

先决条件

- 已登陆到 OpenShift Container Platform Web 控制台。
- 在集群中安装了 OpenShift Serverless Operator、Knative Serving 和 Knative Eventing。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

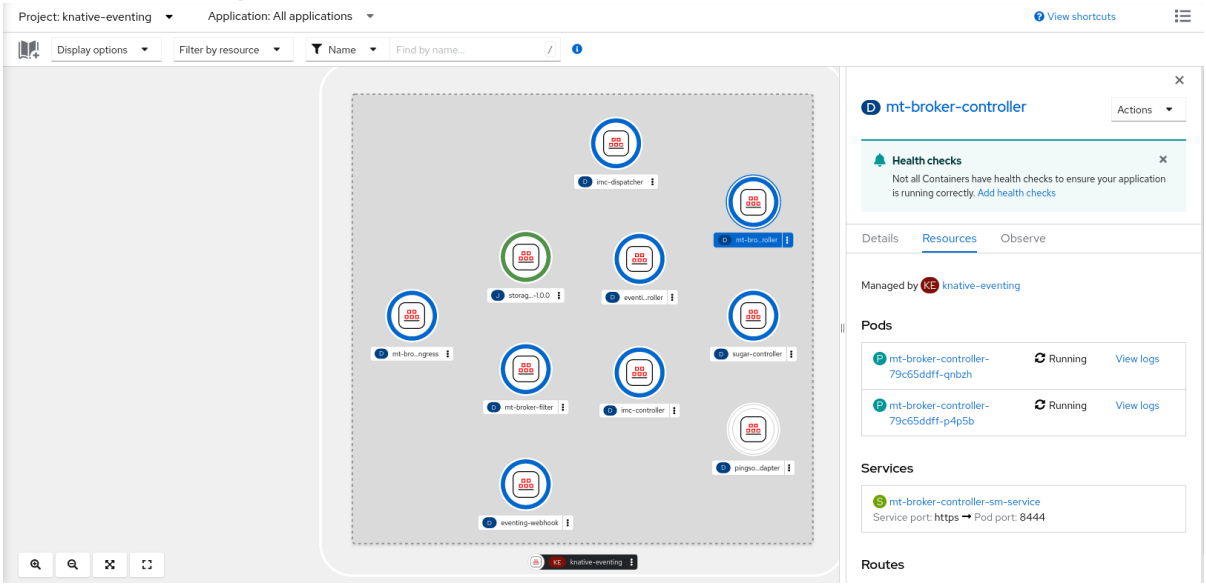
1. 在 **Developer** 视角中，进入到 **+Add → Broker**。此时会显示 **Broker** 页面。
2. 可选。更新代理的**名称**。如果您没有更新名称，则生成的代理名为 **default**。
3. 点 **Create**。

验证

您可以通过在 **Topology** 页面中查看代理组件来验证代理是否已创建。

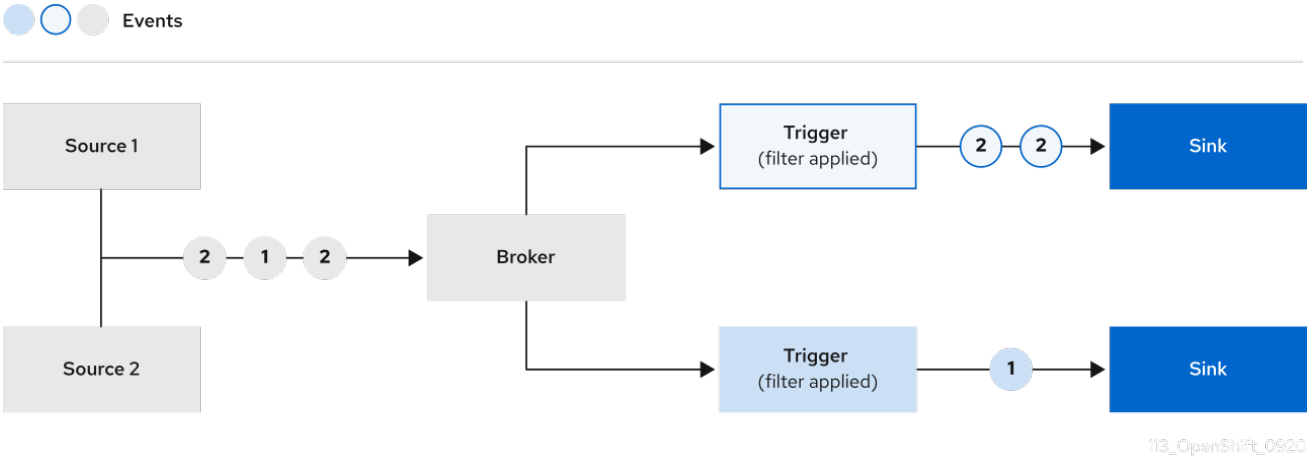
1. 在 **Developer** 视角中，导航到 **Topology**。

2. 查看 **mt-broker-ingress**、**mt-broker-filter** 和 **mt-broker-controller** 组件。



4.3.6. 使用 Administrator 视角创建代理

代理可与触发器结合使用，用于将事件源发送到事件 sink。事件从事件源发送到代理，作为 HTTP **POST** 请求。事件进入代理后，可使用触发器根据 CloudEvent 属性 进行过滤，并作为 HTTP **POST** 请求发送到事件 sink。



先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 您已登录到 Web 控制台，且处于 **Administrator** 视角。
- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Administrator** 视角中，导航到 **Serverless → Eventing**。
2. 在 **Create** 列表中，选择 **Broker**。您将进入 **Create Broker** 页面。

3. 可选：修改代理的 YAML 配置。
4. 点 **Create**。

4.3.7. 后续步骤

- 配置事件交付参数，当事件无法发送到事件 sink 时。

4.3.8. 其他资源

- 配置默认代理类
- 触发器
- 使用 Developer 视角将代理连接到 sink

4.4. 配置默认代理支持频道

如果您使用基于频道的代理，您可以将代理的默认后备频道类型设置为 **InMemoryChannel** 或 **KafkaChannel**。

先决条件

- 在 OpenShift Container Platform 上具有管理员权限。
- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。
- 已安装 OpenShift (**oc**) CLI。
- 如果要使用 Apache Kafka 频道作为默认后备频道类型，还必须在集群中安装 **KnativeKafka** CR。

流程

1. 修改 **KnativeEventing** 自定义资源 (CR)以添加 **config-br-default-channel** 配置映射的配置详情：

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  config: (1)
    config-br-default-channel:
      channel-template-spec: |
        apiVersion: messaging.knative.dev/v1beta1
        kind: KafkaChannel (2)
        spec:
          numPartitions: 6 (3)
          replicationFactor: 3 (4)
```

(1) 在 **spec.config** 中，您可以指定您要为修改的配置添加的配置映射。

2 默认后备频道类型配置。在本例中，集群的默认频道实现是 **KafkaChannel**。

3 支持代理的 Kafka 频道的分区数量。

4 支持代理的 Kafka 频道的复制因素。

2. 应用更新的 **KnativeEventing** CR：

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

4.5. 配置默认代理类

您可以使用 **config-br-defaults** 配置映射来指定 Knative Eventing 的默认代理类设置。您可以为整个集群或一个或多个命名空间指定默认代理类。目前，支持 **MTChannelBasedBroker** 和 **Kafka** 代理类型。

先决条件

- 在 OpenShift Container Platform 上具有管理员权限。
- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。
- 如果要将 Apache Kafka 的 Knative 代理用作默认代理实现，还必须在集群中安装 **KnativeKafka** CR。

流程

- 修改 **KnativeEventing** 自定义资源，以添加 **config-br-defaults** 配置映射的配置详情：

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  defaultBrokerClass: Kafka 1
  config: 2
    config-br-defaults: 3
      default-br-config: |
        clusterDefault: 4
          brokerClass: Kafka
          apiVersion: v1
          kind: ConfigMap
          name: kafka-broker-config 5
          namespace: knative-eventing 6
        namespaceDefaults: 7
          my-namespace:
            brokerClass: MTChannelBasedBroker
            apiVersion: v1
            kind: ConfigMap
            name: config-br-default-channel 8
            namespace: knative-eventing 9
...
```

1. Knative Eventing 的默认代理类。
2. 在 **spec.config** 中，您可以指定您要为修改的配置添加的配置映射。
3. **config-br-defaults** 配置映射指定任何没有指定 **spec.config** 设置或代理类的代理的默认设置。
4. 集群范围的默认代理类配置。在本例中，集群的默认代理类实现是 **Kafka**。
5. **kafka-broker-config** 配置映射指定 Kafka 代理的默认设置。请参阅"添加资源"部分中的"为 Apache Kafka 设置配置 Knative 代理"。
6. 存在 **kafka-broker-config** 配置映射的命名空间。
7. 命名空间范围的默认代理类配置。在本例中，**my-namespace** 命名空间的默认代理类实现是 **MTChannelbasedBroker**。您可以为多个命名空间指定默认代理类实现。
8. **config-br-default-channel** 配置映射指定代理的默认后备频道。请参阅"Additional resources"部分的"配置默认代理支持频道"部分。
9. **config-br-default-channel** 配置映射所在的命名空间。

重要

配置特定于命名空间的默认设置会覆盖任何集群范围的设置。

4.6. APACHE KAFKA 的 KNATIVE 代理实现

对于生产环境就绪的 Knative Eventing 部署，红帽建议为 Apache Kafka 使用 Knative 代理实现。代理是 Knative 代理的 Apache Kafka 原生实现，它将 CloudEvents 直接发送到 Kafka 实例。

Knative 代理与 Kafka 有一个原生集成，用于存储和路由事件。它可以更好地与 Kafka 集成用于代理，并在其他代理类型中触发模型，并减少网络跃点。Knative 代理实现的其他优点包括：

- 最少一次的交付保证
- 根据 CloudEvents 分区扩展排序事件交付
- 数据平面的高可用性
- 水平扩展数据平面

Apache Kafka 的 Knative 代理实现使用二进制内容模式将传入的 CloudEvents 存储为 Kafka 记录。这意味着，所有 CloudEvent 属性和扩展都会在 Kafka 记录上映射，而 CloudEvent 的 **data** 规格与 Kafka 记录的值对应。

4.6.1. 当没有配置为 default 代理类型时，创建 Apache Kafka 代理

如果您的 OpenShift Serverless 部署没有配置为使用 Kafka 代理作为默认代理类型，您仍可使用以下步骤创建基于 Kafka 的代理。

4.6.1.1. 使用 YAML 创建 Apache Kafka 代理

使用 YAML 文件创建 Knative 资源使用声明性 API，它允许您以声明性的方式描述应用程序，并以可重复的方式描述应用程序。要使用 YAML 创建 Kafka 代理，您必须创建一个 YAML 文件来定义 **Broker** 对象，然后使用 **oc apply** 命令应用它。

先决条件

- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** 自定义资源已安装在 OpenShift Container Platform 集群中。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 创建一个基于 Kafka 的代理作为 YAML 文件：

```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  annotations:
    eventing.knative.dev/broker.class: Kafka (1)
  name: example-kafka-broker
spec:
  config:
    apiVersion: v1
    kind: ConfigMap
    name: kafka-broker-config (2)
    namespace: knative-eventing
```

(1) 代理类。如果没有指定，代理使用由集群管理员配置的默认类。要使用 Kafka 代理，这个值必须是 **Kafka**。

(2) Apache Kafka 的 Knative 代理的默认配置映射。当集群管理员在集群中启用 Kafka 代理功能时，会创建此配置映射。

2. 应用基于 Kafka 的代理 YAML 文件：

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

4.6.1.2. 创建使用外部管理的 Kafka 主题的 Apache Kafka 代理

如果要在不创建自己的内部主题的情况下使用 Kafka 代理，您可以使用外部管理的 Kafka 主题。要做到这一点，您必须创建一个使用 **kafka.eventing.knative.dev/external.topic** 注解的 Kafka **Broker** 对象。

先决条件

- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** 自定义资源已安装在 OpenShift Container Platform 集群中。
- 您可以访问一个 Kafka 实例，如 Red Hat AMQ Streams，并创建了 Kafka 主题。

- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 创建一个基于 Kafka 的代理作为 YAML 文件：

```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  annotations:
    eventing.knative.dev/broker.class: Kafka (1)
    kafka.eventing.knative.dev/external.topic: <topic_name> (2)
...
```

(1) 代理类。如果没有指定，代理使用由集群管理员配置的默认类。要使用 Kafka 代理，这个值必须是 **Kafka**。

(2) 要使用的 Kafka 主题的名称。

2. 应用基于 Kafka 的代理 YAML 文件：

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

4.6.1.3. 使用隔离数据平面的 Apache Kafka 的 Knative Broker 实现

重要

带有隔离数据平面的 Apache Kafka 的 Knative Broker 实现只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

Apache Kafka 的 Knative Broker 实现有 2 个平面：

Control plane（控制平面）

由与 Kubernetes API 通信、监视自定义对象和管理数据平面的控制器组成。

数据平面

侦听传入事件、与 Apache Kafka 通信以及向事件接收器发送事件的组件集合。Apache Kafka 数据平面的 Knative Broker 实现是事件流的位置。实现由 **kafka-broker-receiver** 和 **kafka-broker-dispatcher** 部署组成。

当您配置 Kafka 的代理类时，Apache **Kafka** 的 Knative Broker 实现会使用共享的数据平面。这意味着 **knative-eventing** 命名空间中的 **kafka-broker-receiver** 和 **kafka-broker-dispatcher** 部署用于集群中的所有 Apache Kafka Broker。

但是，当您配置 **KafkaNamespaced** 的代理类时，Apache Kafka 代理控制器会为代理存在的每个命名空间创建一个新的数据平面。该数据平面都会被该命名空间中的所有 **KafkaNamespaced** 代理使用。这提

供了数据平面之间的隔离，因此用户命名空间中的 **kafka-broker-receiver** 和 **kafka-broker-dispatcher** 部署仅用于该命名空间中的代理。

重要

由于具有单独的数据平面，这个安全功能会创建更多部署并使用更多资源。除非有这样的隔离要求，请使用一个带有 **Kafka** 类的 **常规** 代理。

4.6.1.4. 为使用隔离的数据平面的 Apache Kafka 创建 Knative 代理

重要

带有隔离数据平面的 Apache Kafka 的 Knative Broker 实现只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

要创建 **KafkaNamespaced** 代理，您必须将 **eventing.knative.dev/broker.class** 注解设置为 **KafkaNamespaced**。

先决条件

- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** 自定义资源已安装在 OpenShift Container Platform 集群中。
- 您可以访问 Apache Kafka 实例，如 Red Hat AMQ Streams，并创建了 Kafka 主题。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 使用 YAML 文件创建基于 Apache Kafka 的代理：

```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  annotations:
    eventing.knative.dev/broker.class: KafkaNamespaced 1
  name: default
  namespace: my-namespace 2
spec:
  config:
    apiVersion: v1
    kind: ConfigMap
    name: my-config 3
...
```

1. 要将 Apache Kafka 代理与隔离的数据平面搭配使用，代理类值必须是 **KafkaNamespaced**。

2 3 引用的 **ConfigMap** 对象 **my-config** 必须与 **Broker** 对象位于同一个命名空间中，本例中为 **my-namespace**。

2. 应用基于 Apache Kafka 的代理 YAML 文件：

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

重要

spec.config 中的 **ConfigMap** 对象必须与 **Broker** 对象位于同一个命名空间中：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: my-namespace
data:
  ...
```

使用 **KafkaNamespaced** 类创建第一个 **Broker** 对象后，在命名空间中创建 **kafka-broker-receiver** 和 **kafka-broker-dispatcher** 部署。因此，同一命名空间中的 **KafkaNamespaced** 类的所有代理都将使用相同的数据平面。如果命名空间中没有 **KafkaNamespaced** 类的代理，则会删除命名空间中的数据平面。

4.6.2. 配置 Apache Kafka 代理设置

您可以通过创建配置映射并在 Kafka **Broker** 对象中引用此配置映射，配置复制因素、bootstrap 服务器和 Kafka 代理的主题分区数量。

先决条件

- 在 OpenShift Container Platform 上具有集群或专用管理员权限。
- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** 自定义资源（CR）已安装在 OpenShift Container Platform 集群中。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 修改 **kafka-broker-config** 配置映射，或创建自己的配置映射来包含以下配置：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: <config_map_name> 1
  namespace: <namespace> 2
data:
```

```
default.topic.partitions: <integer> 3
default.topic.replication.factor: <integer> 4
bootstrap.servers: <list_of_servers> 5
```

1. 配置映射名称。
2. 配置映射所在的命名空间。
3. Kafka 代理的主题分区数量。这控制如何将事件发送到代理的速度。更多分区需要更多计算资源。
4. 主题消息的复制因素。这可防止数据丢失。更高的复制因素需要更大的计算资源和更多存储。
5. 以逗号分隔的 bootstrap 服务器列表。这可以位于 OpenShift Container Platform 集群内部或外部，是代理从发送事件发送到的 Kafka 集群列表。

重要

default.topic.replication.factor 值必须小于或等于集群中的 Kafka 代理实例数量。例如，如果您只有一个 Kafka 代理，则 **default.topic.replication.factor** 值应该不超过 **"1"**。

Kafka 代理配置映射示例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kafka-broker-config
  namespace: knative-eventing
data:
  default.topic.partitions: "10"
  default.topic.replication.factor: "3"
  bootstrap.servers: "my-cluster-kafka-bootstrap.kafka:9092"
```

2. 应用配置映射：

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

3. 指定 Kafka **Broker** 对象的配置映射：

Broker 对象示例

```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  name: <broker_name> 1
  namespace: <namespace> 2
  annotations:
    eventing.knative.dev/broker.class: Kafka 3
spec:
  config:
```

```
    apiVersion: v1
    kind: ConfigMap
    name: <config_map_name> 4
    namespace: <namespace> 5
...
```

1. 代理名称。
2. 代理存在的命名空间。
3. 代理类注解。在本例中，代理是使用类值 Kafka 的 **Kafka** 代理。
4. 配置映射名称。
5. 配置映射所在的命名空间。

4. 应用代理：

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

4.6.3. Apache Kafka 的 Knative 代理实现的安全配置

Kafka 集群通常使用 TLS 或 SASL 身份验证方法进行保护。您可以使用 TLS 或 SASL 将 Kafka 代理或频道配置为针对受保护的 Red Hat AMQ Streams 集群进行操作。

注意

红帽建议您同时启用 SASL 和 TLS。

4.6.3.1. 为 Apache Kafka 代理配置 TLS 身份验证

Apache Kafka 客户端和服务器使用 *传输层安全性* (TLS) 来加密 Knative 和 Kafka 之间的流量，以及用于身份验证。TLS 是 Apache Kafka 的 Knative 代理实现唯一支持的流量加密方法。

先决条件

- 在 OpenShift Container Platform 上具有集群或专用管理员权限。
- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** CR 已安装在 OpenShift Container Platform 集群中。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您有一个 Kafka 集群 CA 证书存储为一个 **.pem** 文件。
- 您有一个 Kafka 集群客户端证书，并存储为 **.pem** 文件的密钥。
- 安装 OpenShift CLI (**oc**) 。

流程

1. 在 **knative-eventing** 命名空间中创建证书文件作为 secret：

```
$ oc create secret -n knative-eventing generic <secret_name> \
  --from-literal=protocol=SSL \
  --from-file=ca.crt=caroot.pem \
  --from-file=user.crt=certificate.pem \
  --from-file=user.key=key.pem
```

重要

使用密钥名称 **ca.crt**、**user.crt** 和 **user.key**。不要更改它们。

2. 编辑 **KnativeKafka** CR，并在 **broker** spec 中添加对 secret 的引用：

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  namespace: knative-eventing
  name: knative-kafka
spec:
  broker:
    enabled: true
    defaultConfig:
      authSecretName: <secret_name>
...
```

4.6.3.2. 为 Apache Kafka 代理配置 SASL 身份验证

Apache Kafka 使用 *简单身份验证和安全层* (SASL) 进行身份验证。如果在集群中使用 SASL 身份验证，用户则必须向 Knative 提供凭证才能与 Kafka 集群通信，否则无法生成或消耗事件。

先决条件

- 在 OpenShift Container Platform 上具有集群或专用管理员权限。
- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** CR 已安装在 OpenShift Container Platform 集群中。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您有一个 Kafka 集群的用户名和密码。
- 您已选择使用 SASL 机制，例如 **PLAIN**、**SCRAM-SHA-256** 或 **SCRAM-SHA-512**。
- 如果启用了 TLS，您还需要 Kafka 集群的 **ca.crt** 证书文件。
- 安装 OpenShift CLI (**oc**) 。

流程

1. 在 **knative-eventing** 命名空间中创建证书文件作为 secret：

```
$ oc create secret -n knative-eventing generic <secret_name> \
  --from-literal=protocol=SASL_SSL \
  --from-literal=sasl.mechanism=<sasl_mechanism> \
```

```
--from-file=ca.crt=caroot.pem \
--from-literal=password="SecretPassword" \
--from-literal=user="my-sasl-user"
```

- 使用键名 **ca.crt**, **password**, 和 **sasl.mechanism**.不要更改它们。
- 如果要将 SASL 与公共 CA 证书搭配使用，您必须在创建 secret 时使用 **tls.enabled=true** 标志，而不是使用 **ca.crt** 参数。例如：

```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
  --from-literal=tls.enabled=true \
  --from-literal=password="SecretPassword" \
  --from-literal=saslType="SCRAM-SHA-512" \
  --from-literal=user="my-sasl-user"
```

2. 编辑 **KnativeKafka** CR，并在 **broker** spec 中添加对 secret 的引用：

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  namespace: knative-eventing
  name: knative-kafka
spec:
  broker:
    enabled: true
    defaultConfig:
      authSecretName: <secret_name>
...
```

4.6.4. 其他资源

- Red Hat AMQ Streams 文档
- Kafka 上的 TLS 和 SASL

4.7. 管理代理

创建代理后，您可以使用 Knative (**kn**) CLI 命令或在 OpenShift Container Platform web 控制台中修改代理。

4.7.1. 使用 CLI 管理代理

Knative (**kn**) CLI 提供了可用于描述和列出现有代理的命令。

4.7.1.1. 使用 Knative CLI 列出现有代理

使用 Knative (**kn**) CLI 列出代理提供了精简且直观的用户界面。您可以使用 **kn broker list** 命令列出集群中的现有代理。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。

- 已安装 Knative (**kn**) CLI。

流程

- 列出所有存在的代理：

```
$ kn broker list
```

输出示例

```
NAME      URL                                                    AGE   CONDITIONS   READY   REASON
default   http://broker-ingress.knative-eventing.svc.cluster.local/test/default   45s   5 OK / 5   True
```

4.7.1.2. 使用 Knative CLI 描述现有代理

使用 Knative (**kn**) 描述代理提供了精简且直观的用户界面。您可以使用 **kn broker describe** 命令通过 Knative CLI 输出集群中现有代理的信息。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 已安装 Knative (**kn**) CLI。

流程

- 描述现有代理：

```
$ kn broker describe <broker_name>
```

使用 default broker 的命令示例

```
$ kn broker describe default
```

输出示例

```
Name:         default
Namespace:    default
Annotations:  eventing.knative.dev/broker.class=MTChannelBasedBroker,
eventing.knative.dev/creato ...
Age:          22s

Address:
  URL:    http://broker-ingress.knative-eventing.svc.cluster.local/default/default

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                22s
  ++ Addressable          22s
```

```
++ FilterReady           22s
++ IngressReady          22s
++ TriggerChannelReady   22s
```

4.7.2. 使用 Developer 视角将代理连接到 sink

您可以通过创建一个触发器，将代理连接到 OpenShift Container Platform **Developer** 视角中的事件 sink。

先决条件

- OpenShift Serverless Operator、Knative Serving 和 Knative Eventing 已在 OpenShift Container Platform 集群中安装。
- 已登陆到 web 控制台，且处于 **Developer** 视角。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您已创建了 sink，如 Knative 服务或频道。
- 您已创建了代理。

流程

1. 在 **Topology** 视图中，指向您创建的代理。此时会出现一个箭头。将箭头拖到您要连接到代理的接收器。此操作将打开 **Add Trigger** 对话框。
2. 在 **Add Trigger** 对话框中，为触发器输入一个名称并点 **Add**。

验证

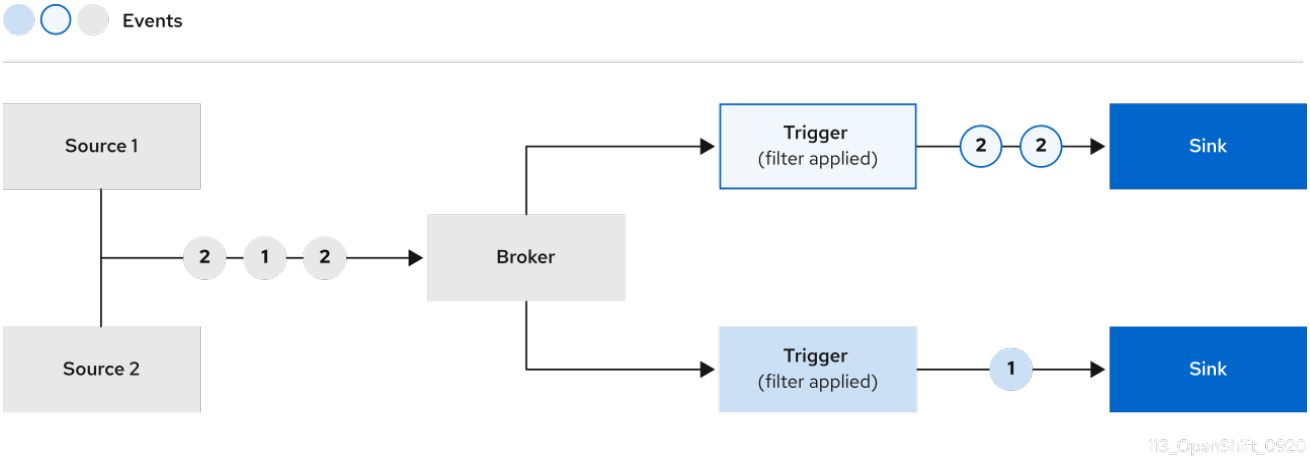
您可以通过查看 **Topology** 页面来验证代理是否已连接到 sink。

1. 在 **Developer** 视角中，导航到 **Topology**。
2. 点击将代理连接到 sink 的行，以便在 **Details** 面板中查看触发器的详情。

第 5 章 触发器

5.1. 触发器概述

代理可与触发器结合使用，用于将事件源发送到事件 sink。事件从事件源发送到代理，作为 HTTP **POST** 请求。事件进入代理后，可使用触发器根据 CloudEvent 属性 进行过滤，并作为 HTTP **POST** 请求发送到事件 sink。



如果您将 Knative 代理用于 Apache Kafka，您可以将事件的交付顺序从触发器配置为事件 sink。请参阅为触发器配置事件交付顺序。

5.1.1. 为触发器配置事件交付顺序

如果使用 Kafka 代理，您可以将事件的交付顺序从触发器配置为事件 sink。

先决条件

- OpenShift Serverless Operator、Knative Eventing 和 Knative Kafka 安装在 OpenShift Container Platform 集群中。
- Kafka 代理被启用在集群中使用，您也创建了一个 Kafka 代理。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift (**oc**) CLI。

流程

1. 创建或修改 **Trigger** 对象并设置 **kafka.eventing.knative.dev/delivery.order** 注解：

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: <trigger_name>
  annotations:
    kafka.eventing.knative.dev/delivery.order: ordered
# ...
```

支持的消费者交付保证有：

unordered

未排序的消费者是一种非阻塞消费者，它能以未排序的方式提供消息，同时保持正确的偏移管理。

排序的

一个订购的消费者是一个按分区阻止消费者，在提供分区的下一个消息前等待来自 CloudEvent 订阅者成功响应。
默认排序保证是 **unordered**。

2. 应用 **Trigger** 对象：

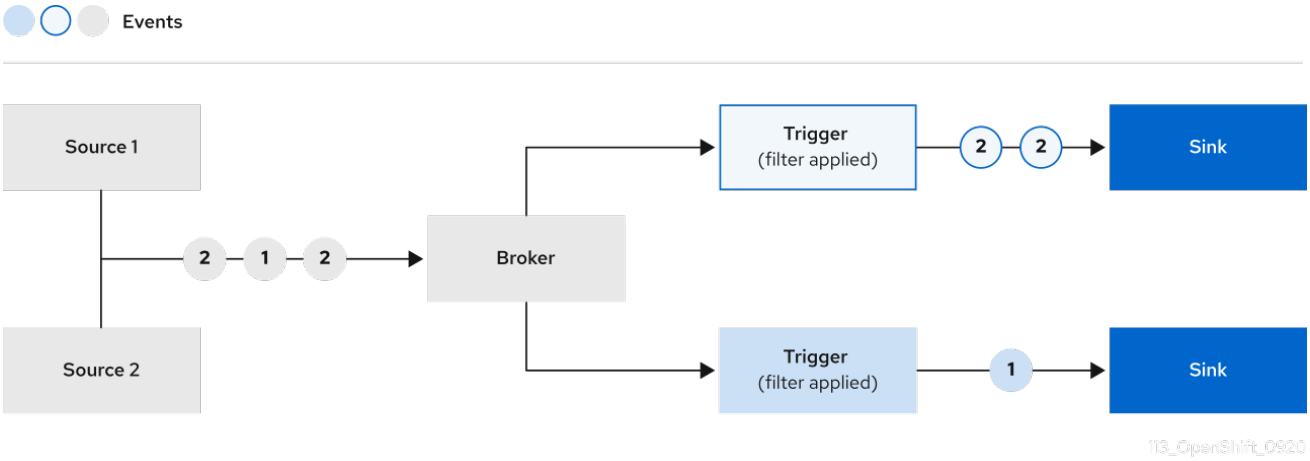
```
$ oc apply -f <filename>
```

5.1.2. 后续步骤

- 配置事件交付参数，当事件无法发送到事件 sink 时。

5.2. 创建触发器

代理可与触发器结合使用，用于将事件源发送到事件 sink。事件从事件源发送到代理，作为 HTTP **POST** 请求。事件进入代理后，可使用触发器根据 CloudEvent 属性 进行过滤，并作为 HTTP **POST** 请求发送到事件 sink。



5.2.1. 使用 Administrator 视角创建触发器

使用 OpenShift Container Platform Web 控制台提供了一个简化且直观的用户界面来创建触发器。在集群中安装 Knative Eventing 并创建了代理后，您可以使用 web 控制台创建触发器。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 您已登录到 Web 控制台，且处于 **Administrator** 视角。
- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。
- 您已创建了 Knative 代理。

- 您已创建了 Knative 服务以用作订阅者。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Administrator** 视角中，导航到 **Serverless → Eventing**。
2. 在 **Broker** 选项卡中，为您要在其中添加触发器的代理选择 Options 菜单  。
3. 点列表中的 **Add Trigger**。
4. 在 **Add Trigger** 对话框中，为触发器选择 **Subscriber**。订阅者是可以从代理接收事件的 Knative 服务。
5. 点 **Add**。

5.2.2. 使用 Developer 视角创建触发器

使用 OpenShift Container Platform Web 控制台提供了一个简化且直观的用户界面来创建触发器。在集群中安装 Knative Eventing 并创建了代理后，您可以使用 web 控制台创建触发器。

先决条件

- OpenShift Serverless Operator、Knative Serving 和 Knative Eventing 已在 OpenShift Container Platform 集群中安装。
- 已登陆到 web 控制台。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您已创建了代理和 Knative 服务或其他事件 sink 以连接触发器。

流程

1. 在 **Developer** 视角中，进入 **Topology** 页。
2. 将鼠标悬停在您要创建触发器的代理上，并拖动箭头。此时会显示 **Add Trigger** 选项。
3. 点 **Add Trigger**。
4. 在 **Subscriber** 列表中选择您的接收器。
5. 点 **Add**。

验证

- 创建订阅后，您可以在 **Topology** 页面中查看它，其中它是一个将代理连接到事件 sink 的行。

删除触发器

1. 在 **Developer** 视角中，进入 **Topology** 页。
2. 点您要删除的触发器。

3. 在 **Actions** 上下文菜单中，选择 **Delete Trigger**。

5.2.3. 使用 Knative CLI 创建触发器

您可以使用 **kn trigger create** 命令创建触发器。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 已安装 Knative (**kn**) CLI。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

- 创建触发器：

```
$ kn trigger create <trigger_name> --broker <broker_name> --filter <key=value> --sink
<sink_name>
```

或者，您可以创建触发器并使用代理注入同时创建 **default** 代理：

```
$ kn trigger create <trigger_name> --inject-broker --filter <key=value> --sink <sink_name>
```

默认情况下，触发器会将发送到代理的所有事件转发到订阅到该代理的 sink。通过对触发器使用 **--filter** 属性，您可以从代理过滤事件，这样订阅者才会根据您定义的标准接收一小部分事件。

5.3. 从命令行列出触发器

使用 Knative (**kn**) CLI 列出触发器提供精简、直观的用户界面。

5.3.1. 使用 Knative CLI 列出触发器

您可以使用 **kn trigger list** 命令列出集群中的现有触发器。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 已安装 Knative (**kn**) CLI。

流程

1. 显示可用触发器列表：

```
$ kn trigger list
```

输出示例

```
NAME    BROKER    SINK          AGE   CONDITIONS   READY   REASON
email   default   ksvc:edisplay   4s    5 OK / 5     True
ping    default   ksvc:edisplay   32s   5 OK / 5     True
```

2. 可选：以 JSON 格式输出触发器列表：

```
$ kn trigger list -o json
```

5.4. 描述从命令行中的触发器

使用 Knative (**kn**) CLI 描述触发器，提供了一个简化且直观的用户界面。

5.4.1. 使用 Knative CLI 描述触发器

您可以通过 **kn trigger describe** 命令使用 Knative CLI 输出集群中现有触发器的信息。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 已安装 Knative (**kn**) CLI。
- 您已创建了触发器。

流程

- 输入命令：

```
$ kn trigger describe <trigger_name>
```

输出示例

```
Name:         ping
Namespace:    default
Labels:       eventing.knative.dev/broker=default
Annotations:  eventing.knative.dev/creator=kube:admin,
eventing.knative.dev/lastModifier=kube:admin
Age:          2m
Broker:       default
Filter:
  type:       dev.knative.event

Sink:
  Name:       edisplay
  Namespace:  default
  Resource:   Service (serving.knative.dev/v1)

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                2m
  ++ BrokerReady          2m
```

```
++ DependencyReady       2m
++ Subscribed            2m
++ SubscriberResolved    2m
```

5.5. 将触发器连接到 SINK

您可以将触发器连接到 sink，以便在将代理的事件发送到 sink 前过滤代理的事件。在 **Trigger** 对象的资源规格中，连接到触发器的 sink 会配置为 **订阅者**。

连接到 Apache Kafka sink 的 Trigger 对象示例

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: <trigger_name> 1
spec:
...
  subscriber:
    ref:
      apiVersion: eventing.knative.dev/v1alpha1
      kind: KafkaSink
      name: <kafka_sink_name> 2
```

1. 连接到 sink 的触发器的名称。
2. **KafkaSink** 对象的名称。

5.6. 从命令行过滤触发器

使用 Knative (**kn**) CLI 使用 kn CLI 通过触发器过滤事件，可提供精简且直观的用户界面。您可以使用 **kn trigger create** 命令和适当的标记来通过使用触发器过滤事件。

5.6.1. 使用 Knative CLI 使用触发器过滤事件

在以下触发器示例中，只有带有属性 **type: dev.knative.samples.helloworld** 的事件才会发送到事件 sink:

```
$ kn trigger create <trigger_name> --broker <broker_name> --filter
type=dev.knative.samples.helloworld --sink ksvc:<service_name>
```

您还可以使用多个属性过滤事件。以下示例演示了如何使用类型、源和扩展属性过滤事件：

```
$ kn trigger create <trigger_name> --broker <broker_name> --sink ksvc:<service_name> \
--filter type=dev.knative.samples.helloworld \
--filter source=dev.knative.samples/helloworldsource \
--filter myextension=my-extension-value
```

5.7. 从命令行更新触发器

使用 Knative (**kn**) CLI 更新触发器提供精简、直观的用户界面。

5.7.1. 使用 Knative CLI 更新触发器

您可以使用带有特定标志的 **kn trigger update** 命令来更新触发器的属性。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 已安装 Knative (**kn**) CLI。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

- 更新触发器：

```
$ kn trigger update <trigger_name> --filter <key=value> --sink <sink_name> [flags]
```

  - 您可以更新触发器来过滤与传入事件匹配的事件属性。例如，使用 **type** 属性：

```
$ kn trigger update <trigger_name> --filter type=knative.dev.event
```

  - 您可以从触发器中删除过滤器属性。例如，您可以使用键 **type** 来删除过滤器属性：

```
$ kn trigger update <trigger_name> --filter type-
```

  - 您可以使用 **--sink** 参数来更改触发器的事件 sink:

```
$ kn trigger update <trigger_name> --sink ksvc:my-event-sink
```

5.8. 从命令行删除触发器

使用 Knative (**kn**) CLI 删除触发器提供精简而直观的用户界面。

5.8.1. 使用 Knative CLI 删除触发器

您可以使用 **kn trigger delete** 命令删除触发器。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 已安装 Knative (**kn**) CLI。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

- 删除触发器：

```
$ kn trigger delete <trigger_name>
```

验证

1. 列出现有触发器：

```
$ kn trigger list
```

2. 验证触发器不再存在：

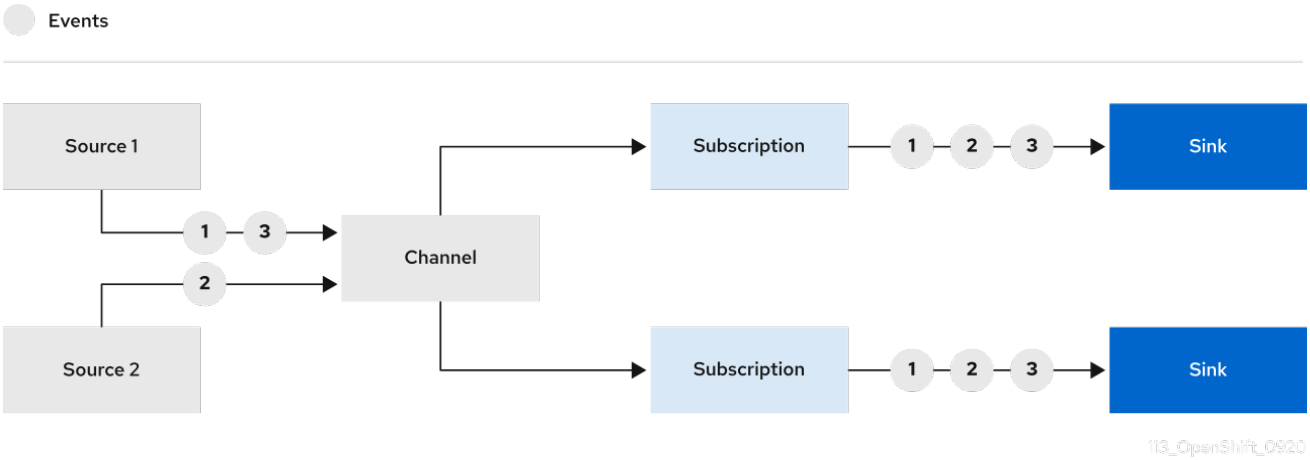
输出示例

```
No triggers found.
```

第 6 章 CHANNELS

6.1. 频道和订阅

频道是定义单一事件转发和持久层的自定义资源。事件源或生成程序在将事件发送到频道后，可使用订阅将这些事件发送到多个 Knative 服务或其他 sink。

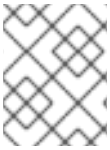


您可以通过实例化受支持的 **Channel** 对象来创建频道，并通过修改 **Subscription** 对象中的 **delivery** 规格来配置重新发送尝试。

创建 **Channel** 对象后，根据默认频道实现，一个经过更改的准入 Webhook 会为 **Channel** 对象添加一组 **spec.channelTemplate** 属性。例如，对于 **InMemoryChannel** 默认实现，**Channel** 对象如下所示：

```
apiVersion: messaging.knative.dev/v1
kind: Channel
metadata:
  name: example-channel
  namespace: default
spec:
  channelTemplate:
    apiVersion: messaging.knative.dev/v1
    kind: InMemoryChannel
```

然后，频道控制器将根据这个 **spec.channelTemplate** 配置创建后备频道实例。



注意

创建后，**spec.channelTemplate** 属性将无法更改，因为它们由默认频道机制设置，而不是由用户设置。

当此机制与上例搭配使用时，会创建两个对象：一个通用的后备频道和一个 **InMemoryChannel** 频道。如果您使用不同的默认频道实现，使用特定于您的实现的频道替换 **InMemoryChannel**。例如，在 Apache Kafka 的 Knative 代理中，会创建 **KafkaChannel** 频道。

后备频道充当将订阅复制到用户创建的频道对象的代理，并设置用户创建的频道对象状态来反映后备频道的状态。

6.1.1. 频道实现类型

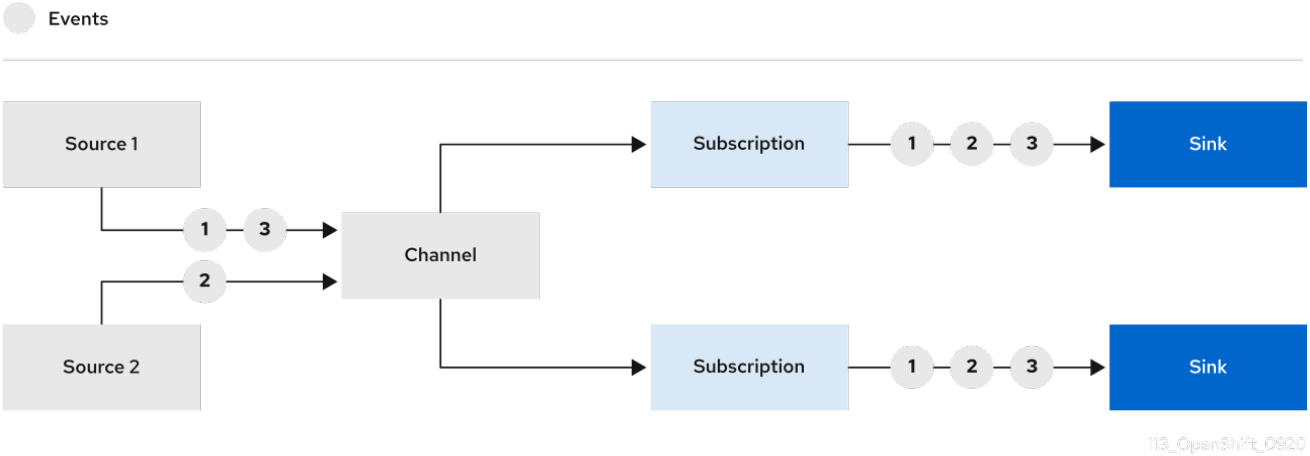
InMemoryChannel 和 **KafkaChannel** 频道实现可用于 OpenShift Serverless 进行开发。

以下是 **InMemoryChannel** 类型频道的限制：

- 事件没有持久性。如果 Pod 停机，则 Pod 上的事件将会丢失。
- **InMemoryChannel** 频道没有实现事件排序，因此同时接收到的两个事件可能会以任何顺序传送给订阅者。
- 如果订阅者拒绝某个事件，则不会默认重新发送尝试。您可以通过修改 **Subscription** 对象中的 **delivery** 规格来配置重新发送尝试。

6.2. 创建频道

频道是定义单一事件转发和持久层的自定义资源。事件源或生成程序在将事件发送到频道后，可使用订阅将这些事件发送到多个 Knative 服务或其他 sink。



您可以通过实例化受支持的 **Channel** 对象来创建频道，并通过修改 **Subscription** 对象中的 **delivery** 规格来配置重新发送尝试。

6.2.1. 使用 Administrator 视角创建频道

在集群中安装 Knative Eventing 后，您可以使用 Administrator 视角创建频道。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 您已登录到 Web 控制台，且处于 **Administrator** 视角。
- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Administrator** 视角中，导航到 **Serverless → Eventing**。
2. 在 **Create** 列表中，选择 **Channel**。您将被定向到 **Channel** 页。

3. 选择您要在 **Type** 列表中创建的 **Channel** 对象类型。

注意

目前默认只支持 **InMemoryChannel** 频道对象。如果您在 OpenShift Serverless 上安装了 Apache Kafka 的 Knative 代理实现，则 Apache Kafka 的 Knative 频道可用。

4. 点 **Create**。

6.2.2. 使用 Developer 视角创建频道

使用 OpenShift Container Platform Web 控制台提供了一个简化的用户界面来创建频道。在集群中安装 Knative Eventing 后，您可以使用 web 控制台创建频道。

先决条件

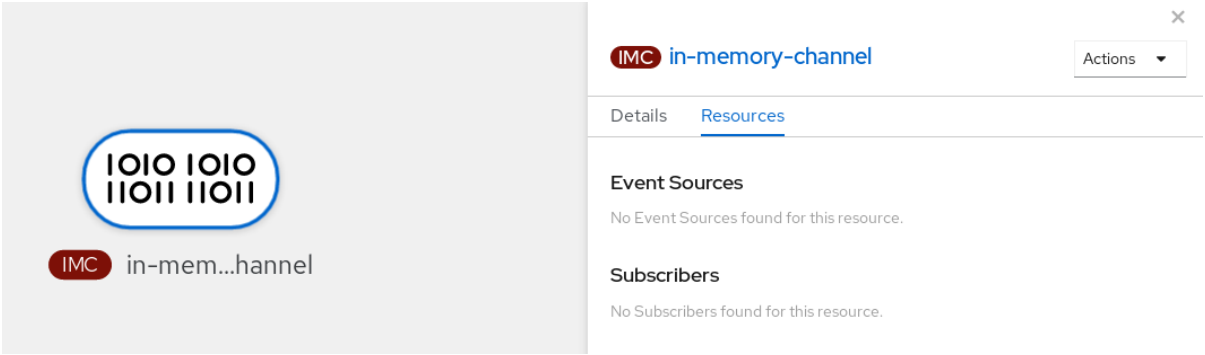
- 已登陆到 OpenShift Container Platform Web 控制台。
- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 在 **Developer** 视角中，导航到 **+Add → Channel**。
2. 选择您要在 **Type** 列表中创建的 **Channel** 对象类型。
3. 点 **Create**。

验证

- 通过导航到 **Topology** 页面确认频道现在存在。



6.2.3. 使用 Knative CLI 创建频道

使用 Knative (**kn**) 创建频道提供了比直接修改 YAML 文件更精简且直观的用户界面。您可以使用 **kn channel create** 命令创建频道。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。
- 已安装 Knative (**kn**) CLI。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

- 创建频道：

```
$ kn channel create <channel_name> --type <channel_type>
```

频道类型是可选的，但如果指定，则必须使用 **Group:Version:Kind** 格式。例如，您可以创建一个 **InMemoryChannel** 对象：

```
$ kn channel create mychannel --type messaging.knative.dev:v1:InMemoryChannel
```

输出示例

```
Channel 'mychannel' created in namespace 'default'.
```

验证

- 要确认该频道现在存在，请列出现有频道并检查输出：

```
$ kn channel list
```

输出示例

```
kn channel list
NAME TYPE URL AGE READY REASON
mychannel InMemoryChannel http://mychannel-kn-channel.default.svc.cluster.local 93s
True
```

删除频道

- 删除频道：

```
$ kn channel delete <channel_name>
```

6.2.4. 使用 YAML 创建默认实现频道

使用 YAML 文件创建 Knative 资源使用声明性 API，它允许您以声明性的方式描述频道，并以可重复的方式描述频道。要使用 YAML 创建无服务器频道，您必须创建一个 YAML 文件来定义 **Channel** 对象，然后使用 **oc apply** 命令应用它。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。
- 安装 OpenShift CLI (**oc**) 。

- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 创建一个 **Channel** 对象作为一个 YAML 文件：

```
apiVersion: messaging.knative.dev/v1
kind: Channel
metadata:
  name: example-channel
  namespace: default
```

2. 应用 YAML 文件：

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

6.2.5. 使用 YAML 为 Apache Kafka 创建频道

使用 YAML 文件创建 Knative 资源使用声明性 API，它允许您以声明性的方式描述频道，并以可重复的方式描述频道。您可以通过创建一个 Kafka 频道，创建由 Kafka 主题支持的 Knative Eventing 频道。要使用 YAML 创建 Kafka 频道，您必须创建一个 YAML 文件来定义 **KafkaChannel** 对象，然后使用 **oc apply** 命令应用它。

先决条件

- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** 自定义资源已安装在 OpenShift Container Platform 集群中。
- 安装 OpenShift CLI (**oc**) 。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 创建一个 **KafkaChannel** 对象作为一个 YAML 文件：

```
apiVersion: messaging.knative.dev/v1beta1
kind: KafkaChannel
metadata:
  name: example-channel
  namespace: default
spec:
  numPartitions: 3
  replicationFactor: 1
```

重要

仅支持 OpenShift Serverless 上的 **KafkaChannel** 对象的 **v1beta1** API 版本。不要使用这个 API 的 **v1alpha1** 版本，因为这个版本现已弃用。

2. 应用 **KafkaChannel** YAML 文件：

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

6.2.6. 后续步骤

- 创建频道后，您可以 将频道连接到 sink，以便 sink 可以接收事件。
- 配置事件交付参数，当事件无法发送到事件 sink 时。

6.3. 将频道连接到 SINK

来自事件源或制作者发送到频道的事件可使用 *订阅* 转发到一个或多个 sink。您可以通过配置 **Subscription** 对象来创建订阅，它指定消耗发送到该频道的事件的频道和接收器（也称为 *订阅者*）。

6.3.1. 使用 Developer 视角创建订阅

创建频道和事件 sink 后，您可以创建一个订阅来启用事件交付。使用 OpenShift Container Platform Web 控制台提供了一个简化且直观的用户界面来创建订阅。

先决条件

- OpenShift Serverless Operator、Knative Serving 和 Knative Eventing 已在 OpenShift Container Platform 集群中安装。
- 已登陆到 web 控制台。
- 您已创建了事件 sink，如 Knative 服务以及频道。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 在 **Developer** 视角中，进入 **Topology** 页。
2. 使用以下方法之一创建订阅：
   a. 将鼠标悬停在您要为其创建订阅的频道上，并拖动箭头。此时会显示 **Add Subscription** 选项。

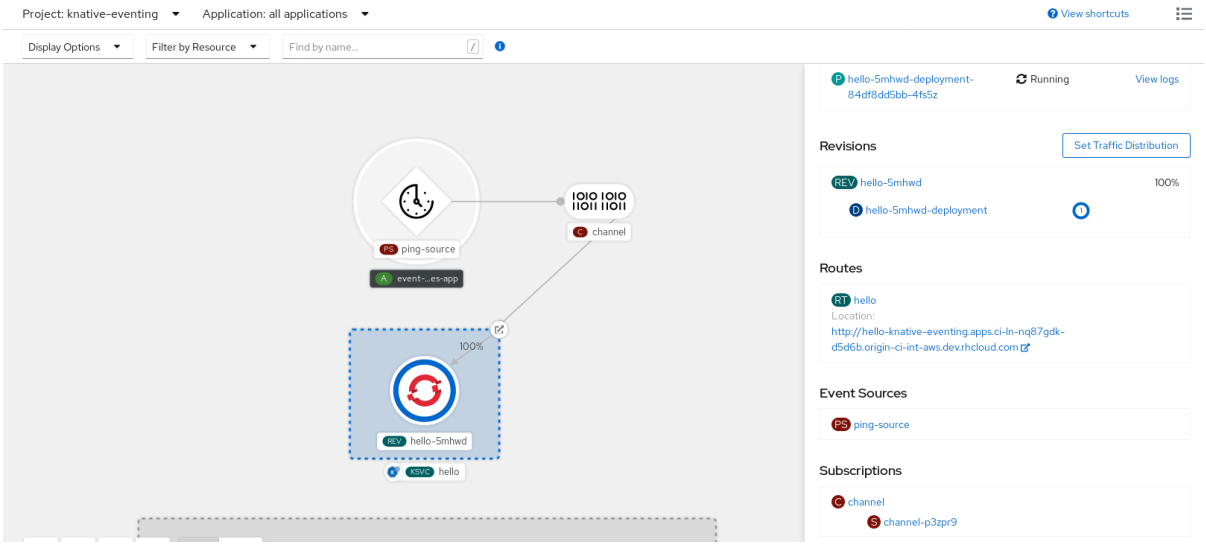
   i. 在 **Subscriber** 列表中选择您的接收器。

ii. 点 **Add**。

b. 如果服务在与频道相同的命名空间或项目下的 **Topology** 视图中可用，点击您要为该频道创建订阅的频道，并将箭头直接拖到服务以立即从频道创建订阅到该服务。

验证

- 创建订阅后，您可以在 **Topology** 视图中将频道连接到该服务的行显示为：



6.3.2. 使用 YAML 创建订阅

创建频道和事件 sink 后，您可以创建一个订阅来启用事件交付。使用 YAML 文件创建 Knative 资源使用声明性 API，它允许您以声明性的方式描述订阅，并以可重复的方式描述订阅。要使用 YAML 创建订阅，您必须创建一个 YAML 文件来定义 **Subscription** 对象，然后使用 **oc apply** 命令应用它。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。
- 安装 OpenShift CLI (**oc**) 。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

- 创建 **Subscription** 对象：
  - 创建 YAML 文件并将以下示例代码复制到其中：

```
apiVersion: messaging.knative.dev/v1beta1
kind: Subscription
metadata:
  name: my-subscription 1
  namespace: default
spec:
  channel: 2
    apiVersion: messaging.knative.dev/v1beta1
    kind: Channel
    name: example-channel
```

```
  delivery: 3
    deadLetterSink:
      ref:
        apiVersion: serving.knative.dev/v1
        kind: Service
        name: error-handler
  subscriber: 4
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

1. 订阅的名称。
2. 订阅连接的频道的配置设置。
3. 事件交付的配置设置。这会告诉订阅无法发送给订阅者的事件。配置后，消耗的事件会发送到 **deadLetterSink**。事件将被丢弃，不会尝试重新发送该事件，并在系统中记录错误。**deadLetterSink** 的值需要是一个 Destination。
4. 订阅用户的配置设置。这是事件从频道发送的事件 sink。

- 应用 YAML 文件：

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

6.3.3. 使用 Knative CLI 创建订阅

创建频道和事件 sink 后，您可以创建一个订阅来启用事件交付。使用 Knative (**kn**) CLI 创建订阅提供了比直接修改 YAML 文件更精简且直观的用户界面。您可以使用带有适当标志的 **kn subscription create** 命令创建订阅。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 已安装 Knative (**kn**) CLI。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

- 创建订阅以将接收器连接到频道：

```
$ kn subscription create <subscription_name> \
  --channel <group:version:kind>:<channel_name> \ 1
  --sink <sink_prefix>:<sink_name> \ 2
  --sink-dead-letter <sink_prefix>:<sink_name> 3
```

1. **--channel** 指定应处理的云事件的来源。您必须提供频道名称。如果您没有使用由 **Channel** 自定义资源支持的默认 **InMemoryChannel** 频道，您必须为指定频道类型添加

2. **--sink** 指定事件要传送到的目标目的地。默认情况下，**<sink_name>** 解释为此名称的 Knative 服务，与订阅位于同一个命名空间中。您可以使用以下前缀之一指定接收器类型：

   ksvc
   : Knative 服务。

   channel
   : 作为目的地的频道。这里只能引用默认频道类型。

   broker
   : Eventing 代理。

3. 可选：**--sink-dead-letter** 是一个可选标志，可用于指定在无法发送事件时哪些事件应发送到的接收器。如需更多信息，请参阅 OpenShift Serverless *事件交付*文档。

   示例命令

   ```
   $ kn subscription create mysubscription --channel mychannel --sink ksvc:event-display
   ```

   输出示例

   ```
   Subscription 'mysubscription' created in namespace 'default'.
   ```

验证

- 要确认频道已连接到事件接收器或 *subscriber*，使用一个订阅列出现有订阅并检查输出：

  ```
  $ kn subscription list
  ```

  输出示例

  ```
  NAME           CHANNEL           SUBSCRIBER          REPLY   DEAD LETTER SINK   READY   REASON
  mysubscription   Channel:mychannel   ksvc:event-display                         True
  ```

删除订阅

- 删除订阅：

  ```
  $ kn subscription delete <subscription_name>
  ```

6.3.4. 使用 Administrator 视角创建订阅

创建频道和事件 sink（也称为 *订阅者*）后，您可以创建一个订阅来启用事件交付。订阅是通过配置 **Subscription** 对象创建的，它指定了要向其发送事件的频道和订阅者。您还可以指定一些特定于订阅者的选项，比如如何处理失败。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 您已登录到 Web 控制台，且处于 **Administrator** 视角。
- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。
- 您已创建了 Knative 频道。
- 您已创建了 Knative 服务以用作订阅者。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Administrator** 视角中，导航到 **Serverless → Eventing**。
2. 在 **Channel** 选项卡中，选择您要在其中添加订阅的频道的 Options 菜单  。
3. 点击列表中的 **Add Subscription**。
4. 在 **Add Subscription** 对话框中，为订阅选择 **Subscriber**。订阅者是可以从频道接收事件的 Knative 服务。
5. 点 **Add**。

6.3.5. 后续步骤

- 配置事件交付参数，当事件无法发送到事件 sink 时。

6.4. 默认频道实现

default-ch-webhook 配置映射可以用来指定 Knative Eventing 的默认频道实现。您可以为整个集群或一个或多个命名空间指定默认频道实现。目前支持 **InMemoryChannel** 和 **KafkaChannel** 频道类型。

6.4.1. 配置默认频道实施

先决条件

- 在 OpenShift Container Platform 上具有管理员权限。
- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。
- 如果要将 Apache Kafka 的 Knative 频道用作默认频道实现，还必须在集群中安装 **KnativeKafka** CR。

流程

- 修改 **KnativeEventing** 自定义资源，以添加 **default-ch-webhook** 配置映射的配置详情：

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
```

```
  name: knative-eventing
  namespace: knative-eventing
spec:
  config: (1)
    default-ch-webhook: (2)
      default-ch-config: |
        clusterDefault: (3)
          apiVersion: messaging.knative.dev/v1
          kind: InMemoryChannel
          spec:
            delivery:
              backoffDelay: PT0.5S
              backoffPolicy: exponential
              retry: 5
        namespaceDefaults: (4)
          my-namespace:
            apiVersion: messaging.knative.dev/v1beta1
            kind: KafkaChannel
            spec:
              numPartitions: 1
              replicationFactor: 1
```

1. 在 **spec.config** 中，您可以指定您要为修改的配置添加的配置映射。
2. **default-ch-webhook** 配置映射可以用来指定集群的默认频道实施，也可以用于一个或多个命名空间。
3. 集群范围的默认频道类型配置。在本例中，集群的默认频道实现是 **InMemoryChannel**。
4. 命名空间范围的默认频道类型配置。在本例中，**my-namespace** 命名空间的默认频道实现是 **KafkaChannel**。

重要

配置特定于命名空间的默认设置会覆盖任何集群范围的设置。

6.5. 频道的安全配置

6.5.1. 为 Apache Kafka 的 Knative 频道配置 TLS 身份验证

Apache Kafka 客户端和服务器使用 *传输层安全性* (TLS) 来加密 Knative 和 Kafka 之间的流量，以及用于身份验证。TLS 是 Apache Kafka 的 Knative 代理实现唯一支持的流量加密方法。

先决条件

- 在 OpenShift Container Platform 上具有集群或专用管理员权限。
- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** CR 已安装在 OpenShift Container Platform 集群中。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

- 您有一个 Kafka 集群 CA 证书存储为一个 **.pem** 文件。
- 您有一个 Kafka 集群客户端证书，并存储为 **.pem** 文件的密钥。
- 安装 OpenShift CLI (**oc**) 。

流程

1. 在所选命名空间中创建证书文件作为 secret：

```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
  --from-file=ca.crt=caroot.pem \
  --from-file=user.crt=certificate.pem \
  --from-file=user.key=key.pem
```

重要

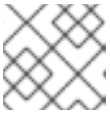
使用密钥名称 **ca.crt**、**user.crt** 和 **user.key**。不要更改它们。

2. 编辑 **KnativeKafka** 自定义资源：

```
$ oc edit knativekafka
```

3. 引用您的 secret 和 secret 的命名空间：

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  namespace: knative-eventing
  name: knative-kafka
spec:
  channel:
    authSecretName: <kafka_auth_secret>
    authSecretNamespace: <kafka_auth_secret_namespace>
    bootstrapServers: <bootstrap_servers>
    enabled: true
  source:
    enabled: true
```



注意

确保指定 bootstrap 服务器中的匹配端口。

例如：

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  namespace: knative-eventing
  name: knative-kafka
spec:
  channel:
    authSecretName: tls-user
```

```
    authSecretNamespace: kafka
    bootstrapServers: eventing-kafka-bootstrap.kafka.svc:9094
    enabled: true
  source:
    enabled: true
```

6.5.2. 为 Apache Kafka 的 Knative 频道配置 SASL 身份验证

Apache Kafka 使用 *简单身份验证和安全层* (SASL) 进行身份验证。如果在集群中使用 SASL 身份验证，用户则必须向 Knative 提供凭证才能与 Kafka 集群通信，否则无法生成或消耗事件。

先决条件

- 在 OpenShift Container Platform 上具有集群或专用管理员权限。
- OpenShift Serverless Operator、Knative Eventing 和 **KnativeKafka** CR 已安装在 OpenShift Container Platform 集群中。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您有一个 Kafka 集群的用户名和密码。
- 您已选择使用 SASL 机制，例如 **PLAIN**、**SCRAM-SHA-256** 或 **SCRAM-SHA-512**。
- 如果启用了 TLS，您还需要 Kafka 集群的 **ca.crt** 证书文件。
- 安装 OpenShift CLI (**oc**) 。

流程

1. 在所选命名空间中创建证书文件作为 secret：

```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
  --from-file=ca.crt=caroot.pem \
  --from-literal=password="SecretPassword" \
  --from-literal=saslType="SCRAM-SHA-512" \
  --from-literal=user="my-sasl-user"
```

   - 使用键名 **ca.crt**, **password**, 和 **sasl.mechanism**.不要更改它们。
   - 如果要将 SASL 与公共 CA 证书搭配使用，您必须在创建 secret 时使用 **tls.enabled=true** 标志，而不是使用 **ca.crt** 参数。例如：

```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
  --from-literal=tls.enabled=true \
  --from-literal=password="SecretPassword" \
  --from-literal=saslType="SCRAM-SHA-512" \
  --from-literal=user="my-sasl-user"
```

2. 编辑 **KnativeKafka** 自定义资源：

```
$ oc edit knativekafka
```

3. 引用您的 secret 和 secret 的命名空间：

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  namespace: knative-eventing
  name: knative-kafka
spec:
  channel:
    authSecretName: <kafka_auth_secret>
    authSecretNamespace: <kafka_auth_secret_namespace>
    bootstrapServers: <bootstrap_servers>
    enabled: true
  source:
    enabled: true
```

注意

确保指定 bootstrap 服务器中的匹配端口。

例如：

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  namespace: knative-eventing
  name: knative-kafka
spec:
  channel:
    authSecretName: scram-user
    authSecretNamespace: kafka
    bootstrapServers: eventing-kafka-bootstrap.kafka.svc:9093
    enabled: true
  source:
    enabled: true
```

第 7 章 订阅

7.1. 创建订阅

创建频道和事件 sink 后，您可以创建一个订阅来启用事件交付。订阅是通过配置 **Subscription** 对象创建的，它指定频道和接收器（也称为 *订阅者*）来发送事件。

7.1.1. 使用 Administrator 视角创建订阅

创建频道和事件 sink（也称为 *订阅者*）后，您可以创建一个订阅来启用事件交付。订阅是通过配置 **Subscription** 对象创建的，它指定了要向其发送事件的频道和订阅者。您还可以指定一些特定于订阅者的选项，比如如何处理失败。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 您已登录到 Web 控制台，且处于 **Administrator** 视角。
- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。
- 您已创建了 Knative 频道。
- 您已创建了 Knative 服务以用作订阅者。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Administrator** 视角中，导航到 **Serverless → Eventing**。
2. 在 **Channel** 选项卡中，选择您要在其中添加订阅的频道的 Options 菜单  。
3. 点击列表中的 **Add Subscription**。
4. 在 **Add Subscription** 对话框中，为订阅选择 **Subscriber**。订阅者是可以从频道接收事件的 Knative 服务。
5. 点 **Add**。

7.1.2. 使用 Developer 视角创建订阅

创建频道和事件 sink 后，您可以创建一个订阅来启用事件交付。使用 OpenShift Container Platform Web 控制台提供了一个简化且直观的用户界面来创建订阅。

先决条件

- OpenShift Serverless Operator、Knative Serving 和 Knative Eventing 已在 OpenShift Container Platform 集群中安装。
- 已登陆到 web 控制台。

- 您已创建了事件 sink，如 Knative 服务以及频道。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

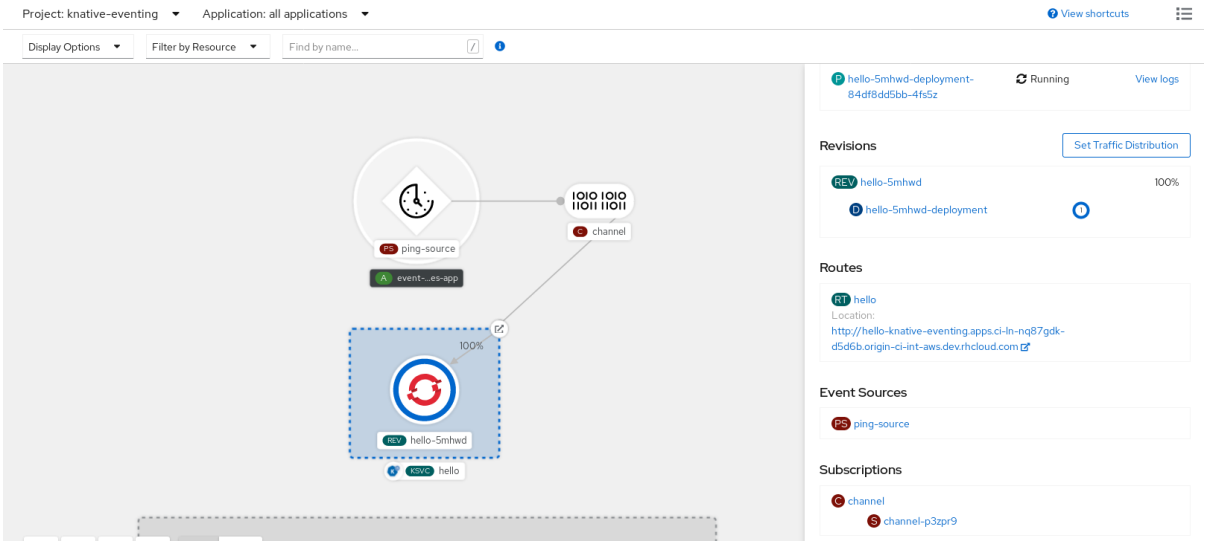
1. 在 **Developer** 视角中，进入 **Topology** 页。
2. 使用以下方法之一创建订阅：
   a. 将鼠标悬停在您要为其创建订阅的频道上，并拖动箭头。此时会显示 **Add Subscription** 选项。

      i. 在 **Subscriber** 列表中选择您的接收器。
      ii. 点 **Add**。

   b. 如果服务在与频道相同的命名空间或项目下的 **Topology** 视图中可用，点击您要为该频道创建订阅的频道，并将箭头直接拖到服务以立即从频道创建订阅到该服务。

验证

- 创建订阅后，您可以在 **Topology** 视图中将频道连接到该服务的行显示为：



7.1.3. 使用 YAML 创建订阅

创建频道和事件 sink 后，您可以创建一个订阅来启用事件交付。使用 YAML 文件创建 Knative 资源使用声明性 API，它允许您以声明性的方式描述订阅，并以可重复的方式描述订阅。要使用 YAML 创建订阅，您必须创建一个 YAML 文件来定义 **Subscription** 对象，然后使用 **oc apply** 命令应用它。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。
- 安装 OpenShift CLI (**oc**) 。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

- 创建 **Subscription** 对象：
  - 创建 YAML 文件并将以下示例代码复制到其中：

```
apiVersion: messaging.knative.dev/v1beta1
kind: Subscription
metadata:
  name: my-subscription (1)
  namespace: default
spec:
  channel: (2)
    apiVersion: messaging.knative.dev/v1beta1
    kind: Channel
    name: example-channel
  delivery: (3)
    deadLetterSink:
      ref:
        apiVersion: serving.knative.dev/v1
        kind: Service
        name: error-handler
  subscriber: (4)
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

(1) 订阅的名称。

(2) 订阅连接的频道的配置设置。

(3) 事件交付的配置设置。这会告诉订阅无法发送给订阅者的事件。配置后，消耗的事件会发送到 **deadLetterSink**。事件将被丢弃，不会尝试重新发送该事件，并在系统中记录错误。**deadLetterSink** 的值需要是一个 Destination。

(4) 订阅用户的配置设置。这是事件从频道发送的事件 sink。

  - 应用 YAML 文件：

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

7.1.4. 使用 Knative CLI 创建订阅

创建频道和事件 sink 后，您可以创建一个订阅来启用事件交付。使用 Knative (**kn**) CLI 创建订阅提供了比直接修改 YAML 文件更精简且直观的用户界面。您可以使用带有适当标志的 **kn subscription create** 命令创建订阅。

先决条件

- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 已安装 Knative (**kn**) CLI。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

- 创建订阅以将接收器连接到频道：

```
$ kn subscription create <subscription_name> \
  --channel <group:version:kind>:<channel_name> \ 1
  --sink <sink_prefix>:<sink_name> \ 2
  --sink-dead-letter <sink_prefix>:<sink_name> 3
```

1. **--channel** 指定应处理的云事件的来源。您必须提供频道名称。如果您没有使用由 **Channel** 自定义资源支持的默认 **InMemoryChannel** 频道，您必须为指定频道类型添加 **<group:version:kind>** 前缀。例如，这是 Apache Kafka 支持的频道的 **messaging.knative.dev:v1beta1:KafkaChannel**。

2. **--sink** 指定事件要传送到的目标目的地。默认情况下，**<sink_name>** 解释为此名称的 Knative 服务，与订阅位于同一个命名空间中。您可以使用以下前缀之一指定接收器类型：

   ksvc
   : Knative 服务。

   channel
   : 作为目的地的频道。这里只能引用默认频道类型。

   broker
   : Eventing 代理。

3. 可选：**--sink-dead-letter** 是一个可选标志，可用于指定在无法发送事件时哪些事件应发送到的接收器。如需更多信息，请参阅 OpenShift Serverless *事件交付*文档。

   示例命令

   ```
   $ kn subscription create mysubscription --channel mychannel --sink ksvc:event-display
   ```

   输出示例

   ```
   Subscription 'mysubscription' created in namespace 'default'.
   ```

验证

- 要确认频道已连接到事件接收器或 *subscriber*，使用一个订阅列出现有订阅并检查输出：

```
$ kn subscription list
```

输出示例

```
NAME           CHANNEL           SUBSCRIBER          REPLY   DEAD LETTER SINK
READY   REASON
mysubscription   Channel:mychannel   ksvc:event-display                      True
```

删除订阅

- 删除订阅：

```
$ kn subscription delete <subscription_name>
```

7.1.5. 后续步骤

- 配置事件交付参数，当事件无法发送到事件 sink 时。

7.2. 管理订阅

7.2.1. 使用 Knative CLI 描述订阅

您可以使用 **kn subscription describe** 命令在终端中使用 Knative (**kn**) 打印有关订阅的信息。使用 Knative CLI 描述订阅可提供比直接查看 YAML 文件更精简且直观的用户界面。

先决条件

- 已安装 Knative (**kn**) CLI。
- 您已在集群中创建了订阅。

流程

- 描述订阅：

```
$ kn subscription describe <subscription_name>
```

输出示例

```
Name:            my-subscription
Namespace:       default
Annotations:     messaging.knative.dev/creator=openshift-user,
messaging.knative.dev/lastModifier=min ...
Age:             43s
Channel:         Channel:my-channel (messaging.knative.dev/v1)
Subscriber:
  URI:           http://edisplay.default.example.com
Reply:
  Name:          default
  Resource:      Broker (eventing.knative.dev/v1)
```

```
DeadLetterSink:
  Name:         my-sink
  Resource:     Service (serving.knative.dev/v1)

Conditions:
  OK TYPE                  AGE REASON
  ++ Ready                 43s
  ++ AddedToChannel        43s
  ++ ChannelReady          43s
  ++ ReferencesResolved    43s
```

7.2.2. 使用 Knative CLI 列出订阅

您可以使用 **kn subscription list** 命令通过 Knative (**kn**) CLI 列出集群中的现有订阅。使用 Knative CLI 列出订阅提供了精简且直观的用户界面。

先决条件

- 已安装 Knative (**kn**) CLI。

流程

- 列出集群中的订阅：

```
$ kn subscription list
```

输出示例

```
NAME            CHANNEL            SUBSCRIBER          REPLY   DEAD LETTER SINK   READY   REASON
mysubscription   Channel:mychannel   ksvc:event-display                             True
```

7.2.3. 使用 Knative CLI 更新订阅

您可以使用 **kn subscription update** 命令以及使用 Knative (**kn**) CLI 从终端更新订阅的适当标志。使用 Knative CLI 更新订阅可提供比直接更新 YAML 文件更精简且直观的用户界面。

先决条件

- 已安装 Knative (**kn**) CLI。
- 您已创建了订阅。

流程

- 更新订阅：

```
$ kn subscription update <subscription_name> \
  --sink <sink_prefix>:<sink_name> \ 1
  --sink-dead-letter <sink_prefix>:<sink_name> 2
```

1 **--sink** 指定要将事件传送到的更新目标目的地。您可以使用以下前缀之一指定接收器类型：

ksvc
: Knative 服务。

channel
: 作为目的地的频道。这里只能引用默认频道类型。

broker
: Eventing 代理。

2 可选： **--sink-dead-letter** 是一个可选标志，可用于指定在无法发送事件时哪些事件应发送到的接收器。如需更多信息，请参阅 OpenShift Serverless *事件交付*文档。

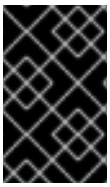
示例命令

```
$ kn subscription update mysubscription --sink ksvc:event-display
```

第 8 章 事件交付

您可以配置事件交付参数，当事件无法发送到事件 sink 时。不同的频道和代理类型都有自己的行为模式，用于事件交付。

配置事件交付参数，包括死信接收器，可确保重试任何无法发送到事件接收器的事件。否则，未验证的事件将被丢弃。



重要

如果事件成功传送到 Apache Kafka 的频道或代理接收器，接收器会使用 **202** 状态代码进行响应，这意味着事件已被安全地存储在 Kafka 主题中，且不会丢失。如果接收器使用任何其他状态代码响应，则事件不会被安全存储，用户必须执行步骤来解决这个问题。

8.1. 可配置事件交付参数

为事件交付配置以下参数：

死信接收器

您可以配置 **deadLetterSink** 交付参数，以便在事件无法发送时，它存储在指定的事件 sink 中。取消请求没有存储在死信接收器中的事件会被丢弃。死信接收器是符合 Knative Eventing sink 合同的任何可寻址对象，如 Knative 服务、Kubernetes 服务或一个 URI。

Retries

您可以通过使用整数值配置重试 delivery 参数，在事件发送到 dead letter sink 前重试交付的次数。

Back off 延迟

您可以设置 **backoffDelay** 交付参数，以在失败后尝试事件交付重试前指定延迟。**backoffDelay** 参数的持续时间使用 ISO 8601 格式指定。例如，**PT1S** 指定 1 秒延迟。

Back off 策略

backoffPolicy 交付参数可以用来指定重试避退策略。该策略可以指定为 **linear** 或 **exponential**。当使用 **linear** back off 策略时，back off 延迟等同于 **backoffDelay * <numberOfRetries>**。当使用 **exponential** backoff 策略时，back off 的延迟等于 **backoffDelay*2^<numberOfRetries>**。

8.2. 配置事件交付参数示例

您可以为 **Broker**、**Trigger**、**Channel** 和 **Subscription** 对象配置事件交付参数。如果您为代理或频道配置事件交付参数，这些参数会传播到为这些对象创建的触发器或订阅。您还可以为触发器或订阅设置事件交付参数，以覆盖代理或频道的设置。

Broker 对象示例

```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
# ...
spec:
  delivery:
    deadLetterSink:
      ref:
        apiVersion: eventing.knative.dev/v1alpha1
        kind: KafkaSink
        name: <sink_name>
    backoffDelay: <duration>
```

```
    backoffPolicy: <policy_type>
    retry: <integer>
# ...
```

Trigger 对象示例

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
# ...
spec:
  broker: <broker_name>
  delivery:
    deadLetterSink:
      ref:
        apiVersion: serving.knative.dev/v1
        kind: Service
        name: <sink_name>
    backoffDelay: <duration>
    backoffPolicy: <policy_type>
    retry: <integer>
# ...
```

Channel 对象示例

```
apiVersion: messaging.knative.dev/v1
kind: Channel
metadata:
# ...
spec:
  delivery:
    deadLetterSink:
      ref:
        apiVersion: serving.knative.dev/v1
        kind: Service
        name: <sink_name>
    backoffDelay: <duration>
    backoffPolicy: <policy_type>
    retry: <integer>
# ...
```

Subscription 对象示例

```
apiVersion: messaging.knative.dev/v1
kind: Subscription
metadata:
# ...
spec:
  channel:
    apiVersion: messaging.knative.dev/v1
    kind: Channel
    name: <channel_name>
  delivery:
    deadLetterSink:
```

```
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: <sink_name>
  backoffDelay: <duration>
  backoffPolicy: <policy_type>
  retry: <integer>
# ...
```

8.3. 为触发器配置事件交付顺序

如果使用 Kafka 代理，您可以将事件的交付顺序从触发器配置为事件 sink。

先决条件

- OpenShift Serverless Operator、Knative Eventing 和 Knative Kafka 安装在 OpenShift Container Platform 集群中。
- Kafka 代理被启用在集群中使用，您也创建了一个 Kafka 代理。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 已安装 OpenShift (**oc**) CLI。

流程

1. 创建或修改 **Trigger** 对象并设置 **kafka.eventing.knative.dev/delivery.order** 注解：

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: <trigger_name>
  annotations:
    kafka.eventing.knative.dev/delivery.order: ordered
# ...
```

支持的消费者交付保证有：

unordered

未排序的消费者是一种非阻塞消费者，它能以未排序的方式提供消息，同时保持正确的偏移管理。

排序的

一个订购的消费者是一个按分区阻止消费者，在提供分区的下一个消息前等待来自 CloudEvent 订阅者成功响应。
默认排序保证是 **unordered**。

2. 应用 **Trigger** 对象：

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

第 9 章 事件发现

9.1. 列出事件源和事件源类型

可以查看存在的事件源或事件源类型的列表，也可以在 OpenShift Container Platform 集群中使用。您可以使用 OpenShift Container Platform Web 控制台中的 Knative (**kn**) CLI 或 **Developer** 视角列出可用事件源或事件源类型。

9.2. 从命令行列出事件源类型

使用 Knative (**kn**) CLI 提供了简化和直观的用户界面，用来在集群中查看可用事件源类型。

9.2.1. 使用 Knative CLI 列出可用事件源类型

您可以使用 **kn source list-types** CLI 命令列出集群中创建和使用的事件源类型。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。
- 已安装 Knative (**kn**) CLI。

流程

1. 列出终端中的可用事件源类型：

```
$ kn source list-types
```

输出示例

```
TYPE              NAME                                    DESCRIPTION
ApiServerSource   apiserversources.sources.knative.dev     Watch and send Kubernetes API events to a sink
PingSource        pingsources.sources.knative.dev         Periodically send ping events to a sink
SinkBinding       sinkbindings.sources.knative.dev        Binding for connecting a PodSpecable to a sink
```

2. 可选：在 OpenShift Container Platform 中，您也可以使用 YAML 格式列出可用事件源类型：

```
$ kn source list-types -o yaml
```

9.3. 从 DEVELOPER 视角列出事件源类型

您可以查看集群中所有可用事件源类型的列表。使用 OpenShift Container Platform Web 控制台提供了一个简化的用户界面，可用事件源类型。

9.3.1. 在 Developer 视角中查看可用事件源类型

先决条件

- 已登陆到 OpenShift Container Platform Web 控制台。
- OpenShift Serverless Operator 和 Knative Eventing 已安装在 OpenShift Container Platform 集群中。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 访问 **Developer** 视角。
2. 点 **+Add**。
3. 点 **Event Source**。
4. 查看可用的事件源类型。

9.4. 从命令行列出事件源

使用 Knative (**kn**) CLI 提供了简化和直观的用户界面，用来查看集群中的现有事件源。

9.4.1. 使用 Knative CLI 列出可用事件源

您可以使用 **kn source list** 命令列出现有的事件源。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。
- 已安装 Knative (**kn**) CLI。

流程

1. 列出终端中的现有事件源：

```
$ kn source list
```

输出示例

```
NAME   TYPE              RESOURCE                              SINK         READY
a1     ApiServerSource   apiserversources.sources.knative.dev   ksvc:eshow2   True
b1     SinkBinding       sinkbindings.sources.knative.dev       ksvc:eshow3   False
p1     PingSource        pingsources.sources.knative.dev        ksvc:eshow1   True
```

2. 可选： 您可以使用 **--type** 标志来只列出特定类型的事件源：

```
$ kn source list --type <event_source_type>
```

示例命令

```
$ kn source list --type PingSource
```

输出示例

```
NAME   TYPE          RESOURCE                          SINK        READY
p1     PingSource    pingsources.sources.knative.dev   ksvc:eshow1   True
```

第 10 章 调优事件配置

10.1. 覆盖 KNATIVE EVENTING 系统部署配置

您可以通过修改 **KnativeEventing** 自定义资源 (CR) 中的 **deployments** spec 来覆盖某些特定部署的默认配置。目前，对于 **eventing-controller**、**eventing-webhook** 和 **imc-controller** 字段以及探测的 **readiness** 和 **liveness** 字段支持覆盖默认配置设置。

重要

replicas spec 无法覆盖使用 Horizontal Pod Autoscaler (HPA) 的部署副本数，且不适用于 **eventing-webhook** 部署。

注意

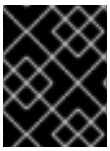
您只能覆盖部署中默认定义的探测。

所有 Knative Serving 部署都会默认定义一个就绪度和存活度探测，但以下例外：

- **net-kourier-controller** 和 **3scale-kourier-gateway** 仅定义就绪度探测。
- **net-istio-controller** 和 **net-istio-webhook** 定义没有探测。

10.1.1. 覆盖部署配置

目前，对于 **eventing-controller**、**eventing-webhook** 和 **imc-controller** 字段以及探测的 **readiness** 和 **liveness** 字段支持覆盖默认配置设置。



重要

replicas spec 无法覆盖使用 Horizontal Pod Autoscaler (HPA) 的部署副本数，且不适用于 **eventing-webhook** 部署。

在以下示例中，**KnativeEventing** CR 覆盖 **eventing-controller** 部署，以便：

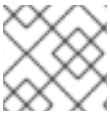
- **readiness** 探测超时 **eventing-controller** 被设置为 10 秒。
- 部署指定了 CPU 和内存资源限制。
- 部署有 3 个副本。
- 添加 **example-label: label** 标签。
- 添加 **example-annotation:** 注解。
- **nodeSelector** 字段被设置为选择带有 **disktype: hdd** 标签的节点。

KnativeEventing CR 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
```

```
  namespace: knative-eventing
spec:
  deployments:
  - name: eventing-controller
    readinessProbes: (1)
      - container: controller
        timeoutSeconds: 10
    resources:
    - container: eventing-controller
      requests:
        cpu: 300m
        memory: 100Mi
      limits:
        cpu: 1000m
        memory: 250Mi
    replicas: 3
    labels:
      example-label: label
    annotations:
      example-annotation: annotation
    nodeSelector:
      disktype: hdd
```

(1) 您可以使用 **readiness** 和 **liveness** 探测覆盖来覆盖在 Kubernetes API 中指定的一个部署中的一个容器探测的所有字段，与探测 handler: **exec**, **grpc**, **httpGet**, 和 **tcpSocket** 相关的字段除外。



注意

KnativeEventing CR 标签和注解设置覆盖部署本身和生成的 Pod 的部署标签和注解。

其他资源

- Kubernetes API 文档中的探测配置部分

10.2. 高可用性

高可用性 (HA) 是 Kubernetes API 的标准功能，有助于确保在出现中断时 API 保持正常运行。在 HA 部署中，如果活跃控制器崩溃或被删除，另一个控制器就可以使用。此控制器会接管处理由现在不可用的控制器提供服务的 API。

OpenShift Serverless 中的 HA 可通过领导选举机制获得，该机制会在安装 Knative Serving 和 Eventing control plane 后默认启用。在使用领导选举 HA 模式时，控制器实例在需要前应该已在集群内调度并运行。这些控制器实例争用共享资源，即领导选举锁定。在任何给定时间可以访问领导选举机制锁定资源的控制器实例被称为领导 (leader) 。

OpenShift Serverless 中的 HA 可通过领导选举机制获得，该机制会在安装 Knative Serving 和 Eventing control plane 后默认启用。在使用领导选举 HA 模式时，控制器实例在需要前应该已在集群内调度并运行。这些控制器实例争用共享资源，即领导选举锁定。在任何给定时间可以访问领导选举机制锁定资源的控制器实例被称为领导 (leader) 。

10.2.1. 为 Knative Eventing 配置高可用性副本

默认情况下，Knative Eventing **eventing-controller**、**eventing-webhook**、**imc-controller**、**imc-dispatcher** 和 **mt-broker-controller** 组件都会具有高可用性 (HA) 。您可以通过修改 **KnativeEventing** 自定义资源 (CR) 中的 **spec.high-availability.replicas** 值来更改这些组件的副本数。

注意

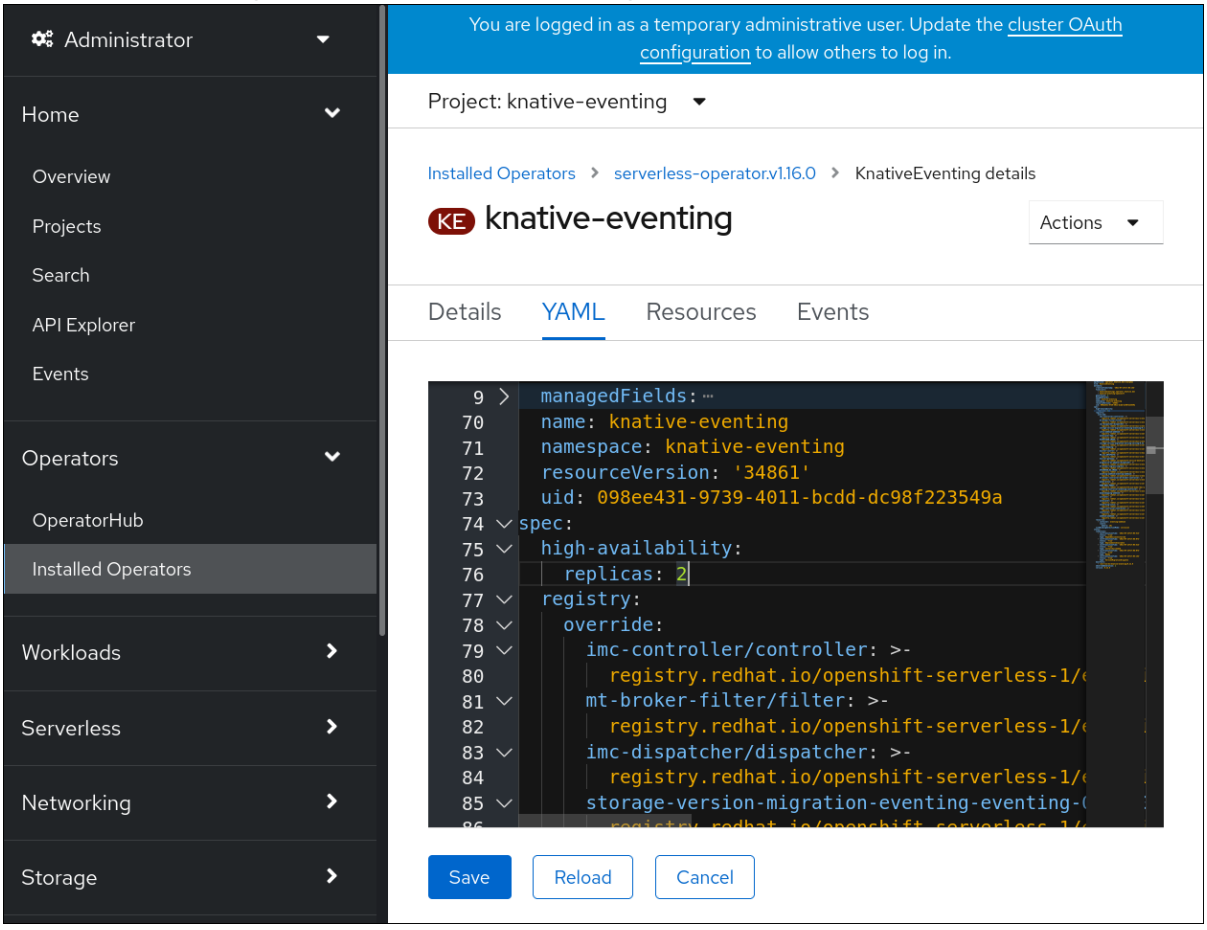
对于 Knative Eventing，HA 不会扩展 **mt-broker-filter** 和 **mt-broker-ingress** 部署。如果需要多个部署，请手动扩展这些组件。

先决条件

- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。
- 在集群中安装了 OpenShift Serverless Operator 和 Knative Eventing。

流程

1. 在 OpenShift Container Platform web 控制台的 **Administrator** 视角中，进入 **OperatorHub → Installed Operators**。
2. 选择 **knative-eventing** 命名空间。
3. 点 OpenShift Serverless Operator 的 **Provided APIs** 列表中的 **Knative Eventing** 来进入 **Knative Eventing** 选项卡。
4. 点 **knative-eventing**，然后进入 **knative-eventing** 页面中的 **YAML** 选项卡。



5. 修改 **KnativeEventing** CR 中的副本数量：

YAML 示例

YAML 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  high-availability:
    replicas: 3
```

10.2.2. 为 Apache Kafka 为 Knative 代理实现配置高可用性副本

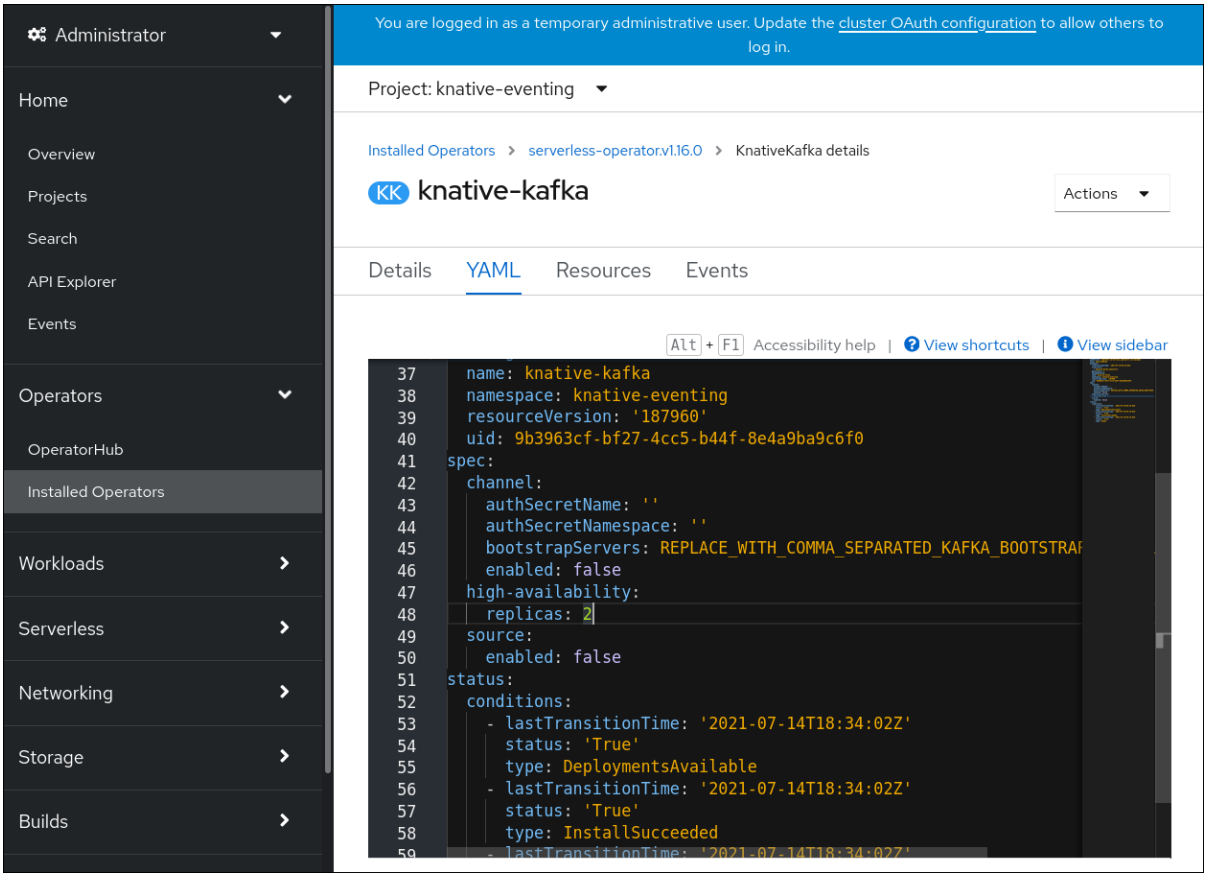
默认情况下，对于 Apache Kafka 组件 **kafka-controller** 和 **kafka-webhook-eventing** 的 Knative 代理实现，高可用性 (HA) 默认为每个副本有两个副本。您可以通过修改 **KnativeKafka** 自定义资源 (CR) 中的 **spec.high-availability.replicas** 值来更改这些组件的副本数。

先决条件

- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。
- 在集群中安装了 OpenShift Serverless Operator 和 Apache Kafka 的 Knative 代理。

流程

1. 在 OpenShift Container Platform web 控制台的 **Administrator** 视角中，进入 **OperatorHub → Installed Operators**。
2. 选择 **knative-eventing** 命名空间。
3. 点 OpenShift Serverless Operator 的 **Provided APIs** 列表中的 **Knative Kafka** 进入 **Knative Kafka** 标签页。
4. 点 **knative-kafka**，然后进入 **knative-kafka** 页面中的 **YAML** 选项卡。



5. 修改 **KnativeKafka** CR 中的副本数量：

YAML 示例

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  name: knative-kafka
  namespace: knative-eventing
spec:
  high-availability:
    replicas: 3
```

第 11 章 为 EVENTING 配置 KUBE-RBAC-PROXY

kube-rbac-proxy 组件为 Knative Eventing 提供内部身份验证和授权功能。

11.1. 为 EVENTING 配置 KUBE-RBAC-PROXY 资源

您可以使用 OpenShift Serverless Operator CR 全局覆盖 **kube-rbac-proxy** 容器的资源分配。

> You can also override resource allocation for a specific deployment.

以下配置设置 Knative Eventing **kube-rbac-proxy** 最小值和最大 CPU 和内存分配：

KnativeEventing CR 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  config:
    deployment:
      "kube-rbac-proxy-cpu-request": "10m" 1
      "kube-rbac-proxy-memory-request": "20Mi" 2
      "kube-rbac-proxy-cpu-limit": "100m" 3
      "kube-rbac-proxy-memory-limit": "100Mi" 4
```

1. 设置最小 CPU 分配。
2. 设置最小 RAM 分配。
3. 设置最大 CPU 分配。
4. 设置最大 RAM 分配。