Red Hat OpenShift Serverless 1.33

Eventing

OpenShift Serverless에서 이벤트 중심 아키텍처 사용

Last Updated: 2024-09-18

Red Hat OpenShift Serverless 1.33 Eventing

OpenShift Serverless에서 이벤트 중심 아키텍처 사용

법적 공지

초록

이 문서에서는 이벤트 소스 및 싱크, 브로커, 트리거, 채널 및 서브스크립션과 같은 Eventing 기능에 대한 정보를 제공합니다.

차례

1장. KNATIVE EVENTING

OpenShift Container Platform에서 Knative Eventing을 사용하면 개발자가 서버리스 애플리케이션에 이벤트 중심 아키텍처를 사용할 수 있습니다. 이벤트 중심 아키텍처는 이벤트 생산자와 이벤트 소비자 간의 분리된 관계에 대한 개념을 기반으로 합니다.

이벤트 생산자는 이벤트를 생성하고 이벤트 *싱크* 또는 소비자를 통해 이벤트를 수신합니다. Knative Eventing은 표준 HTTP POST 요청을 사용하여 이벤트 프로듀서와 싱크 사이에서 이벤트를 전송하고 수신합니다. 이러한 이벤트는 이벤트를 모든 프로그래밍 언어로 생성, 구문 분석, 전송, 수신할 수 있도록 CloudEvents 사양을 준수합니다.

1.1. KNATIVE EVENTING 사용 사례:

Knative Eventing에서는 다음 유스 케이스를 지원합니다.

소비자를 생성하지 않고 이벤트 게시

이벤트를 HTTP POST에 브로커로 보내고 바인딩을 사용하여 이벤트를 생성하는 애플리케이션에서 대상 구성을 분리할 수 있습니다.

게시자를 생성하지 않고 이벤트 사용

트리거를 사용하면 이벤트 특성을 기반으로 브로커의 이벤트를 사용할 수 있습니다. 애플리케이션은 이벤트를 HTTP POST로 수신합니다.

Knative Eventing에서는 다양한 싱크 유형으로 전달할 수 있도록 다음과 같이 여러 Kubernetes 리소스에서 구현할 수 있는 일반 인터페이스를 정의합니다.

주소 지정 가능 리소스

HTTP를 통해 전달되는 이벤트를 이벤트의 **status.address.url** 필드에 정의된 주소로 수신 및 승인할 수 있습니다. Kubernetes **Service** 리소스도 주소 지정 가능 인터페이스의 조건을 충족합니다.

호출 가능한 리소스

HTTP를 통해 전달되는 이벤트를 수신하고 변환하여 HTTP 응답 페이로드에서 **0** 또는 **1**개의 새 이벤트를 반환합니다. 반환된 이벤트는 외부 이벤트 소스의 이벤트를 처리하는 것과 동일한 방식으로 추가로 처리할 수 있습니다.

2장. 이벤트 소스

2.1. 이벤트 소스

Knative *이벤트 소스*는 클라우드 이벤트를 생성하거나 가져오는 모든 Kubernetes 오브젝트일 수 있으며 *싱크*라는 다른 끝점으로 해당 이벤트를 릴레이할 수 있습니다. 이벤트에 대응하는 분산 시스템을 개발하는 데 소싱 이벤트가 중요합니다.

OpenShift Container Platform 웹 콘솔의 **개발자** 화면, Knative(**kn**) CLI를 사용하거나 YAML 파일을 적용하여 Knative 이벤트 소스를 생성하고 관리할 수 있습니다.

현재 OpenShift Serverless에서는 다음 이벤트 소스 유형을 지원합니다.

API 서버 소스

Kubernetes API 서버 이벤트를 Knative로 가져옵니다. API 서버 소스는 Kubernetes 리소스가 생성, 업데이트 또는 삭제될 때마다 새 이벤트를 보냅니다.

ping 소스

지정된 cron 일정에 고정된 페이로드를 사용하여 이벤트를 생성합니다.

Kafka 이벤트 소스

Apache Kafka 클러스터를 이벤트 소스로 싱크에 연결합니다.

사용자 지정 이벤트 소스를 생성할 수도 있습니다.

2.2. 관리자 관점에서 이벤트 소스

이벤트에 대응하는 분산 시스템을 개발하는 데 소싱 이벤트가 중요합니다.

2.2.1. 관리자 화면을 사용하여 이벤트 소스 생성

Knative *이벤트 소스*는 클라우드 이벤트를 생성하거나 가져오는 모든 Kubernetes 오브젝트일 수 있으며 *싱크*라는 다른 끝점으로 해당 이벤트를 릴레이할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 **관리자** 화면에 있습니다.
- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 있습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔의 **관리자** 화면에서 **Serverless → Eventing**으로 이동합니다.
2. **생성** 목록에서 **이벤트 소스**를 선택합니다. 그러면 **이벤트 소스** 페이지로 이동합니다.
3. 생성할 이벤트 소스 유형을 선택합니다.

2.3. API 서버 소스 생성

API 서버 소스는 Knative 서비스와 같은 이벤트 싱크를 Kubernetes API 서버에 연결하는 데 사용할 수 있는 이벤트 소스입니다. API 서버 소스는 Kubernetes 이벤트를 조사하고 해당 이벤트를 Knative Eventing 브로커에 전달합니다.

2.3.1. 웹 콘솔을 사용하여 API 서버 소스 생성

Knative Eventing이 클러스터에 설치되면 웹 콘솔을 사용하여 API 서버 소스를 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔을 사용하면 간소화되고 직관적인 사용자 인터페이스를 통해 이벤트 소스를 생성할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 웹 콘솔에 로그인했습니다.
- OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.

프로세스

기존 서비스 계정을 다시 사용하려면 새 리소스를 생성하는 대신 필요한 권한을 포함하도록 기존 **ServiceAccount** 리소스를 수정할 수 있습니다.

1. 이벤트 소스에 대한 서비스 계정, 역할, 역할 바인딩을 YAML 파일로 만듭니다.

```
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 이 네임스페이스를 이벤트 소스 설치를 위해 선택한 네임스페이스로 변경합니다.

2. YAML 파일을 적용합니다.

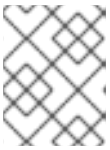
```
$ oc apply -f <filename>
```

3. **개발자** 화면에서 **+추가 → 이벤트 소스**로 이동합니다. **이벤트 소스** 페이지가 표시됩니다.

4. 선택 사항: 이벤트 소스에 대한 공급자가 여러 개인 경우 **공급자** 목록에서 필요한 공급자를 선택하여 해당 공급자의 사용 가능한 이벤트 소스를 필터링합니다.

5. **ApiServerSource**를 선택한 다음 **이벤트 소스 생성**을 클릭합니다. **이벤트 소스 생성** 페이지가 표시됩니다.

6. **양식 보기** 또는 **YAML 보기**를 사용하여 **ApiServerSource** 설정을 구성합니다.



참고

양식 보기와 **YAML 보기**를 전환할 수 있습니다. 다른 보기로 전환해도 데이터는 유지됩니다.

   a. **APIVERSION**으로 **v1**을, **KIND**로 **Event**를 입력합니다.

   b. 생성한 서비스 계정의 **서비스 계정 이름**을 선택합니다.

   c. **대상** 섹션에서 이벤트 싱크를 선택합니다. **리소스** 또는 **URI** 일 수 있습니다.

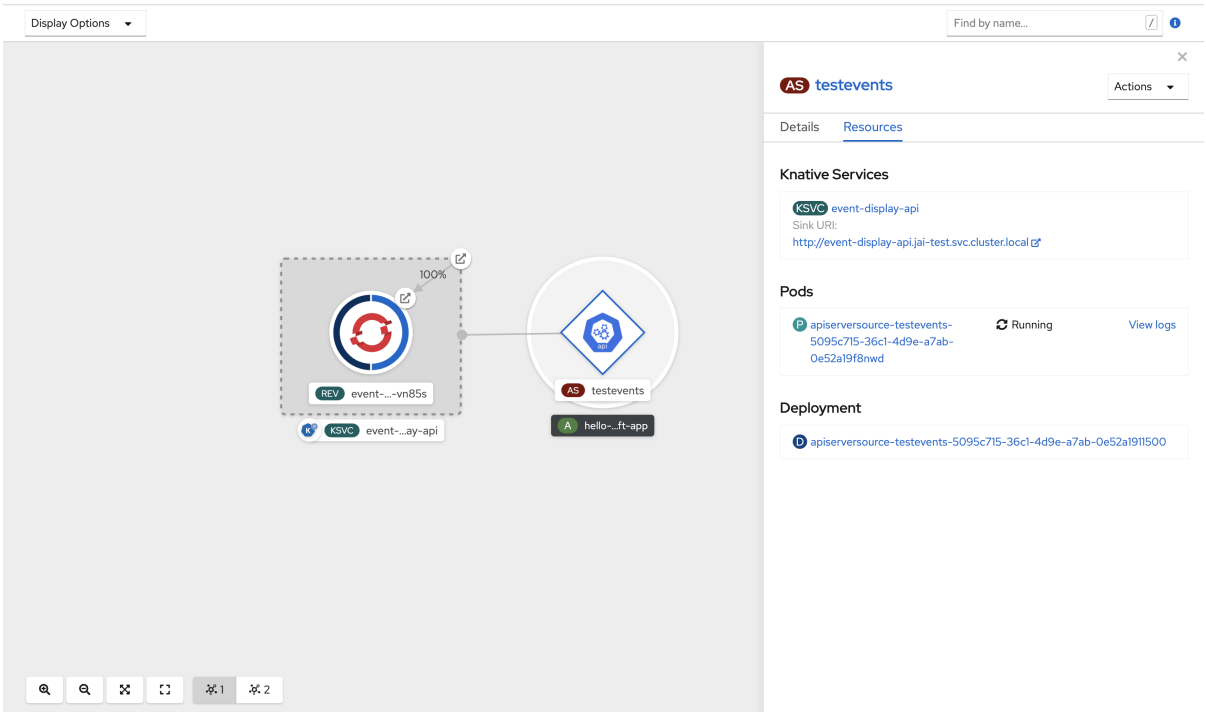
      i. 채널, 브로커 또는 서비스를 이벤트 소스의 이벤트 싱크로 사용하려면 **리소스** 를 선택합니다.

      ii. **URI** 를 선택하여 이벤트가 라우팅되는 URI(Uniform Resource Identifier)를 지정합니다.

7. **생성**을 클릭합니다.

검증

- API 서버 소스를 생성한 후 **토폴로지 보기**에서 확인하여 이벤트 싱크에 연결되어 있는지 확인합니다.

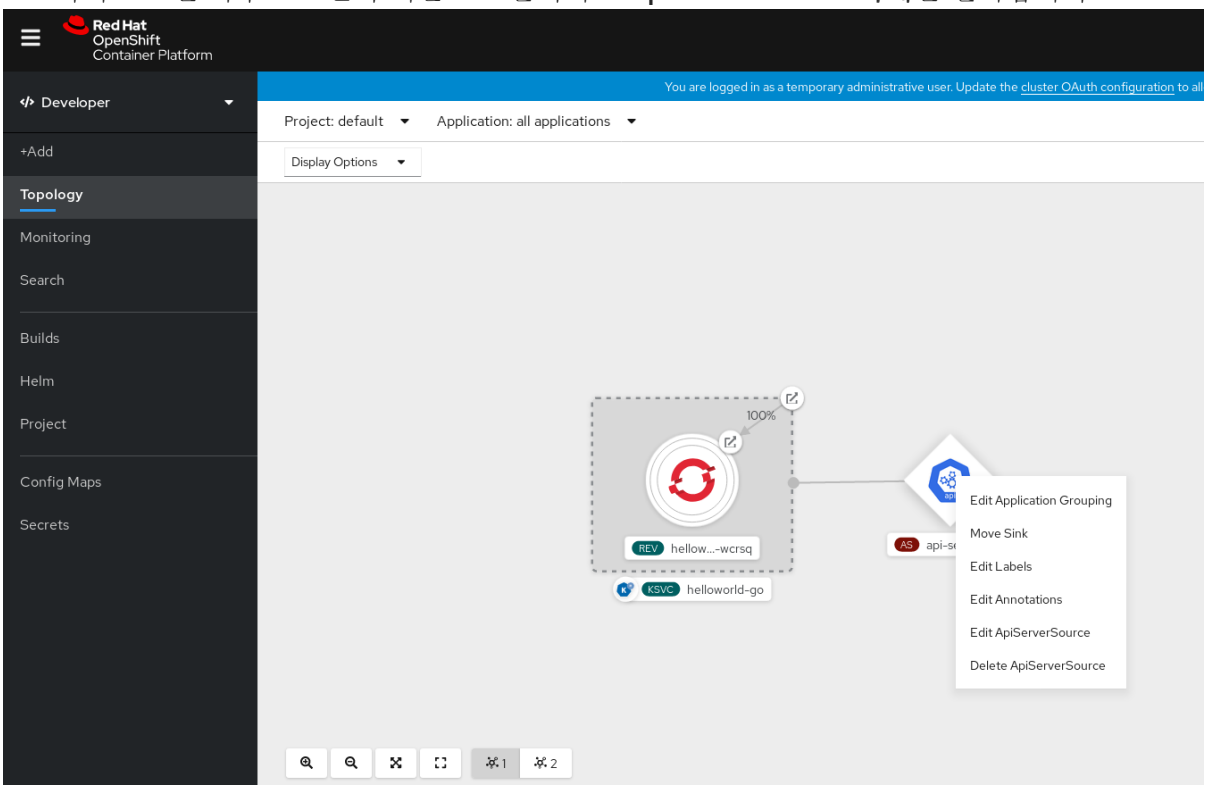


참고

URI 싱크를 사용하는 경우 URI **싱크 → URI편집을 마우스 오른쪽 버튼으로 클릭하여 URI**를 수정할 수 있습니다.

API 서버 소스 삭제

1. **토폴로지** 보기로 이동합니다.
2. API 서버 소스를 마우스 오른쪽 버튼으로 클릭하고 **ApiServerSource 삭제**를 선택합니다.



2.3.2. Knative CLI를 사용하여 API 서버 소스 생성

kn source apiserver create 명령을 사용하여 **kn** CLI를 사용하여 API 서버 소스를 생성할 수 있습니다. **kn** CLI를 사용하여 API 서버 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

프로세스

기존 서비스 계정을 다시 사용하려면 새 리소스를 생성하는 대신 필요한 권한을 포함하도록 기존 **ServiceAccount** 리소스를 수정할 수 있습니다.

1. 이벤트 소스에 대한 서비스 계정, 역할, 역할 바인딩을 YAML 파일로 만듭니다.

```
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 이 네임스페이스를 이벤트 소스 설치를 위해 선택한 네임스페이스로 변경합니다.

2. YAML 파일을 적용합니다.

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

3. 이벤트 싱크가 있는 API 서버 소스를 생성합니다. 다음 예에서 싱크는 브로커입니다.

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

4. API 서버 소스가 올바르게 설정되었는지 확인하려면 수신되는 메시지를 로그로 덤프하는 Knative 서비스를 생성합니다.

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

5. 브로커를 이벤트 싱크로 사용한 경우 **default** 브로커의 이벤트를 서비스로 필터링하는 트리거를 생성합니다.

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

6. 기본 네임스페이스에서 Pod를 시작하여 이벤트를 생성합니다.

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

7. 생성된 출력을 다음 명령으로 검사하여 컨트롤러가 올바르게 매핑되는지 확인합니다.

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

출력 예

```
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
```

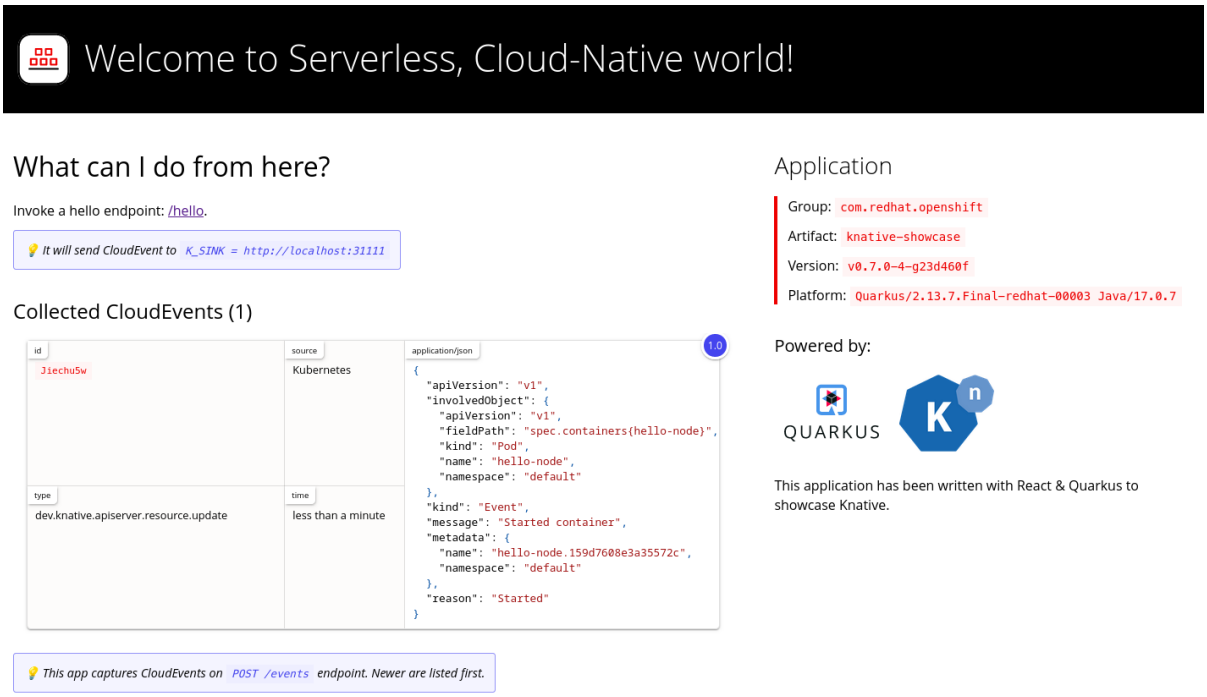
검증

Kubernetes 이벤트가 Knative로 전송되었는지 확인하려면 event-display 로그를 보거나 웹 브라우저를 사용하여 이벤트를 확인합니다.

- 웹 브라우저에서 이벤트를 보려면 다음 명령에서 반환된 링크를 엽니다.

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

그림 2.1. 브라우저 페이지 예



- 또는 터미널에서 로그를 보려면 다음 명령을 입력하여 Pod의 event-display 로그를 확인합니다.

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

출력 예

```
☁  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",
  ...
}
```

API 서버 소스 삭제

1. 트리거를 삭제합니다.

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

2. 이벤트 소스를 삭제합니다.

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

3. 서비스 계정, 클러스터 역할, 클러스터 바인딩을 삭제합니다.

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

2.3.2.1. Knative CLI sink 플래그

Knative(**kn**) CLI를 사용하여 이벤트 소스를 생성할 때 **--sink** 플래그를 사용하여 해당 리소스에서 이벤트가 전송되는 싱크를 지정할 수 있습니다. 싱크는 다른 리소스에서 들어오는 이벤트를 수신할 수 있는 주소 지정 가능 또는 호출 가능 리소스일 수 있습니다.

다음 예제에서는 싱크로 서비스 **http://event-display.svc.cluster.local** 를 사용하는 싱크 바인딩을 생성합니다.

sink 플래그를 사용하는 명령의 예

```
$ 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** 의 **svc** 는 싱크가 Knative 서비스인지 확인합니다. 기타 기본 싱크 접두사에는 **channel**, 및 **broker**가 포함됩니다.

2.3.3. YAML 파일을 사용하여 API 서버 소스 생성

YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 API를 사용하므로 선언적 및 재현 가능한 방식으로 이벤트 소스를 설명할 수 있습니다. YAML을 사용하여 API 서버 소스를 생성하려면 **ApiServerSource** 오브젝트를 정의하는 YAML 파일을 생성한 다음 **oc apply** 명령을 사용하여 적용해야 합니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- API 서버 소스 YAML 파일에 정의된 것과 동일한 네임스페이스에 **default** 브로커를 생성했습니다.
- OpenShift CLI(**oc**)를 설치합니다.

프로세스

기존 서비스 계정을 다시 사용하려면 새 리소스를 생성하는 대신 필요한 권한을 포함하도록 기존 **ServiceAccount** 리소스를 수정할 수 있습니다.

1. 이벤트 소스에 대한 서비스 계정, 역할, 역할 바인딩을 YAML 파일로 만듭니다.

```
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 이 네임스페이스를 이벤트 소스 설치를 위해 선택한 네임스페이스로 변경합니다.

2. YAML 파일을 적용합니다.

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

3. API 서버 소스를 YAML 파일로 만듭니다.

```
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. **ApiServerSource** YAML 파일을 적용합니다.

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

5. API 서버 소스가 올바르게 설정되었는지 확인하려면 수신되는 메시지를 로그로 덤프하는 Knative 서비스를 YAML 파일로 생성합니다.

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

6. **Service** YAML 파일을 적용합니다.

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

7. **default** 브로커에서 이전 단계에서 생성된 서비스로 이벤트를 필터링하는 YAML 파일로 **Trigger** 오브젝트를 생성합니다.

```
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. **Trigger** YAML 파일을 적용합니다.

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

9. 기본 네임스페이스에서 Pod를 시작하여 이벤트를 생성합니다.

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

10. 다음 명령을 입력하고 출력을 검사하여 컨트롤러가 올바르게 매핑되는지 확인합니다.

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

출력 예

```
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
```

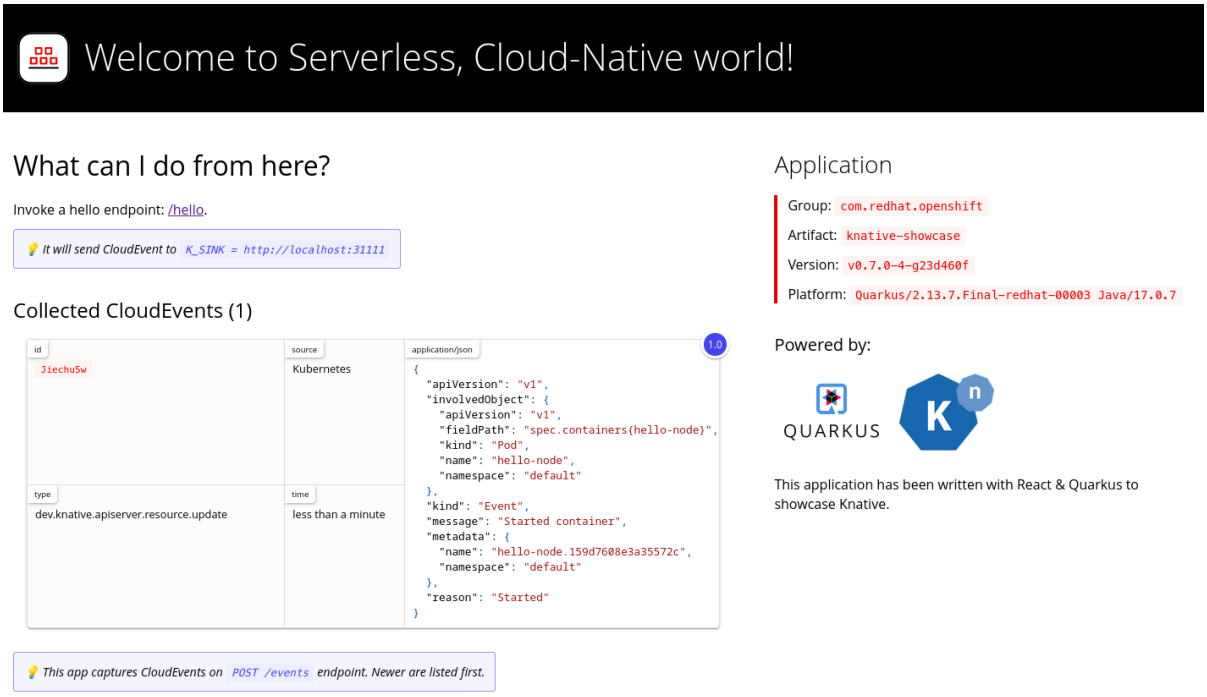
검증

Kubernetes 이벤트가 Knative로 전송되었는지 확인하려면 event-display 로그를 보거나 웹 브라우저를 사용하여 이벤트를 볼 수 있습니다.

- 웹 브라우저에서 이벤트를 보려면 다음 명령에서 반환된 링크를 엽니다.

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

그림 2.2. 브라우저 페이지 예



- 터미널에서 로그를 보려면 다음 명령을 입력하여 Pod의 event-display 로그를 확인합니다.

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

출력 예

```
☁  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",
  ...
}
```

API 서버 소스 삭제

1. 트리거를 삭제합니다.

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

2. 이벤트 소스를 삭제합니다.

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

3. 서비스 계정, 클러스터 역할, 클러스터 바인딩을 삭제합니다.

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

2.4. PING 소스 생성

ping 소스는 이벤트 소비자에게 일정한 페이로드를 사용하여 주기적으로 ping 이벤트를 보내는 데 사용할 수 있는 이벤트 소스입니다. ping 소스를 사용하면 타이머와 유사하게 전송 이벤트를 예약할 수 있습니다.

2.4.1. 웹 콘솔을 사용하여 ping 소스 생성

Knative Eventing이 클러스터에 설치되면 웹 콘솔을 사용하여 ping 소스를 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔을 사용하면 간소화되고 직관적인 사용자 인터페이스를 통해 이벤트 소스를 생성할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 웹 콘솔에 로그인했습니다.
- OpenShift Serverless Operator, Knative Serving 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

1. ping 소스가 작동하는지 확인하려면 수신 메시지를 서비스 로그에 덤프하는 간단한 Knative 서비스를 생성합니다.

   a. **개발자** 화면에서 **+추가 → YAML**로 이동합니다.

   b. 예제 YAML을 복사합니다.

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

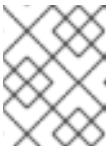
   c. **생성**을 클릭합니다.

2. 이전 단계에서 생성한 서비스와 동일한 네임스페이스 또는 이벤트를 보낼 다른 싱크에 ping 소스를 생성합니다.

   a. **개발자** 화면에서 **+추가 → 이벤트 소스**로 이동합니다. **이벤트 소스** 페이지가 표시됩니다.

   b. 선택 사항: 이벤트 소스에 대한 공급자가 여러 개인 경우 **공급자** 목록에서 필요한 공급자를 선택하여 해당 공급자의 사용 가능한 이벤트 소스를 필터링합니다.

   c. **Ping 소스**를 선택한 다음 **이벤트 소스 생성**을 클릭합니다. **이벤트 소스 생성** 페이지가 표시됩니다.

   참고

   양식 보기 또는 **YAML 보기**를 사용하여 **PingSource** 설정을 구성하고 보기 간에 전환할 수 있습니다. 다른 보기로 전환해도 데이터는 유지됩니다.

   d. **스케줄** 값을 입력합니다. 이 예에서 값은 ***/2 * * * ***이며, 2분마다 메시지를 전송하는 PingSource를 생성합니다.

   e. 선택 사항: 메시지 페이로드에 해당하는 **데이터** 값을 입력할 수 있습니다.

   f. **대상** 섹션에서 이벤트 싱크를 선택합니다. **리소스** 또는 **URI** 일 수 있습니다.

      i. 채널, 브로커 또는 서비스를 이벤트 소스의 이벤트 싱크로 사용하려면 **리소스** 를 선택합니다. 이 예에서는 이전 단계에서 생성한 **event-display** 서비스가 대상 리소스로 **사용됩니다**.

      ii. **URI** 를 선택하여 이벤트가 라우팅되는 URI(Uniform Resource Identifier)를 지정합니다.

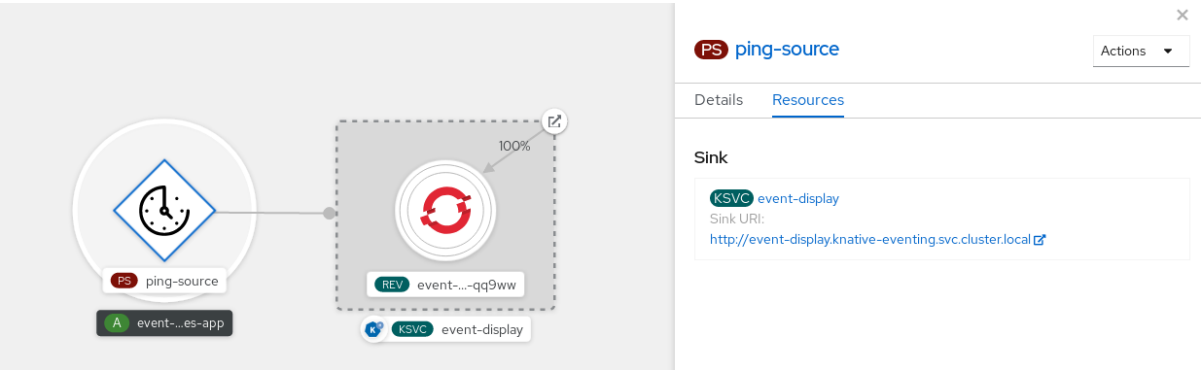
   g. **생성**을 클릭합니다.

검증

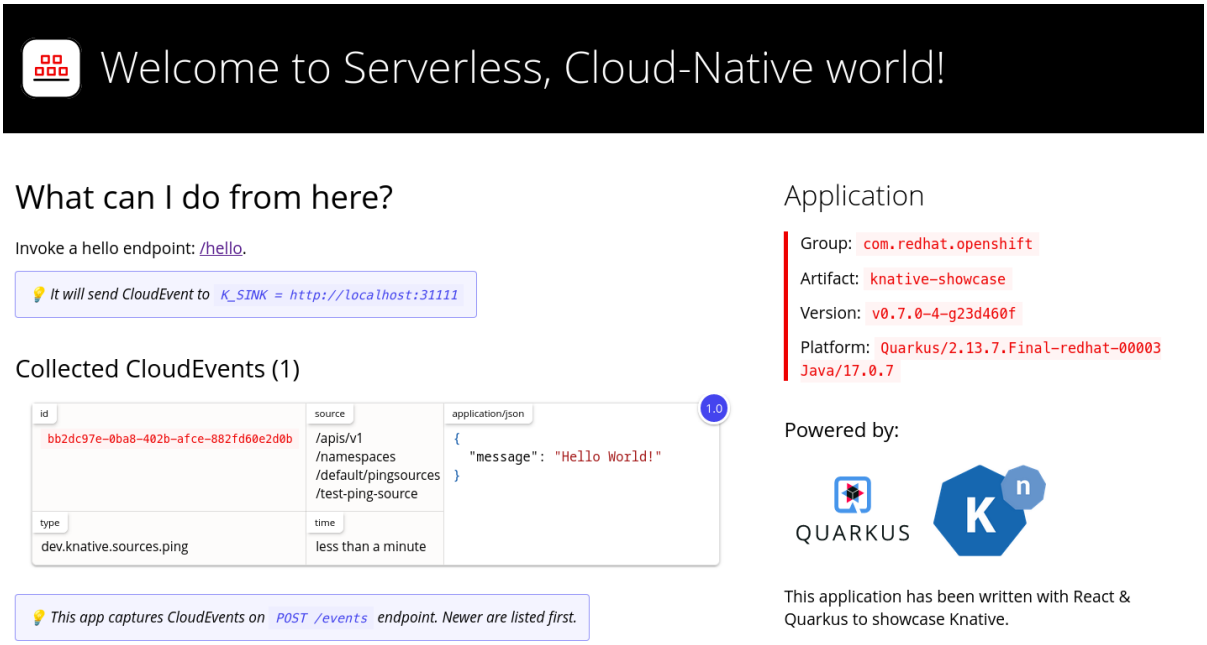
토폴로지 페이지를 확인하여 ping 소스가 생성되었고 싱크에 연결되어 있는지 확인할 수 있습니다.

1. **개발자** 화면에서 **토폴로지**로 이동합니다.

2. ping 소스 및 싱크를 확인합니다.



3. 웹 브라우저에서 event-display 서비스를 확인합니다. 웹 UI에서 ping 소스 이벤트가 표시되어야 합니다.



ping 소스 삭제

1. **토폴로지** 보기로 이동합니다.
2. API 서버 소스를 마우스 오른쪽 버튼으로 클릭하고 **Ping Source 삭제** 를 선택합니다.

2.4.2. Knative CLI를 사용하여 ping 소스 생성

kn source ping create 명령을 사용하여 Knative(**kn**) CLI를 사용하여 ping 소스를 생성할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Serving 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 선택 사항: 이 프로세스에 확인 단계를 사용하려면 OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. ping 소스가 작동하는지 확인하려면 수신 메시지를 서비스 로그에 덤프하는 간단한 Knative 서비스를 생성합니다.

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

2. 요청할 각 ping 이벤트 세트에 대해 이벤트 소비자와 동일한 네임스페이스에 ping 소스를 생성합니다.

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

3. 다음 명령을 입력하고 출력을 검사하여 컨트롤러가 올바르게 매핑되는지 확인합니다.

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

출력 예

```
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
```

검증

싱크 Pod의 로그를 보면 Kubernetes 이벤트가 Knative 이벤트 싱크로 전송되었는지 확인할 수 있습니다.

Knative 서비스는 기본적으로 60초 이내에 트래픽이 수신되지 않으면 Pod를 종료합니다. 이 가이드에 표시된 예제에서는 2분마다 메시지를 전송하는 ping 소스를 생성하므로 새로 생성된 Pod에서 각 메시지를 관찰해야 합니다.

1. 새 Pod가 생성되었는지 확인합니다.

```
$ watch oc get pods
```

2. Ctrl+C를 사용하여 Pod를 감시한 다음 생성한 Pod의 로그를 확인합니다.

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

출력 예

```
☁  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!"
  }
```

ping 소스 삭제

- ping 소스를 삭제합니다.

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

2.4.2.1. Knative CLI sink 플래그

Knative(**kn**) CLI를 사용하여 이벤트 소스를 생성할 때 **--sink** 플래그를 사용하여 해당 리소스에서 이벤트가 전송되는 싱크를 지정할 수 있습니다. 싱크는 다른 리소스에서 들어오는 이벤트를 수신할 수 있는 주소 지정 가능 또는 호출 가능 리소스일 수 있습니다.

다음 예제에서는 싱크로 서비스 **http://event-display.svc.cluster.local** 를 사용하는 싱크 바인딩을 생성합니다.

sink 플래그를 사용하는 명령의 예

```
$ 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** 의 **svc** 는 싱크가 Knative 서비스인지 확인합니다. 기타 기본 싱크 접두사에는 **channel**, 및 **broker**가 포함됩니다.

2.4.3. YAML을 사용하여 ping 소스 생성

YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 API를 사용하므로 선언적 및 재현 가능한 방식으로 이벤트 소스를 설명할 수 있습니다. YAML을 사용하여 서버리스 ping 소스를 생성하려면 **PingSource** 오브젝트를 정의하는 YAML 파일을 생성한 다음 **oc apply** 를 사용하여 적용해야 합니다.

PingSource 오브젝트의 예

```
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. CRON 표현식을 사용하여 지정한 이벤트 스케줄입니다.
2. JSON으로 인코딩된 데이터 문자열로 표시된 이벤트 메시지 본문입니다.
3. 이는 이벤트 소비자에 대한 세부 정보입니다. 이 예제에서는 **event-display**라는 Knative 서비스를 사용하고 있습니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Serving 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

1. ping 소스가 작동하는지 확인하려면 수신 메시지를 서비스 로그에 덤프하는 간단한 Knative 서비스를 생성합니다.
   a. 서비스 YAML 파일을 생성합니다.

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

b. 서비스를 생성합니다.

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

2. 요청할 각 ping 이벤트 세트에 대해 이벤트 소비자와 동일한 네임스페이스에 ping 소스를 생성합니다.

a. ping 소스에 대한 YAML 파일을 생성합니다.

```
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. ping 소스를 생성합니다.

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

3. 다음 명령을 입력하여 컨트롤러가 올바르게 매핑되는지 확인합니다.

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

출력 예

```
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
```

검증

싱크 Pod의 로그를 보면 Kubernetes 이벤트가 Knative 이벤트 싱크로 전송되었는지 확인할 수 있습니다.

Knative 서비스는 기본적으로 60초 이내에 트래픽이 수신되지 않으면 Pod를 종료합니다. 이 가이드에 표시된 예제에서는 2분마다 메시지를 전송하는 PingSource를 생성하므로 새로 생성된 Pod에서 각 메시지를 관찰해야 합니다.

1. 새 Pod가 생성되었는지 확인합니다.

```
$ watch oc get pods
```

2. Ctrl+C를 사용하여 Pod를 감시한 다음 생성한 Pod의 로그를 확인합니다.

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

출력 예

```
☁  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!"
  }
```

ping 소스 삭제

- ping 소스를 삭제합니다.

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

명령 예

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

2.5. APACHE KAFKA의 소스

Apache Kafka 클러스터에서 이벤트를 읽고 이러한 이벤트를 싱크에 전달하는 Apache Kafka 소스를 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔, Knative(**kn**) CLI를 사용하거나 **KafkaSource** 개체를 YAML 파일로 직접 생성하고 OpenShift CLI(**oc**)를 사용하여 이를 적용하여 Kafka 소스를 생성할 수 있습니다.

참고

Apache Kafka용 Knative 브로커 설치 설명서를 참조하십시오.

2.5.1. 웹 콘솔을 사용하여 Apache Kafka 이벤트 소스 생성

Apache Kafka에 대한 Knative 브로커 구현을 클러스터에 설치한 후 웹 콘솔을 사용하여 Apache Kafka 소스를 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔을 사용하면 Kafka 소스를 생성할 수 있는 간소화되고 직관적인 사용자 인터페이스가 제공됩니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing, **KnativeKafka** 사용자 정의 리소스가 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인했습니다.
- 가져오려는 Kafka 메시지를 생성하는 Red Hat AMQ Streams(Kafka) 클러스터에 액세스할 수 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

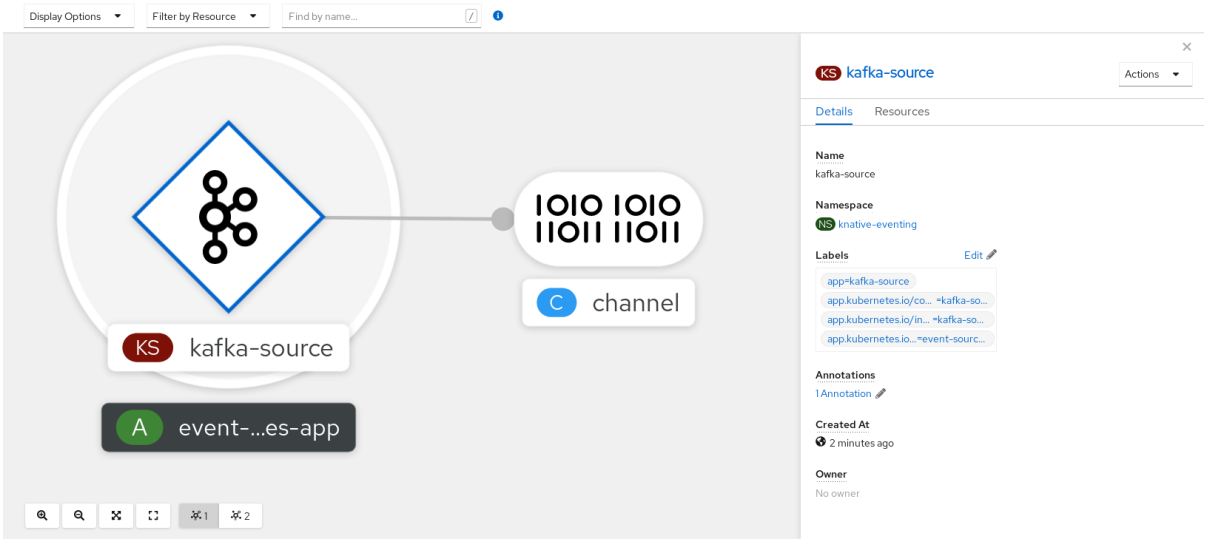
프로세스

1. **개발자** 화면에서 **+추가** 페이지로 이동하여 **이벤트 소스** 를 선택합니다.
2. **이벤트 소스** 페이지의 **유형** 섹션에서 **Kafka 소스**를 선택합니다.
3. **Kafka 소스** 설정을 구성합니다.
   a. 쉼표로 구분된 **부트스트랩 서버** 목록을 추가합니다.
   b. 쉼표로 구분된 **주제** 목록을 추가합니다.
   c. **소비자 그룹**을 추가합니다.
   d. 생성한 서비스 계정의 **서비스 계정 이름**을 선택합니다.
   e. **대상** 섹션에서 이벤트 싱크를 선택합니다. **리소스** 또는 **URI** 일 수 있습니다.
      i. 채널, 브로커 또는 서비스를 이벤트 소스의 이벤트 싱크로 사용하려면 **리소스** 를 선택합니다.
      ii. **URI** 를 선택하여 이벤트가 라우팅되는 URI(Uniform Resource Identifier)를 지정합니다.
   f. Kafka 이벤트 소스로 **이름**을 입력합니다.
4. **생성**을 클릭합니다.

검증

토폴로지 페이지를 확인하여 Kafka 이벤트 소스가 생성되었고 싱크에 연결되어 있는지 확인할 수 있습니다.

1. **개발자** 화면에서 **토폴로지**로 이동합니다.
2. Kafka 이벤트 소스 및 싱크를 확인합니다.



2.5.2. Knative CLI를 사용하여 Apache Kafka 이벤트 소스 생성

kn source kafka create 명령을 사용하여 Knative(**kn**) CLI를 사용하여 Kafka 소스를 생성할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing, Knative Serving, **KnativeKafka** 사용자 정의 리소스(CR)가 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 가져오려는 Kafka 메시지를 생성하는 Red Hat AMQ Streams(Kafka) 클러스터에 액세스할 수 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.
- 선택 사항: 이 절차의 확인 단계를 사용하려는 경우 OpenShift CLI(**oc**)를 설치했습니다.

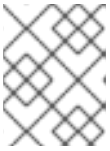
프로세스

1. Kafka 이벤트 소스가 작동하는지 확인하려면 수신 이벤트를 서비스 로그에 덤프하는 Knative 서비스를 생성합니다.

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

2. **KafkaSource** CR을 생성합니다.

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



참고

이 명령의 자리 표시자 값을 소스 이름, 부트스트랩 서버 및 주제의 값으로 바꿉니다.

--servers, **--topics**, **--consumergroup** 옵션은 Kafka 클러스터에 대한 연결 매개 변수를 지정합니다. **--consumergroup** 옵션은 선택 사항입니다.

3. 선택 사항: 생성한 **KafkaSource** CR에 대한 세부 정보를 확인합니다.

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

출력 예

```
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
```

검증 단계

1. Kafka 인스턴스를 트리거하여 메시지를 항목에 보냅니다.

```
$ 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
```

프롬프트에 메시지를 입력합니다. 이 명령은 다음을 가정합니다.

- Kafka 클러스터는 **kafka** 네임스페이스에 설치됩니다.
- **my-topic** 주제를 사용하도록 **KafkaSource** 오브젝트가 구성되어 있습니다.

2. 로그를 보고 메시지가 도착했는지 확인합니다.

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

출력 예

```
☁  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 플래그

Knative(**kn**) CLI를 사용하여 이벤트 소스를 생성할 때 **--sink** 플래그를 사용하여 해당 리소스에서 이벤트가 전송되는 싱크를 지정할 수 있습니다. 싱크는 다른 리소스에서 들어오는 이벤트를 수신할 수 있는 주소 지정 가능 또는 호출 가능 리소스일 수 있습니다.

다음 예제에서는 싱크로 서비스 **http://event-display.svc.cluster.local** 를 사용하는 싱크 바인딩을 생성합니다.

sink 플래그를 사용하는 명령의 예

```
$ 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** 의 **svc** 는 싱크가 Knative 서비스인지 확인합니다. 기타 기본 싱크 접두사에는 **channel**, 및 **broker**가 포함됩니다.

2.5.3. YAML을 사용하여 Apache Kafka 이벤트 소스 생성

YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 API를 사용하므로 선언적 및 재현 가능한 방식으로 애플리케이션을 설명할 수 있습니다. YAML을 사용하여 Kafka 소스를 생성하려면 **KafkaSource** 오브젝트를 정의하는 YAML 파일을 생성한 다음 **oc apply** 명령을 사용하여 적용해야 합니다.

사전 요구 사항

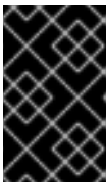
- OpenShift Serverless Operator, Knative Eventing, **KnativeKafka** 사용자 정의 리소스가 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 가져오려는 Kafka 메시지를 생성하는 Red Hat AMQ Streams(Kafka) 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. **KafkaSource** 오브젝트를 YAML 파일로 생성합니다.

```
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. 소비자 그룹은 동일한 그룹 ID를 사용하고 한 주제의 데이터를 사용하는 소비자 그룹입니다.
2. 주제에서는 데이터 스토리지의 대상을 제공합니다. 각 주제는 하나 이상의 파티션으로 나뉩니다.
3. 싱크는 소스에서 이벤트를 보내는 위치를 지정합니다.



중요

OpenShift Serverless의 **KafkaSource** 개체에 대한 API의 **v1beta1** 버전만 지원됩니다. 이 버전은 더 이상 사용되지 않으므로 이 API의 **v1alpha1** 버전을 사용하지 마십시오.

KafkaSource 오브젝트의 예

```
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. **KafkaSource** YAML 파일을 적용합니다.

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

검증

- 다음 명령을 입력하여 Kafka 이벤트 소스가 생성되었는지 확인합니다.

```
$ oc get pods
```

출력 예

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

2.5.4. Apache Kafka 소스에 대한 SASL 인증 구성

SASL(*Simple Authentication and Security Layer*)은 Apache Kafka에서 인증에 사용됩니다. 클러스터에서 SASL 인증을 사용하는 경우 Kafka 클러스터와 통신하기 위해 Knative에 인증 정보를 제공해야 합니다. 그렇지 않으면 이벤트를 생성하거나 사용할 수 없습니다.

사전 요구 사항

- OpenShift Container Platform에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- OpenShift Serverless Operator, Knative Eventing, **KnativeKafka** CR이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Kafka 클러스터의 사용자 이름과 암호가 있습니다.
- **PLAIN**,**SCRAM-SHA-256** 또는 **SCRAM-SHA-512** 와 같이 사용할 SASL 메커니즘을 선택했습니다.
- TLS가 활성화된 경우 Kafka 클러스터의 **ca.crt** 인증서 파일도 필요합니다.
- OpenShift(**oc**) CLI가 설치되어 있습니다.

프로세스

1. 선택한 네임스페이스에서 인증서 파일을 시크릿으로 생성합니다.

```
$ 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 유형은 **PLAIN**,**SCRAM-SHA-256** 또는 **SCRAM-SHA-512** 일 수 있습니다.

2. 다음 **사양** 구성이 포함되도록 Kafka 소스를 생성하거나 수정합니다.

```
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 퍼블릭 클라우드 Kafka 서비스를 사용하는 경우 **caCert** 사양이 필요하지 않습니다.

2.5.5. KafkaSource에 대한 KEDA 자동 스케일링 구성

Kubernetes Event Driven Autoscaler(KEDA)를 기반으로 하는 Custom Metrics Autoscaler Operator를 사용하여 Apache Kafka(KafkaSource)의 Knative Eventing 소스를 자동 스케일링하도록 구성할 수 있습니다.

중요

KafkaSource에 대한 KEDA 자동 스케일링을 구성하는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing, **KnativeKafka** 사용자 정의 리소스가 클러스터에 설치되어 있습니다.

프로세스

1. **KnativeKafka** 사용자 정의 리소스에서 KEDA 스케일링을 활성화합니다.

   YAML의 예

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  name: knative-kafka
  namespace: knative-eventing
spec:
  config:
    kafka-features:
      controller-autoscaler-keda: enabled
```

2. **KnativeKafka** YAML 파일을 적용합니다.

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

2.6. 사용자 정의 이벤트 소스

Knative에 포함되지 않은 이벤트 생산자 또는 **CloudEvent** 형식이 아닌 이벤트를 내보내는 생산자에서 이벤트를 수신해야 하는 경우 사용자 정의 이벤트 소스를 생성하여 이 작업을 수행할 수 있습니다. 다음 방법 중 하나를 사용하여 사용자 지정 이벤트 소스를 생성할 수 있습니다.

- 싱크 바인딩을 생성하여 **PodSpecable** 오브젝트를 이벤트 소스로 사용합니다.
- 컨테이너 소스를 생성하여 컨테이너 소스로 컨테이너를 사용합니다.

2.6.1. 싱크 바인딩

SinkBinding 오브젝트는 전달 주소에서 이벤트 프로덕션의 분리를 지원합니다. 싱크 바인딩은 *이벤트 생산자를 이벤트* 소비자 또는 *싱크*에 연결하는 데 사용됩니다. 이벤트 생산자는 **PodSpec** 템플릿을 포함하고 이벤트를 생성하는 Kubernetes 리소스입니다. 싱크는 이벤트를 수신할 수 있는 주소 지정 가능한 Kubernetes 오브젝트입니다.

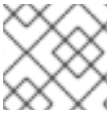
SinkBinding 오브젝트는 싱크의 **PodTemplateSpec**에 환경 변수를 삽입합니다. 즉, 애플리케이션 코드가 이벤트 대상을 찾기 위해 Kubernetes API와 직접 상호 작용할 필요가 없습니다. 이러한 환경 변수는 다음과 같습니다.

K_SINK

확인된 싱크의 URL입니다.

K_CE_OVERRIDES

아웃바운드 이벤트에 대한 재정의를 지정하는 JSON 오브젝트입니다.



참고

SinkBinding 오브젝트는 현재 서비스에 대한 사용자 정의 버전 이름을 지원하지 않습니다.

2.6.1.1. YAML을 사용하여 싱크 바인딩 생성

YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 API를 사용하므로 선언적 및 재현 가능한 방식으로 이벤트 소스를 설명할 수 있습니다. YAML을 사용하여 싱크 바인딩을 생성하려면 **SinkBinding** 오브젝트를 정의하는 YAML 파일을 생성한 다음 **oc apply** 명령을 사용하여 적용해야 합니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Serving 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

1. 싱크 바인딩이 올바르게 설정되었는지 확인하려면 Knative 이벤트 표시 서비스를 생성하여 수신되는 메시지를 로그로 덤프합니다.

   a. 서비스 YAML 파일을 생성합니다.

   서비스 YAML 파일의 예

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

   b. 서비스를 생성합니다.

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

2. 이벤트를 서비스로 보내는 싱크 바인딩 인스턴스를 생성합니다.

   a. 싱크 바인딩 YAML 파일을 생성합니다.

   서비스 YAML 파일의 예

   ```
   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 이 예제에서는 **app: Heartbeat-cron** 라벨이 있는 모든 작업이 이벤트 싱크에 바인딩됩니다.

b. 싱크 바인딩을 생성합니다.

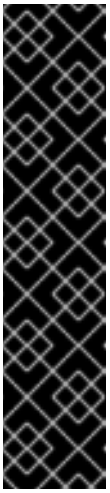
```
$ oc apply -f <filename>
```

3. **CronJob** 오브젝트를 생성합니다.

a. cron 작업 YAML 파일을 생성합니다.

cron 작업 YAML 파일의 예

```
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
```



중요

싱크 바인딩을 사용하려면 Knative 리소스에 **bindings.knative.dev/include=true** 라벨을 수동으로 추가해야 합니다.

예를 들어 이 라벨을 **CronJob** 리소스에 추가하려면 **Job** 리소스 YAML 정의에 다음 행을 추가합니다.

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

b. cron 작업을 생성합니다.

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

4. 다음 명령을 입력하고 출력을 검사하여 컨트롤러가 올바르게 매핑되는지 확인합니다.

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

출력 예

```
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
```

검증

메시지 덤퍼 기능 로그를 보면 Kubernetes 이벤트가 Knative 이벤트 싱크로 전송되었는지 확인할 수 있습니다.

1. 명령을 입력합니다.

```
$ oc get pods
```

2. 명령을 입력합니다.

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

출력 예

```
☁  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. Knative CLI를 사용하여 싱크 바인딩 생성

kn source binding create 명령을 사용하여 Knative(**kn**) CLI를 사용하여 싱크 바인딩을 생성할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Serving 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Knative(**kn**) CLI를 설치합니다.
- OpenShift CLI(**oc**)를 설치합니다.

참고

다음 절차에 따라 YAML 파일을 생성해야 합니다.

예제에 사용된 YAML 파일의 이름을 변경하는 경우 해당 CLI 명령도 업데이트해야 합니다.

프로세스

1. 싱크 바인딩이 올바르게 설정되었는지 확인하려면 Knative 이벤트 표시 서비스를 생성하여 수신되는 메시지를 로그로 덤프합니다.

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

2. 이벤트를 서비스로 보내는 싱크 바인딩 인스턴스를 생성합니다.

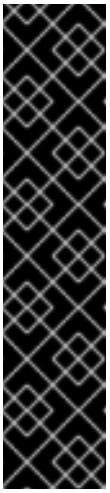
```
$ kn source binding create bind-heartbeat --subject Job:batch/v1:app=heartbeat-cron --sink
ksvc:event-display
```

3. **CronJob** 오브젝트를 생성합니다.

   a. cron 작업 YAML 파일을 생성합니다.

   cron 작업 YAML 파일의 예

```
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
```



중요

싱크 바인딩을 사용하려면 Knative CR에 **bindings.knative.dev/include=true** 라벨을 수동으로 추가해야 합니다.

예를 들어 이 라벨을 **CronJob** CR에 추가하려면 **Job** CR YAML 정의에 다음 행을 추가합니다.

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

   b. cron 작업을 생성합니다.

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

4. 다음 명령을 입력하고 출력을 검사하여 컨트롤러가 올바르게 매핑되는지 확인합니다.

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

출력 예

```
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
```

검증

메시지 덤퍼 기능 로그를 보면 Kubernetes 이벤트가 Knative 이벤트 싱크로 전송되었는지 확인할 수 있습니다.

- 다음 명령을 입력하여 메시지 덤퍼 기능 로그를 확인합니다.

```
$ oc get pods
```

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

출력 예

```
☁  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 플래그

Knative(**kn**) CLI를 사용하여 이벤트 소스를 생성할 때 **--sink** 플래그를 사용하여 해당 리소스에서 이벤트가 전송되는 싱크를 지정할 수 있습니다. 싱크는 다른 리소스에서 들어오는 이벤트를 수신할 수 있는 주소 지정 가능 또는 호출 가능 리소스일 수 있습니다.

다음 예제에서는 싱크로 서비스 **http://event-display.svc.cluster.local** 를 사용하는 싱크 바인딩을 생성합니다.

sink 플래그를 사용하는 명령의 예

```
$ 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** 의 **svc** 는 싱크가 Knative 서비스인지 확인합니다. 기타 기본 싱크 접두사에는 **channel**, 및 **broker**가 포함됩니다.

2.6.1.3. 웹 콘솔을 사용하여 싱크 바인딩 생성

Knative Eventing이 클러스터에 설치되면 웹 콘솔을 사용하여 싱크 바인딩을 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔을 사용하면 간소화되고 직관적인 사용자 인터페이스를 통해 이벤트 소스를 생성할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 웹 콘솔에 로그인했습니다.
- OpenShift Serverless Operator, Knative Serving, Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

1. 싱크로 사용할 Knative 서비스를 생성합니다.
   a. **개발자** 화면에서 **+추가 → YAML**로 이동합니다.
   b. 예제 YAML을 복사합니다.

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
spec:
  template:
```

```
    spec:
      containers:
        - image: quay.io/openshift-knative/showcase
```

c. **생성**을 클릭합니다.

2. 이벤트 소스로 사용되고 1분마다 이벤트를 전송하는 **CronJob** 리소스를 생성합니다.

a. **개발자** 화면에서 **+추가 → YAML**로 이동합니다.

b. 예제 YAML을 복사합니다.

```
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) **bindings.knative.dev/include: true** 라벨을 포함해야 합니다. OpenShift Serverless의 기본 네임스페이스 선택 동작은 포함 모드를 사용합니다.

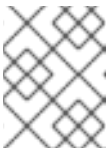
c. **생성**을 클릭합니다.

3. 이전 단계에서 생성한 서비스와 동일한 네임스페이스 또는 이벤트를 보낼 다른 싱크 바인딩을 생성합니다.

a. **개발자** 화면에서 **+추가 → 이벤트 소스**로 이동합니다. **이벤트 소스** 페이지가 표시됩니다.

b. 선택 사항: 이벤트 소스에 대한 공급자가 여러 개인 경우 **공급자** 목록에서 필요한 공급자를 선택하여 해당 공급자의 사용 가능한 이벤트 소스를 필터링합니다.

c. **싱크 바인딩**을 선택한 다음 **이벤트 소스 생성**을 클릭합니다. **이벤트 소스 생성** 페이지가 표시됩니다.



참고

양식 보기 또는 **YAML 보기**를 사용하여 **Sink 바인딩** 설정을 구성하고 보기 간에 전환할 수 있습니다. 다른 보기로 전환해도 데이터는 유지됩니다.

d. **apiVersion** 필드에 **batch/v1**을 입력합니다.

e. **유형** 필드에 **Job**을 입력합니다.

참고

CronJob 종류는 OpenShift Serverless 싱크 바인딩에서 직접 지원하지 않으므로 **kind** 필드에서 cron 작업 오브젝트 자체 대신 cron 작업 오브젝트에서 생성한 **Job** 오브젝트를 대상으로 해야 합니다.

f. **대상** 섹션에서 이벤트 싱크를 선택합니다. **리소스** 또는 **URI** 일 수 있습니다.

i. 채널, 브로커 또는 서비스를 이벤트 소스의 이벤트 싱크로 사용하려면 **리소스** 를 선택합니다. 이 예에서는 이전 단계에서 생성한 **event-display** 서비스가 대상 리소스로 **사용됩니다**.

ii. **URI** 를 선택하여 이벤트가 라우팅되는 URI(Uniform Resource Identifier)를 지정합니다.

g. **레이블 일치** 섹션에서 다음을 수행합니다.

i. **이름** 필드에 **app**을 입력합니다.

ii. **값** 필드에 **heartbeat-cron**을 입력합니다.

참고

레이블 선택기는 리소스 이름이 아니라 싱크 바인딩과 함께 cron 작업을 사용할 때 필요합니다. cron 작업에서 생성한 작업에 예측 가능한 이름이 없고 이름에 무작위로 생성된 문자열이 있기 때문입니다. 예: **hearthbeat-cron-1cc23f**

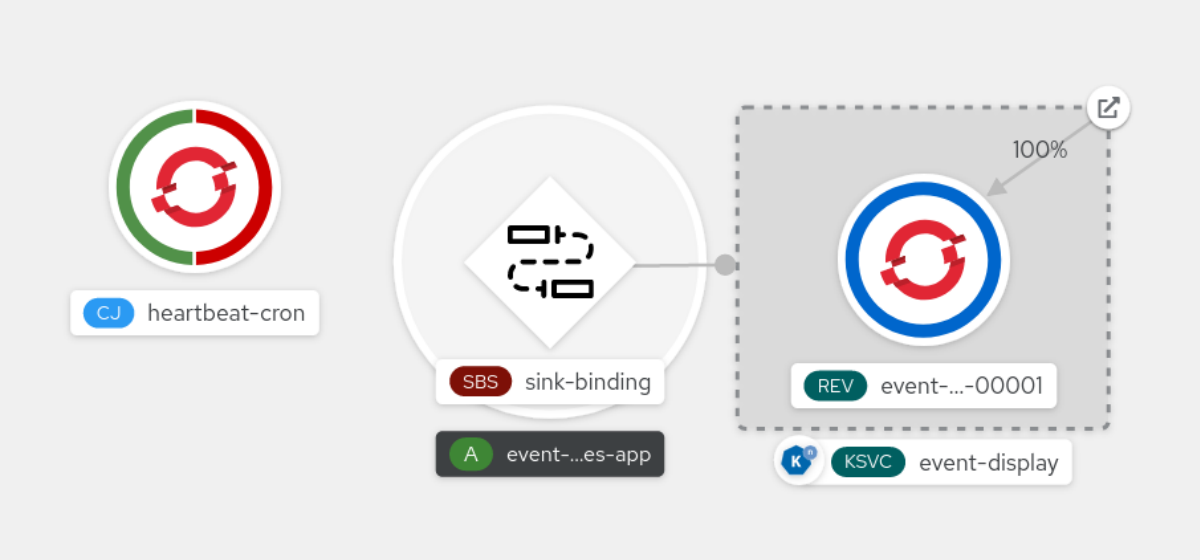
h. **생성**을 클릭합니다.

검증

토폴로지 페이지 및 Pod 로그를 확인하여 싱크 바인딩, 싱크 및 cron 작업이 생성되었으며 올바르게 작동하는지 확인할 수 있습니다.

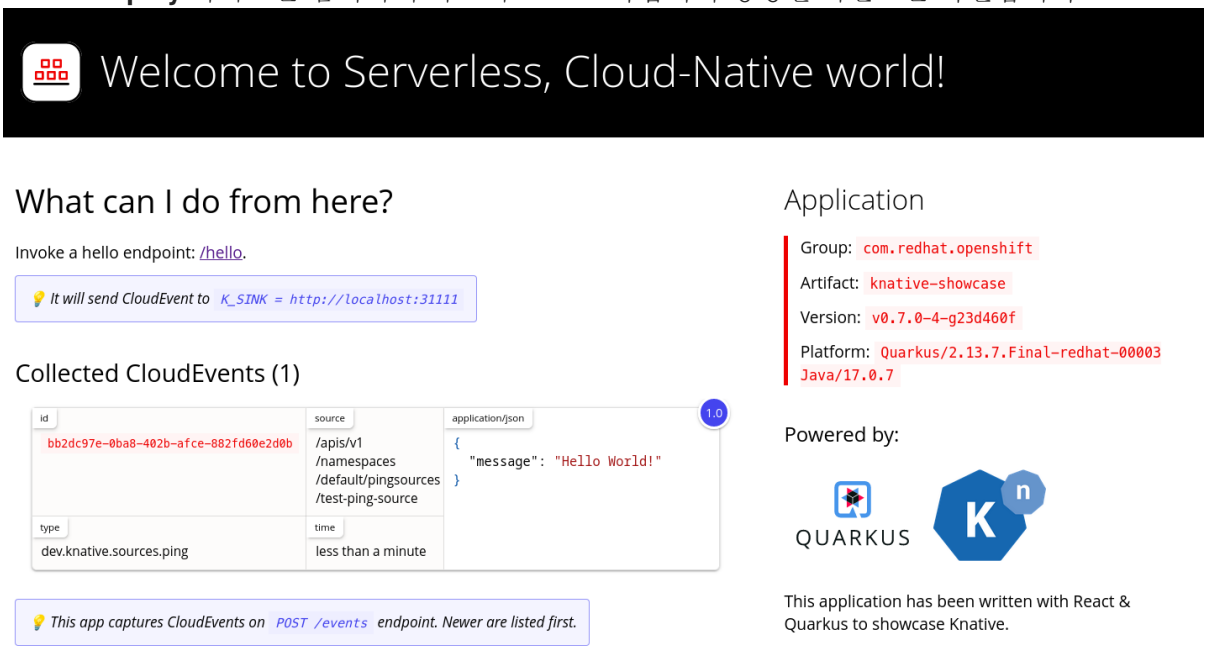
1. **개발자** 화면에서 **토폴로지**로 이동합니다.

2. 싱크 바인딩, 싱크 및 하트비트 cron 작업을 확인합니다.



3. 싱크 바인딩이 추가되면 cron 작업에 의해 성공한 작업이 등록되고 있는지 확인합니다. 즉, 싱크 바인딩이 cron 작업에서 생성한 작업을 성공적으로 재구성합니다.

4. **event-display** 서비스를 검색하여 하트비트 cron 작업에서 생성한 이벤트를 확인합니다.



2.6.1.4. 싱크 바인딩 참조

싱크 바인딩을 생성하여 **PodSpecable** 오브젝트를 이벤트 소스로 사용할 수 있습니다. **SinkBinding** 오브젝트를 생성할 때 여러 매개변수를 구성할 수 있습니다.

SinkBinding 오브젝트는 다음 매개변수를 지원합니다.

필드	설명	필수 또는 선택 사항
apiVersion	API 버전을 지정합니다(예: **sources.knative.dev/v1**).	필수 항목

필드	설명	필수 또는 선택 사항
kind	이 리소스 오브젝트를 **SinkBinding** 오브젝트로 식별합니다.	필수 항목
메타데이터	**SinkBinding** 오브젝트를 고유하게 식별하는 메타데이터를 지정합니다. 예를 들면 다음과 같습니다.	필수 항목
spec	이 **SinkBinding** 오브젝트에 대한 구성 정보를 지정합니다.	필수 항목
spec.sink	싱크로 사용할 URI로 확인되는 오브젝트에 대한 참조입니다.	필수 항목
spec.subject	런타임 계약이 바인딩 구현에 의해 보강되는 리소스를 참조합니다.References the resources for which the runtime contract is augmented by binding implementations.	필수 항목
spec.ceOverrides	출력 형식을 제어하고 싱크로 전송된 이벤트에 대한 수정을 제어하기 위한 덮어쓰기를 정의합니다.	선택 사항

2.6.1.4.1. subject 매개변수

Subject 매개변수는 바인딩 구현에 의해 런타임 계약이 보강되는 리소스를 참조합니다. **주체** 정의에 대해 여러 필드를 구성할 수 있습니다.

주체 정의는 다음 필드를 지원합니다.

필드	설명	필수 또는 선택 사항
apiVersion	참조의 API 버전입니다.	필수 항목
kind	일종의 추천입니다.	필수 항목
네임스페이스	참조의 네임스페이스입니다. 생략하면 기본값은 오브젝트의 네임스페이스입니다.	선택 사항

필드	설명	필수 또는 선택 사항
name	추천자의 이름입니다.	**선택기** 를 구성하는 경우 를 사용하지 마십시오.
선택기	추천자의 선택기입니다.	**이름을** 구성하는 경우 를 사용하지 마십시오.
selector.matchExpressions	라벨 선택기 요구 사항 목록입니다.	**matchExpressions** 또는 **matchLabels** 중 하나만 사용하십시오.
selector.matchExpressions.key	선택기가 적용되는 라벨 키입니다.	**matchExpressions** 를 사용하는 경우 필수 항목입니다.
selector.matchExpressions.operator	값 집합에 대한 키의 관계를 나타냅니다. 유효한 연산자는 **In**,**NotIn**,**Exists** 및 **DoesNotExist** 입니다.	**matchExpressions** 를 사용하는 경우 필수 항목입니다.
selector.matchExpressions.values	문자열 값의 배열입니다. **operator** 매개변수 값이 **In** 또는 **NotIn** 인 경우 값 배열은 비어 있지 않아야 합니다. **operator** 매개변수 값이 **Exists** 또는 **DoesNotExist** 이면 값 배열이 비어 있어야 합니다. 이 배열은 전략적 병합 패치 중에 교체됩니다.	**matchExpressions** 를 사용하는 경우 필수 항목입니다.
selector.matchLabels	키-값 쌍의 맵입니다. **matchLabels** 맵의 각 키-값 쌍은 **matchExpressions** 의 요소와 동일합니다. 여기서 key 필드는 **matchLabels.** <key> , **operator** 는 **In** 이며, **값** 배열에는 **matchLabels.<value** > 만 포함됩니다.	**matchExpressions** 또는 **matchLabels** 중 하나만 사용하십시오.

subject 매개변수 예

다음 YAML에 따라 **기본** 네임스페이스에서 **mysubject** 라는 **Deployment** 오브젝트가 선택됩니다.

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

```
    namespace: default
    name: mysubject
  ...
```

다음 YAML에 따라 **기본** 네임스페이스에 있는 **working=example** 레이블이 있는 모든 **Job** 오브젝트가 선택됩니다.

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

다음 YAML에서 **기본** 네임스페이스의 **working=example** 또는 **working=sample** 라벨이 있는 모든 **Pod** 오브젝트가 선택됩니다.

```
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
  ...
```

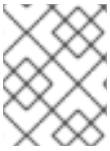
2.6.1.4.2. CloudEvent 덮어쓰기

ceOverrides 정의에서는 CloudEvent의 출력 형식과 싱크로 전송된 수정 사항을 제어하는 덮어쓰기를 제공합니다. **ceOverrides** 정의에 대해 여러 필드를 구성할 수 있습니다.

ceOverrides 정의에서는 다음 필드를 지원합니다.

필드	설명	필수 또는 선택 사항

필드	설명	필수 또는 선택 사항
확장	아웃바운드 이벤트에서 추가되거나 재정의되는 특성을 지정합니다. 각 **확장** 키-값 쌍은 이벤트에서 속성 확장으로 독립적으로 설정됩니다.	선택 사항



참고

유효한 **CloudEvent** 속성 이름만 확장으로 허용됩니다. 확장 덮어쓰기 구성에서 spec 정의 속성을 설정할 수 없습니다. 예를 들어 **type** 속성을 수정할 수 없습니다.

CloudEvent 덮어쓰기 예

```
apiVersion: sources.knative.dev/v1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  ...
  ceOverrides:
    extensions:
      extra: this is an extra attribute
      additional: 42
```

이렇게 하면 제목의 **K_CE_OVERRIDES** 환경 변수가 **설정됩니다**.

출력 예

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

2.6.1.4.3. include 레이블

싱크 바인딩을 사용하려면 **바인딩.knative.dev/include: "true"** 라벨을 리소스가 포함된 리소스 또는 네임스페이스에 할당해야 합니다. 리소스 정의에 레이블이 포함되지 않은 경우 다음을 실행하여 클러스터 관리자가 네임스페이스에 연결할 수 있습니다.

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

2.6.1.5. Service Mesh와 싱크 바인딩 통합

사전 요구 사항

- OpenShift Serverless와 통합 Service Mesh가 있습니다.

프로세스

1. **Service MeshMemberRoll** 의 멤버인 네임스페이스에 서비스를 생성합니다.
   -

```
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/showcase
```

(1) **ServiceMeshMemberRoll** 의 멤버인 네임스페이스입니다.

(2) Service Mesh 사이드카를 Knative 서비스 Pod에 삽입합니다.

2. **Service** 리소스를 적용합니다.

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

3. **SinkBinding** 리소스를 생성합니다.

```
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) **ServiceMeshMemberRoll** 의 멤버인 네임스페이스입니다.

(2) 이 예에서는 **app: heartbeat-cron** 레이블이 있는 모든 작업이 이벤트 싱크에 바인딩됩니다.

4. **SinkBinding** 리소스를 적용합니다.

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

5. **CronJob** 생성:

```
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. **ServiceMeshMemberRoll** 의 멤버인 네임스페이스입니다.
2. 서비스 메시 사이드카를 **CronJob** Pod에 삽입합니다.

6. **CronJob** 리소스를 적용합니다.

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

검증

이벤트가 Knative 이벤트 싱크로 전송되었는지 확인하려면 메시지 덤퍼 기능 로그를 확인합니다.

1. 다음 명령을 실행합니다.

```
$ oc get pods
```

2. 다음 명령을 실행합니다.

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

출력 예

```
☁  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": ""
  }
```

추가 리소스

- OpenShift Serverless와 Service Mesh 통합

2.6.2. 컨테이너 소스

컨테이너 소스는 이벤트를 생성하고 이벤트를 싱크로 보내는 컨테이너 이미지를 생성합니다. 컨테이너 소스를 사용하여 컨테이너 이미지 및 이미지 URI를 사용하는 **ContainerSource** 오브젝트를 생성하여 사용자 정의 이벤트 소스를 생성할 수 있습니다.

2.6.2.1. 컨테이너 이미지 생성을 위한 지침

컨테이너 소스 컨트롤러에서 두 개의 환경 변수인 **K_SINK** 및 **K_CE_OVERRIDES**를 삽입합니다. 이러한 변수는 **sink** 및 **ceOverrides** 사양에서 확인됩니다. 이벤트는 **K_SINK** 환경 변수에 지정된 싱크 URI로 전송됩니다. 메시지는 **CloudEvent** HTTP 형식을 사용하여 **POST** 로 보내야 합니다.

컨테이너 이미지의 예

다음은 하트비트 컨테이너 이미지의 예입니다.

```
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
 }
}
```

다음은 이전 하트비트 컨테이너 이미지를 참조하는 컨테이너 소스의 예입니다.

```
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. Knative CLI를 사용하여 컨테이너 소스 생성 및 관리

kn source 컨테이너 명령을 사용하여 Knative(kn) CLI를 사용하여 컨테이너 소스를 생성하고 관리할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다.

컨테이너 소스 생성

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

컨테이너 소스 삭제

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

컨테이너 소스 설명

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

기존 컨테이너 소스 나열

```
$ kn source container list
```

YAML 형식으로 기존 컨테이너 소스 나열

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

컨테이너 소스 업데이트

이 명령은 기존 컨테이너 소스의 이미지 URI를 업데이트합니다.

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

2.6.2.3. 웹 콘솔을 사용하여 컨테이너 소스 생성

Knative Eventing이 클러스터에 설치되면 웹 콘솔을 사용하여 컨테이너 소스를 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔을 사용하면 간소화되고 직관적인 사용자 인터페이스를 통해 이벤트 소스를 생성할 수 있습니다.

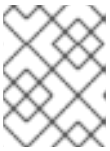
사전 요구 사항

- OpenShift Container Platform 웹 콘솔에 로그인했습니다.
- OpenShift Serverless Operator, Knative Serving, Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

1. **개발자** 화면에서 **+추가 → 이벤트 소스**로 이동합니다. **이벤트 소스** 페이지가 표시됩니다.
2. **컨테이너 소스**를 선택한 다음 **이벤트 소스 생성**을 클릭합니다. **이벤트 소스 생성** 페이지가 표시됩니다.

3. **양식 보기** 또는 **YAML 보기**를 사용하여 **컨테이너 소스** 설정을 구성합니다.



참고

양식 보기와 **YAML 보기**를 전환할 수 있습니다. 다른 보기로 전환해도 데이터는 유지됩니다.

   a. **이미지** 필드에 컨테이너 소스에서 생성한 컨테이너에서 실행할 이미지의 URI를 입력합니다.

   b. **이름** 필드에 이미지 이름을 입력합니다.

   c. 선택 사항: **인수** 필드에 컨테이너에 전달할 인수를 입력합니다.

   d. 선택 사항: **환경 변수** 필드에서 컨테이너에 설정할 환경 변수를 추가합니다.

   e. **대상** 섹션에서 이벤트 싱크를 선택합니다. **리소스** 또는 **URI** 일 수 있습니다.

      i. 채널, 브로커 또는 서비스를 이벤트 소스의 이벤트 싱크로 사용하려면 **리소스** 를 선택합니다.

      ii. **URI** 를 선택하여 이벤트가 라우팅되는 URI(Uniform Resource Identifier)를 지정합니다.

4. 컨테이너 소스 구성을 완료한 후 **생성**을 클릭합니다.

2.6.2.4. 컨테이너 소스 참조

ContainerSource 오브젝트를 생성하여 컨테이너를 이벤트 소스로 사용할 수 있습니다. **ContainerSource** 오브젝트를 생성할 때 여러 매개변수를 구성할 수 있습니다.

ContainerSource 개체는 다음 필드를 지원합니다.

필드	설명	필수 또는 선택 사항
apiVersion	API 버전을 지정합니다(예: **sources.knative.dev/v1**).	필수 항목
kind	이 리소스 오브젝트를 **ContainerSource** 오브젝트로 식별합니다.	필수 항목
메타데이터	**ContainerSource** 개체를 고유하게 식별하는 메타데이터를 지정합니다. 예를 들면 다음과 같습니다.	필수 항목
spec	이 **ContainerSource** 개체의 구성 정보를 지정합니다.	필수 항목

필드	설명	필수 또는 선택 사항
spec.sink	싱크로 사용할 URI로 확인되는 오브젝트에 대한 참조입니다.	필수 항목
spec.template	**ContainerSource** 오브젝트의 **템플릿** 사양입니다.	필수 항목
spec.ceOverrides	출력 형식을 제어하고 싱크로 전송된 이벤트에 대한 수정을 제어하기 위한 덮어쓰기를 정의합니다.	선택 사항

템플릿 매개변수 예

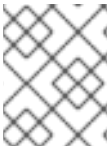
```
apiVersion: sources.knative.dev/v1
kind: ContainerSource
metadata:
  name: test-heartbeats
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/heartbeats:latest
          name: heartbeats
          args:
            - --period=1
          env:
            - name: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

2.6.2.4.1. CloudEvent 덮어쓰기

ceOverrides 정의에서는 CloudEvent의 출력 형식과 싱크로 전송된 수정 사항을 제어하는 덮어쓰기를 제공합니다. **ceOverrides** 정의에 대해 여러 필드를 구성할 수 있습니다.

ceOverrides 정의에서는 다음 필드를 지원합니다.

필드	설명	필수 또는 선택 사항
확장	아웃바운드 이벤트에서 추가되거나 재정의되는 특성을 지정합니다. 각 **확장** 키-값 쌍은 이벤트에서 속성 확장으로 독립적으로 설정됩니다.	선택 사항



참고

유효한 **CloudEvent** 속성 이름만 확장으로 허용됩니다. 확장 덮어쓰기 구성에서 spec 정의 속성을 설정할 수 없습니다. 예를 들어 **type** 속성을 수정할 수 없습니다.

CloudEvent 덮어쓰기 예

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

이렇게 하면 제목의 **K_CE_OVERRIDES** 환경 변수가 **설정됩니다**.

출력 예

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

2.6.2.5. ContainerSource와 서비스 메시 통합

사전 요구 사항

- OpenShift Serverless와 통합 Service Mesh가 있습니다.

프로세스

1. **Service MeshMemberRoll** 의 멤버인 네임스페이스에 서비스를 생성합니다.

```
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/showcase
```

1. **ServiceMeshMemberRoll** 의 멤버인 네임스페이스입니다.
2. Service Mesh 사이드카를 Knative 서비스 Pod에 삽입합니다.

2. **Service** 리소스를 적용합니다.

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

3. **ServiceMeshMemberRoll** 및 sink의 멤버인 네임스페이스에 **event-display** 로 설정된 **ContainerSource** 오브젝트를 생성합니다.

```
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. 네임스페이스는 **ServiceMeshMemberRoll** 의 일부입니다.
2. **ContainerSource** 오브젝트와 Service Mesh 통합을 활성화합니다.

4. **ContainerSource** 리소스를 적용합니다.

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

검증

이벤트가 Knative 이벤트 싱크로 전송되었는지 확인하려면 메시지 덤퍼 기능 로그를 확인합니다.

1. 다음 명령을 실행합니다.

```
$ oc get pods
```

2. 다음 명령을 실행합니다.

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

-

출력 예

```
☁  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": ""
  }
```

추가 리소스

- OpenShift Serverless와 Service Mesh 통합

2.7. 개발자 화면을 사용하여 이벤트 싱크에 이벤트 소스 연결

OpenShift Container Platform 웹 콘솔을 사용하여 이벤트 소스를 생성할 때 해당 소스에서 이벤트가 전송되는 대상 이벤트 싱크를 지정할 수 있습니다. 이벤트 싱크는 다른 리소스에서 들어오는 이벤트를 수신할 수 있는 주소 지정 가능 또는 호출 가능 리소스일 수 있습니다.

2.7.1. 개발자 화면을 사용하여 이벤트 싱크에 이벤트 소스 연결

사전 요구 사항

- OpenShift Serverless Operator, Knative Serving, Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 **개발자** 화면으로 갑니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Knative 서비스, 채널 또는 브로커와 같은 이벤트 싱크를 생성했습니다.

프로세스

1. **+추가** → 이벤트 소스로 이동하고 생성할 **이벤트** 소스 유형을 선택하여 모든 유형의 이벤트 소스를 생성합니다.
2. 이벤트 **소스 생성 양식 보기의 대상 섹션에서 이벤트** 싱크를 선택합니다. **리소스** 또는 **URI** 일 수 있습니다.

a. 채널, 브로커 또는 서비스를 이벤트 소스의 이벤트 싱크로 사용하려면 **리소스** 를 선택합니다.

b. **URI** 를 선택하여 이벤트가 라우팅되는 URI(Uniform Resource Identifier)를 지정합니다.

3. **생성**을 클릭합니다.

검증

토폴로지 페이지를 확인하여 이벤트 소스가 생성되었고 싱크에 연결되어 있는지 확인할 수 있습니다.

1. **개발자** 화면에서 **토폴로지**로 이동합니다.
2. 이벤트 소스를 보고 연결된 이벤트 싱크를 클릭하여 오른쪽 패널에서 싱크 세부 정보를 확인합니다.

3장. 이벤트 싱크

3.1. 이벤트 싱크

이벤트 소스를 생성할 때 소스에서 이벤트가 전송되는 이벤트 싱크를 지정할 수 있습니다. 이벤트 싱크는 다른 리소스에서 들어오는 이벤트를 수신할 수 있는 주소 지정 가능 리소스 또는 호출 가능 리소스입니다. Knative 서비스, 채널 및 브로커는 모두 이벤트 싱크의 예입니다. 특정 Apache Kafka 싱크 유형도 사용할 수 있습니다.

주소 지정 가능 오브젝트는 HTTP를 통해 전달되는 이벤트를 **status.address.url** 필드에 정의된 주소로 수신하고 승인합니다. 특수한 경우 코어 Kubernetes **서비스** 오브젝트도 주소 지정 가능한 인터페이스를 수행합니다.

호출 가능 오브젝트는 HTTP를 통해 전달되는 이벤트를 수신하고 이벤트를 변환하여 HTTP 응답에서 **0** 또는 **1** 개의 새 이벤트를 반환할 수 있습니다. 반환된 이벤트는 외부 이벤트 소스의 이벤트를 처리하는 것과 동일한 방식으로 추가로 처리할 수 있습니다.

3.1.1. Knative CLI sink 플래그

Knative(**kn**) CLI를 사용하여 이벤트 소스를 생성할 때 **--sink** 플래그를 사용하여 해당 리소스에서 이벤트가 전송되는 싱크를 지정할 수 있습니다. 싱크는 다른 리소스에서 들어오는 이벤트를 수신할 수 있는 주소 지정 가능 또는 호출 가능 리소스일 수 있습니다.

다음 예제에서는 싱크로 서비스 **http://event-display.svc.cluster.local** 를 사용하는 싱크 바인딩을 생성합니다.

sink 플래그를 사용하는 명령의 예

```
$ 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** 의 **svc** 는 싱크가 Knative 서비스인지 확인합니다. 기타 기본 싱크 접두사에는 **channel**, 및 **broker**가 포함됩니다.

작은 정보

kn을 사용자 지정하여 Knative(**kn**) CLI 명령에 **--sink** 플래그와 함께 사용할 수 있는 CR을 구성할 수 있습니다.

3.2. 이벤트 싱크 생성

이벤트 소스를 생성할 때 소스에서 이벤트가 전송되는 이벤트 싱크를 지정할 수 있습니다. 이벤트 싱크는 다른 리소스에서 들어오는 이벤트를 수신할 수 있는 주소 지정 가능 리소스 또는 호출 가능 리소스입니다. Knative 서비스, 채널 및 브로커는 모두 이벤트 싱크의 예입니다. 특정 Apache Kafka 싱크 유형도 사용할 수 있습니다.

이벤트 싱크로 사용할 수 있는 리소스 생성에 대한 자세한 내용은 다음 설명서를 참조하십시오.

- 서버리스 애플리케이션

- 브로커 생성
- 채널 생성
- Kafka 싱크

3.3. APACHE KAFKA의 싱크

Apache Kafka 싱크는 클러스터 관리자가 클러스터에서 Apache Kafka를 활성화한 경우 사용할 수 있는 이벤트 싱크 유형입니다. Kafka 싱크를 사용하여 이벤트 소스에서 Kafka 주제로 직접 이벤트를 보낼 수 있습니다.

3.3.1. YAML을 사용하여 Apache Kafka 싱크 생성

Kafka 주제로 이벤트를 전송하는 Kafka 싱크를 생성할 수 있습니다. 기본적으로 Kafka 싱크는 구조화된 모드보다 더 효율적인 바이너리 콘텐츠 모드를 사용합니다. YAML을 사용하여 Kafka 싱크를 생성하려면 **KafkaSink** 오브젝트를 정의하는 YAML 파일을 생성한 다음 **oc apply** 명령을 사용하여 적용해야 합니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing 및 **KnativeKafka** CR(사용자 정의 리소스)이 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 가져오려는 Kafka 메시지를 생성하는 Red Hat AMQ Streams(Kafka) 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. **KafkaSink** 오브젝트 정의를 YAML 파일로 생성합니다.

   Kafka 싱크 YAML

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

2. Kafka 싱크를 생성하려면 **KafkaSink** YAML 파일을 적용합니다.

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

3. 싱크가 사양에 지정되도록 이벤트 소스를 구성합니다.

   API 서버 소스에 연결된 Kafka 싱크의 예

```
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. 이벤트 소스의 이름입니다.
2. 이벤트 소스의 네임스페이스입니다.
3. 이벤트 소스의 서비스 계정입니다.
4. Kafka 싱크 이름입니다.

3.3.2. OpenShift Container Platform 웹 콘솔을 사용하여 Apache Kafka의 이벤트 싱크 생성

OpenShift Container Platform 웹 콘솔의 **개발자** 화면을 사용하여 Kafka 주제로 이벤트를 전송하는 Kafka 싱크를 생성할 수 있습니다. 기본적으로 Kafka 싱크는 구조화된 모드보다 더 효율적인 바이너리 콘텐츠 모드를 사용합니다.

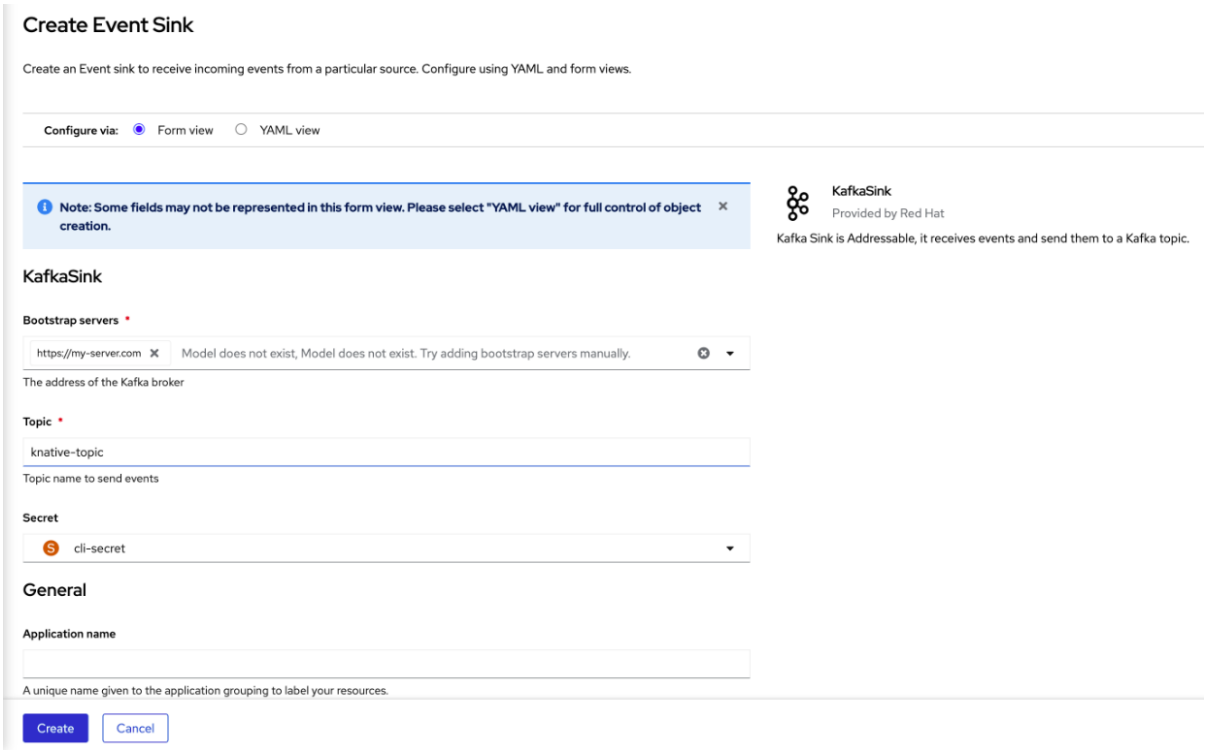
개발자는 이벤트 싱크를 생성하여 특정 소스에서 이벤트를 수신하고 Kafka 주제로 보낼 수 있습니다.

사전 요구 사항

- OperatorHub에서 Knative Serving, Knative Eventing 및 Knative 브로커 for Apache Kafka API를 사용하여 OpenShift Serverless Operator를 설치했습니다.
- Kafka 환경에서 Kafka 주제를 생성했습니다.

프로세스

1. **개발자** 화면에서 **+추가** 보기로 이동합니다.
2. **Eventing 카탈로그** 에서 **이벤트 싱크** 를 클릭합니다.
3. 카탈로그 항목에서 **KafkaSink** 를 검색하고 클릭합니다.
4. **이벤트 싱크 생성을** 클릭합니다.
5. 양식 뷰에서 호스트 이름과 포트의 조합인 부트스트랩 서버의 URL을 입력합니다.



6. 이벤트 데이터를 보낼 주제의 이름을 입력합니다.
7. 이벤트 싱크의 이름을 입력합니다.
8. **생성**을 클릭합니다.

검증

1. **개발자** 화면에서 **토폴로지** 보기로 이동합니다.
2. 생성된 이벤트 싱크를 클릭하여 오른쪽 패널에서 세부 정보를 확인합니다.

3.3.3. Apache Kafka 싱크에 대한 보안 구성

TLS(*Transport Layer Security*)는 Apache Kafka 클라이언트와 서버에서 Knative와 Kafka 간의 트래픽을 암호화하고 인증을 위해 사용됩니다. TLS는 Apache Kafka에 대한 Knative 브로커 구현에 지원되는 유일한 트래픽 암호화 방법입니다.

SASL(*Simple Authentication and Security Layer*)은 Apache Kafka에서 인증에 사용됩니다. 클러스터에서 SASL 인증을 사용하는 경우 Kafka 클러스터와 통신하기 위해 Knative에 인증 정보를 제공해야 합니다. 그렇지 않으면 이벤트를 생성하거나 사용할 수 없습니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing 및 **KnativeKafka** CR(사용자 정의 리소스)이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Kafka 싱크는 **KnativeKafka** CR에서 활성화됩니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Kafka 클러스터 CA 인증서가 **.pem** 파일로 저장되어 있습니다.

- Kafka 클러스터 클라이언트 인증서와 **.pem** 파일로 저장된 키가 있습니다.
- OpenShift(**oc**) CLI가 설치되어 있습니다.
- **PLAIN**,**SCRAM-SHA-256** 또는 **SCRAM-SHA-512** 와 같이 사용할 SASL 메커니즘을 선택했습니다.

프로세스

1. **KafkaSink** 오브젝트와 동일한 네임스페이스에서 인증서 파일을 시크릿으로 생성합니다.

중요

인증서와 키는 PEM 형식이어야 합니다.

- 암호화 없이 SASL을 사용한 인증의 경우:

```
$ 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>
```

- TLS를 사용한 SASL 및 암호화를 사용한 인증의 경우:

```
$ 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 퍼블릭 클라우드 관리 Kafka 서비스를 사용하는 경우 시스템의 루트 CA 세트를 사용하도록 **ca.crt** 를 생략할 수 있습니다.

- TLS를 사용한 인증 및 암호화의 경우:

```
$ 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 퍼블릭 클라우드 관리 Kafka 서비스를 사용하는 경우 시스템의 루트 CA 세트를 사용하도록 **ca.crt** 를 생략할 수 있습니다.

2. **KafkaSink** 오브젝트를 생성하거나 수정하고 **auth** 사양에서 보안에 대한 참조를 추가합니다.

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

```
    namespace: <namespace>
spec:
...
    auth:
      secret:
        ref:
          name: <secret_name>
...
```

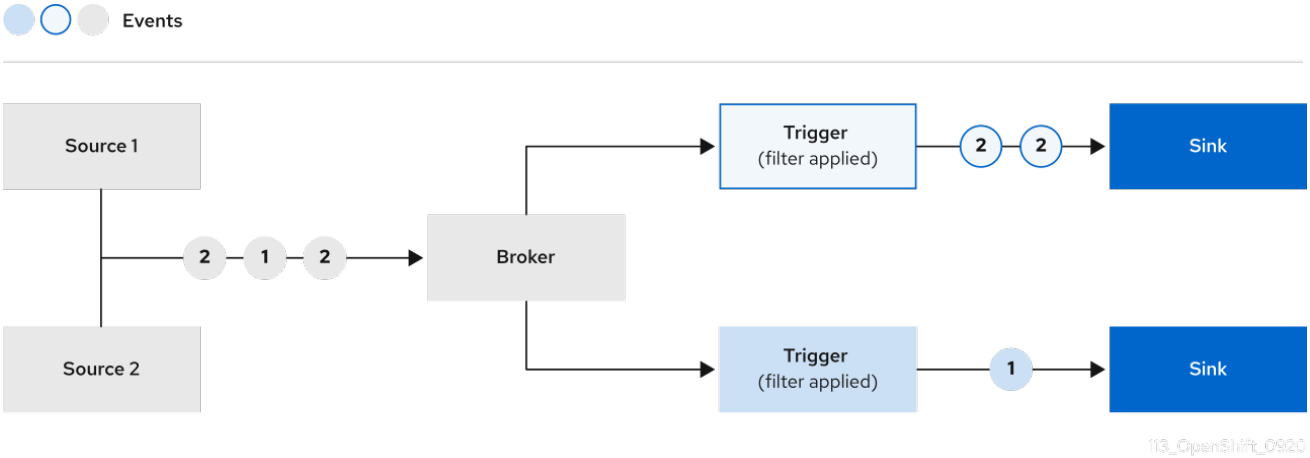
3. **KafkaSink** 오브젝트를 적용합니다.

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

4장. 브로커

4.1. 브로커

브로커는 트리거와 함께 이벤트 소스에서 이벤트 싱크로 이벤트를 전달할 수 있습니다. 이벤트는 이벤트 소스에서 HTTP **POST** 요청으로 브로커로 전송됩니다. 이벤트가 브로커에 진입하면 트리거를 사용하여 CloudEvent 속성으로 필터링하고 이벤트 싱크에 HTTP **POST** 요청으로 보낼 수 있습니다.



4.2. 브로커 유형

클러스터 관리자는 클러스터의 기본 브로커 구현을 설정할 수 있습니다. 브로커를 생성할 때 **Broker** 오브젝트에 설정된 구성을 제공하지 않는 한 기본 브로커 구현이 사용됩니다.

4.2.1. 개발 목적을 위한 기본 브로커 구현

Knative는 기본 채널 기반 브로커 구현을 제공합니다. 이 채널 기반 브로커는 개발 및 테스트 목적으로 사용할 수 있지만 프로덕션 환경에 적합한 이벤트 전달 보장은 제공하지 않습니다. 기본 브로커는 기본적으로 **InMemoryChannel** 채널 구현에서 지원합니다.

Apache Kafka를 사용하여 네트워크 홉을 줄이려면 Apache Kafka에 Knative 브로커 구현을 사용합니다. **KafkaChannel** 채널 구현에서 지원하도록 채널 기반 브로커를 구성하지 마십시오.

4.2.2. Apache Kafka에 대한 프로덕션 지원 Knative 브로커 구현

프로덕션 지원 Knative Eventing 배포의 경우 Red Hat은 Apache Kafka에 Knative 브로커 구현을 사용하는 것이 좋습니다. 브로커는 Knative 브로커의 Apache Kafka 기본 구현으로, CloudEvents를 Kafka 인스턴스로 직접 보냅니다.

Knative 브로커에는 이벤트 저장 및 라우팅을 위한 Kafka와 기본 통합이 있습니다. 이를 통해 다른 브로커 유형보다 브로커 및 트리거 모델에 대한 Kafka와 보다 효과적으로 통합할 수 있으며 네트워크 홉이 줄어듭니다. Knative 브로커 구현의 기타 이점은 다음과 같습니다.

- at-least-once 전달 보장
- CloudEvents 파티션 확장에 따라 이벤트를 정렬된 제공
- 컨트롤 플레인 고가용성
- 수평으로 확장 가능한 데이터 플레인

Apache Kafka의 Knative 브로커 구현은 바이너리 콘텐츠 모드를 사용하여 들어오는 CloudEvents를 Kafka 레코드로 저장합니다. 즉, 모든 CloudEvent 속성 및 확장이 Kafka 레코드의 헤더로 매핑되지만 CloudEvent의 **data** 사양은 Kafka 레코드의 값에 해당합니다.

4.3. 브로커 생성

Knative는 기본 채널 기반 브로커 구현을 제공합니다. 이 채널 기반 브로커는 개발 및 테스트 목적으로 사용할 수 있지만 프로덕션 환경에 적합한 이벤트 전달 보장은 제공하지 않습니다.

클러스터 관리자가 Apache Kafka를 기본 브로커 유형으로 사용하도록 OpenShift Serverless 배포를 구성한 경우 기본 설정을 사용하여 브로커를 생성하면 Apache Kafka용 Knative 브로커가 생성됩니다.

OpenShift Serverless 배포가 Apache Kafka에 Knative 브로커를 기본 브로커 유형으로 사용하도록 구성되지 않은 경우 다음 절차의 기본 설정을 사용할 때 채널 기반 브로커가 생성됩니다.

4.3.1. Knative CLI를 사용하여 브로커 생성

브로커는 트리거와 함께 이벤트 소스에서 이벤트 싱크로 이벤트를 전달할 수 있습니다. Knative(**kn**) CLI를 사용하여 브로커를 생성하면 YAML 파일을 직접 수정하는 것보다 간소화되고 직관적인 사용자 인터페이스가 제공됩니다. **kn broker create** 명령을 사용하여 브로커를 생성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

- 브로커를 생성합니다.

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

검증

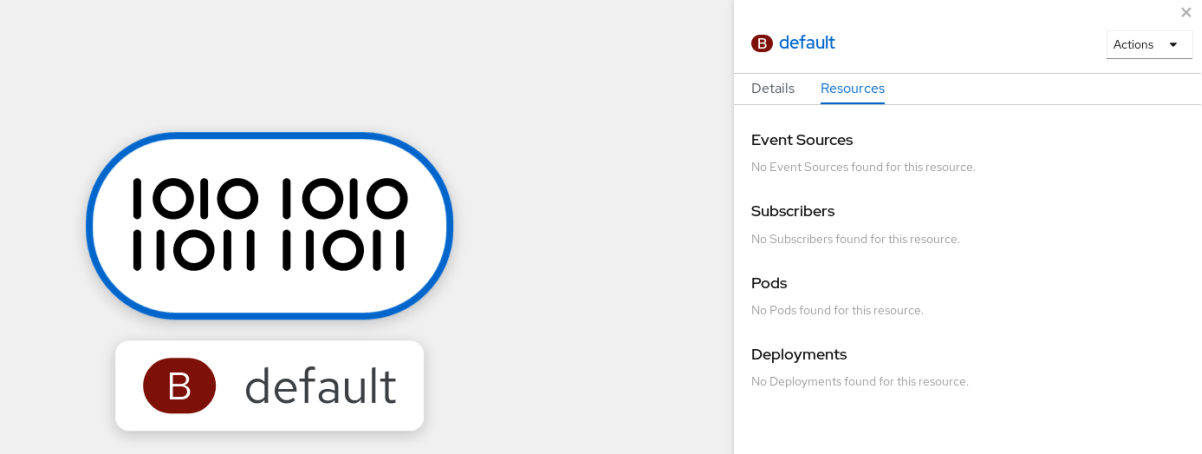
1. **kn** 명령을 사용하여 기존 브로커를 모두 나열합니다.

```
$ kn broker list
```

출력 예

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

2. 선택 사항: OpenShift Container Platform 웹 콘솔을 사용하는 경우 **개발자** 화면에서 **토폴로지** 보기로 이동하여 브로커가 존재하는지 관찰할 수 있습니다.



4.3.2. 트리거에 주석을 달아 브로커 생성

브로커는 트리거와 함께 이벤트 소스에서 이벤트 싱크로 이벤트를 전달할 수 있습니다. **Trigger** 오브젝트에 **eventing.knative.dev/injection: enabled** 주석을 추가하여 브로커를 생성할 수 있습니다.

중요

eventing.knative.dev/injection: enabled 주석을 사용하여 브로커를 생성하는 경우 클러스터 관리자 권한이 없어 이 브로커를 삭제할 수 없습니다. 클러스터 관리자가 이 주석을 먼저 제거하기 전에 브로커를 삭제하면 삭제 후 브로커가 다시 생성됩니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

1. **eventing.knative.dev/injection: enabled** 주석이 있는 **Trigger** 오브젝트를 YAML 파일로 생성합니다.

```
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 트리거에서 이벤트를 전송할 대상 이벤트 싱크 또는 *구독자*에 대한 세부 정보를 지정합니다.

2. **Trigger** YAML 파일을 적용합니다.

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

검증

oc CLI를 사용하거나 웹 콘솔의 **토폴로지** 보기에서 브로커가 생성되었는지 확인할 수 있습니다.

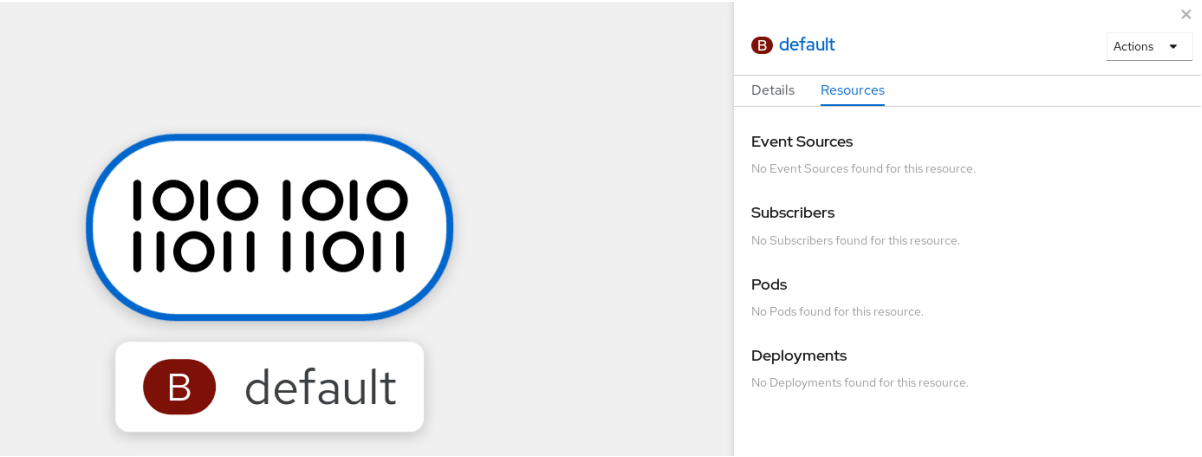
1. **oc** 명령을 사용하여 브로커를 가져옵니다.

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

출력 예

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

2. 선택 사항: OpenShift Container Platform 웹 콘솔을 사용하는 경우 **개발자** 화면에서 **토폴로지** 보기로 이동하여 브로커가 존재하는지 관찰할 수 있습니다.



4.3.3. 네임스페이스에 라벨을 지정하여 브로커 생성

브로커는 트리거와 함께 이벤트 소스에서 이벤트 싱크로 이벤트를 전달할 수 있습니다. 소유하고 있거나 쓰기 권한이 있는 네임스페이스에 라벨을 지정하여 **default** 브로커를 자동으로 생성할 수 있습니다.

참고

이 방법을 사용하여 생성한 브로커는 라벨을 제거하면 제거되지 않습니다. 수동으로 삭제해야 합니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

- AWS 또는 OpenShift Dedicated에서 Red Hat OpenShift Service를 사용하는 경우 클러스터 또는 전용 관리자 권한이 있습니다.

프로세스

- **eventing.knative.dev/injection=enabled**를 사용하여 네임스페이스에 라벨을 지정합니다.

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

검증

oc CLI를 사용하거나 웹 콘솔의 **토폴로지** 보기에서 브로커가 생성되었는지 확인할 수 있습니다.

1. **oc** 명령을 사용하여 브로커를 가져옵니다.

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

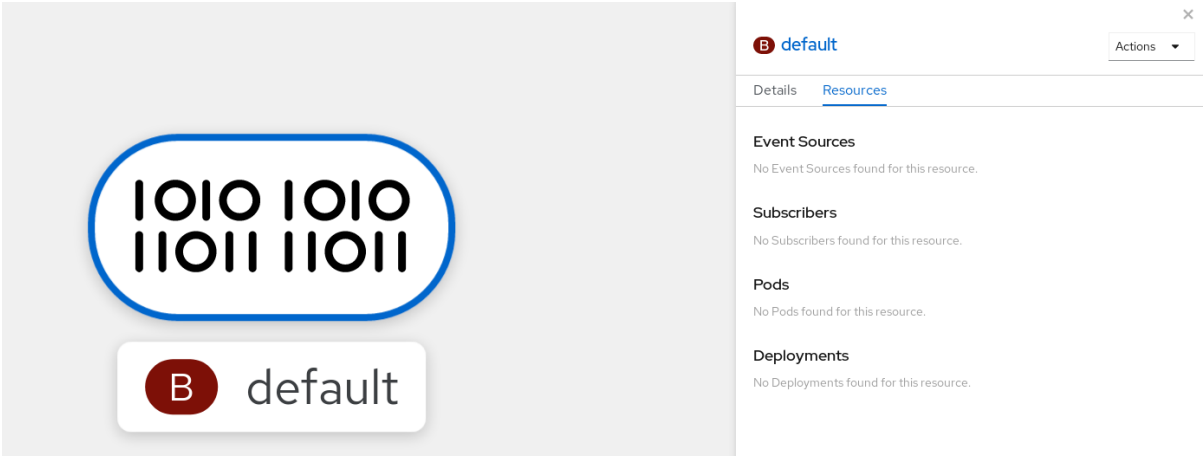
명령 예

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

출력 예

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

2. 선택 사항: OpenShift Container Platform 웹 콘솔을 사용하는 경우 **개발자** 화면에서 **토폴로지** 보기로 이동하여 브로커가 존재하는지 관찰할 수 있습니다.



4.3.4. 삽입을 통해 생성된 브로커 삭제

삽입을 통해 브로커를 생성하고 나중에 삭제하려는 경우 수동으로 삭제해야 합니다. 레이블 또는 주석을 제거하면 네임스페이스 레이블 또는 트리거 주석을 사용하여 생성한 브로커는 영구적으로 삭제되지 않습니다.

사전 요구 사항

- OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. 네임스페이스에서 **eventing.knative.dev/injection=enabled** 라벨을 제거합니다.

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

주석을 제거하면 Knative에서 브로커를 삭제한 후 다시 생성하지 않습니다.

2. 선택한 네임스페이스에서 브로커를 삭제합니다.

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

검증

- **oc** 명령을 사용하여 브로커를 가져옵니다.

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

명령 예

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

출력 예

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

4.3.5. 웹 콘솔을 사용하여 브로커 생성

Knative Eventing이 클러스터에 설치되면 웹 콘솔을 사용하여 브로커를 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔을 사용하면 간소화되고 직관적인 사용자 인터페이스를 제공하여 브로커를 생성할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 웹 콘솔에 로그인했습니다.
- OpenShift Serverless Operator, Knative Serving 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

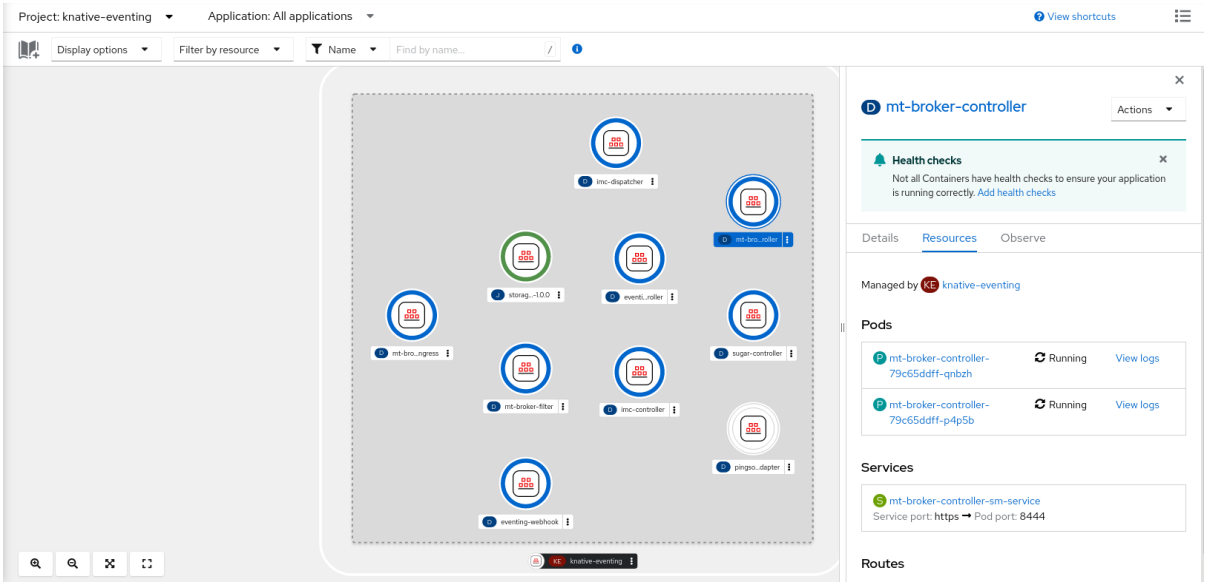
프로세스

1. **개발자** 화면에서 **+추가 → 브로커로 이동합니다**. **브로커** 페이지가 표시됩니다.
2. 선택 사항: 브로커 **의** 이름을 업데이트합니다. 이름을 업데이트하지 않으면 생성된 브로커의 이름이 **default** 입니다.
3. **생성**을 클릭합니다.

검증

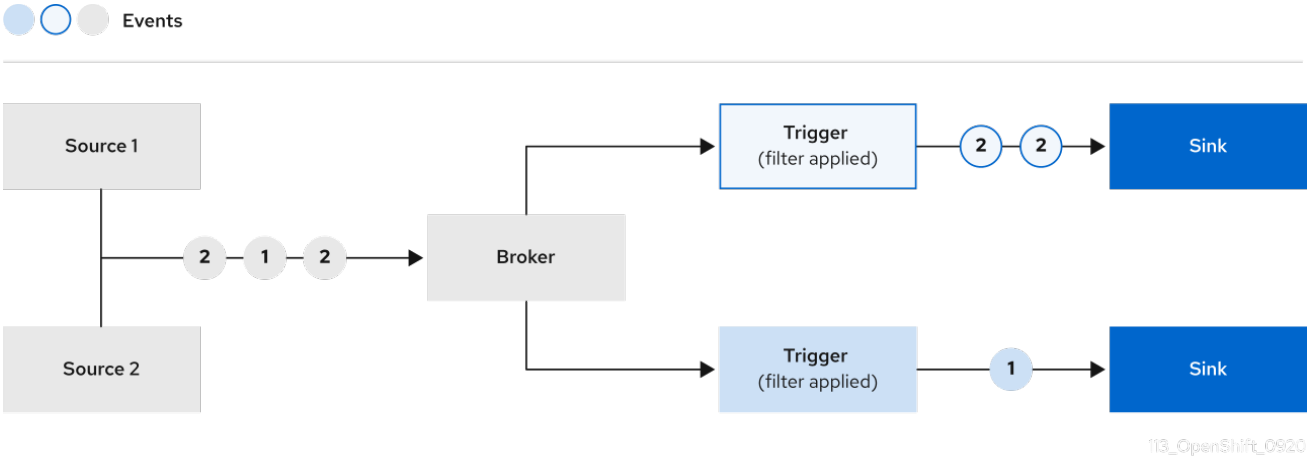
토폴로지 페이지에서 브로커 구성 요소를 확인하여 브로커가 생성되었는지 확인할 수 있습니다.

1. **개발자** 화면에서 **토폴로지**로 이동합니다.
2. **mt-broker-ingress,mt-broker-filter, mt-broker-controller** 구성 요소를 확인합니다.



4.3.6. 관리자 화면을 사용하여 브로커 생성

브로커는 트리거와 함께 이벤트 소스에서 이벤트 싱크로 이벤트를 전달할 수 있습니다. 이벤트는 이벤트 소스에서 HTTP **POST** 요청으로 브로커로 전송됩니다. 이벤트가 브로커에 진입하면 트리거를 사용하여 CloudEvent 속성으로 필터링하고 이벤트 싱크에 HTTP **POST** 요청으로 보낼 수 있습니다.



사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 **관리자** 화면에 있습니다.
- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 있습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔의 **관리자** 화면에서 **Serverless → Eventing**으로 이동합니다.
2. **생성** 목록에서 **브로커**를 선택합니다. 그러면 **브로커 생성** 페이지로 이동합니다.
3. 선택 사항: 브로커의 YAML 구성을 수정합니다.
4. **생성**을 클릭합니다.

4.3.7. 다음 단계

- 이벤트를 이벤트 싱크로 전달하지 못하는 경우 적용되는 이벤트 전달 매개변수를 구성합니다.

4.3.8. 추가 리소스

- 기본 브로커 클래스 구성
- Trigger
- 개발자 화면을 사용하여 브로커를 싱크에 연결

4.4. 기본 브로커 지원 채널 구성

채널 기반 브로커를 사용하는 경우 브로커의 기본 지원 채널 유형을 **InMemoryChannel** 또는 **KafkaChannel** 으로 설정할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform에 대한 관리자 권한이 있습니다.
- OpenShift Serverless Operator 및 Knative Eventing을 클러스터에 설치했습니다.
- OpenShift(**oc**) CLI가 설치되어 있습니다.
- Apache Kafka 채널을 기본 지원 채널 유형으로 사용하려면 클러스터에 **KnativeKafka** CR도 설치해야 합니다.

프로세스

1. **KnativeEventing** CR(사용자 정의 리소스)을 수정하여 **config-br-default-channel** 구성 맵에 대한 구성 세부 정보를 추가합니다.

```
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. **spec.config**에서 수정된 구성을 추가할 구성 맵을 지정할 수 있습니다.
2. 기본 지원 채널 유형 구성입니다. 이 예에서 클러스터의 기본 채널 구현은 **KafkaChannel** 입니다.
3. 브로커를 지원하는 Kafka 채널의 파티션 수입니다.
4. 브로커를 백업하는 Kafka 채널의 복제 요소입니다.

2. 업데이트된 **KnativeEventing** CR을 적용합니다.

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

4.5. 기본 브로커 클래스 구성

config-br-defaults 구성 맵을 사용하여 Knative Eventing의 기본 브로커 클래스 설정을 지정할 수 있습니다. 전체 클러스터 또는 하나 이상의 네임스페이스에 대한 기본 브로커 클래스를 지정할 수 있습니다. 현재 **MTChannelBasedBroker** 및 **Kafka** 브로커 유형이 지원됩니다.

사전 요구 사항

- OpenShift Container Platform에 대한 관리자 권한이 있습니다.
- OpenShift Serverless Operator 및 Knative Eventing을 클러스터에 설치했습니다.
- Apache Kafka에 Knative 브로커를 기본 브로커 구현으로 사용하려면 클러스터에 **KnativeKafka** CR도 설치해야 합니다.

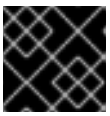
프로세스

- **KnativeEventing** 사용자 정의 리소스를 수정하여 **config-br-defaults** 구성 맵에 대한 구성 세부 정보를 추가합니다.

```
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 Eventing의 기본 브로커 클래스입니다.
2. **spec.config**에서 수정된 구성을 추가할 구성 맵을 지정할 수 있습니다.
3. **config-br-defaults** 구성 맵은 **spec.config** 설정 또는 브로커 클래스를 지정하지 않는 브로커의 기본 설정을 지정합니다.
4. 클러스터 전체 기본 브로커 클래스 구성입니다. 이 예에서 클러스터의 기본 브로커 클래스 구현은 **Kafka** 입니다.
5. **kafka-broker-config** 구성 맵은 Kafka 브로커의 기본 설정을 지정합니다. "추가 리소스" 섹션의 " Apache Kafka 설정에 대한 Knative 브로커 구성"을 참조하십시오.
6. **kafka-broker-config** 구성 맵이 있는 네임스페이스입니다.
7. 네임스페이스 범위의 기본 브로커 클래스 구성입니다. 이 예에서 **my-namespace** 네임스페이스의 기본 브로커 클래스 구현은 **MTChannelBasedBroker** 입니다. 여러 네임스페이스에 대한 기본 브로커 클래스 구현을 지정할 수 있습니다.
8. **config-br-default-channel** 구성 맵은 브로커의 기본 지원 채널을 지정합니다. "추가 리소스" 섹션의 "기본 브로커 백업 채널 구성"을 참조하십시오.
9. **config-br-default-channel** 구성 맵이 있는 네임스페이스입니다.



중요

네임스페이스별 기본값을 구성하면 클러스터 수준 설정을 덮어씁니다.

4.6. APACHE KAFKA용 KNATIVE 브로커 구현

프로덕션 지원 Knative Eventing 배포의 경우 Red Hat은 Apache Kafka에 Knative 브로커 구현을 사용하는 것이 좋습니다. 브로커는 Knative 브로커의 Apache Kafka 기본 구현으로, CloudEvents를 Kafka 인스턴스로 직접 보냅니다.

Knative 브로커에는 이벤트 저장 및 라우팅을 위한 Kafka와 기본 통합이 있습니다. 이를 통해 다른 브로커 유형보다 브로커 및 트리거 모델에 대한 Kafka와 보다 효과적으로 통합할 수 있으며 네트워크 홉이 줄어듭니다. Knative 브로커 구현의 기타 이점은 다음과 같습니다.

- at-least-once 전달 보장
- CloudEvents 파티션 확장에 따라 이벤트를 정렬된 제공
- 컨트롤 플레인 고가용성
- 수평으로 확장 가능한 데이터 플레인

Apache Kafka의 Knative 브로커 구현은 바이너리 콘텐츠 모드를 사용하여 들어오는 CloudEvents를 Kafka 레코드로 저장합니다. 즉, 모든 CloudEvent 속성 및 확장이 Kafka 레코드의 헤더로 매핑되지만 CloudEvent의 **data** 사양은 Kafka 레코드의 값에 해당합니다.

4.6.1. 기본 브로커 유형으로 구성되지 않은 경우 Apache Kafka 브로커 생성

OpenShift Serverless 배포가 Kafka 브로커를 기본 브로커 유형으로 사용하도록 구성되지 않은 경우 다음 절차 중 하나를 사용하여 Kafka 기반 브로커를 생성할 수 있습니다.

4.6.1.1. YAML을 사용하여 Apache Kafka 브로커 생성

YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 API를 사용하므로 선언적 및 재현 가능한 방식으로 애플리케이션을 설명할 수 있습니다. YAML을 사용하여 Kafka 브로커를 생성하려면 **Broker** 오브젝트를 정의하는 YAML 파일을 생성한 다음 **oc apply** 명령을 사용하여 적용해야 합니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing, **KnativeKafka** 사용자 정의 리소스가 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.

프로세스

1. Kafka 기반 브로커를 YAML 파일로 생성합니다.

```
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) 브로커 클래스입니다. 지정하지 않으면 브로커는 클러스터 관리자가 구성한 대로 기본 클래스를 사용합니다. Kafka 브로커를 사용하려면 이 값은 **Kafka** 여야 합니다.

(2) Apache Kafka의 Knative 브로커의 기본 구성 맵입니다. 이 구성 맵은 클러스터 관리자가 클러스터에서 Kafka 브로커 기능을 활성화할 때 생성됩니다.

2. Kafka 기반 브로커 YAML 파일을 적용합니다.

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

4.6.1.2. 외부에서 관리되는 Kafka 주제를 사용하는 Apache Kafka 브로커 생성

자체 내부 주제를 생성하지 않고 Kafka 브로커를 사용하려면 대신 외부 관리 Kafka 주제를 사용할 수 있습니다. 이렇게 하려면 **kafka.eventing.knative.dev/external.topic** 주석을 사용하는 Kafka **Broker** 오브젝트를 생성해야 합니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing, **KnativeKafka** 사용자 정의 리소스가 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Red Hat AMQ Streams 와 같은 Kafka 인스턴스에 액세스할 수 있으며 Kafka 주제를 생성했습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.

프로세스

1. Kafka 기반 브로커를 YAML 파일로 생성합니다.

```
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)
...
```

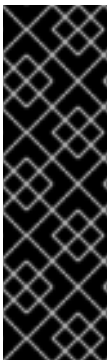
(1) 브로커 클래스입니다. 지정하지 않으면 브로커는 클러스터 관리자가 구성한 대로 기본 클래스를 사용합니다. Kafka 브로커를 사용하려면 이 값은 **Kafka** 여야 합니다.

(2) 사용하려는 Kafka 주제의 이름입니다.

2. Kafka 기반 브로커 YAML 파일을 적용합니다.

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

4.6.1.3. 격리된 데이터 플레인이 있는 Apache Kafka에 대한 Knative 브로커 구현



중요

격리된 데이터 플레인이 있는 Apache Kafka에 대한 Knative 브로커 구현은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

Apache Kafka의 Knative 브로커 구현에는 두 개의 플레인이 있습니다.

컨트롤 플레인

Kubernetes API와 통신하고 사용자 정의 오브젝트를 감시하고 데이터 플레인을 관리하는 컨트롤러로 구성됩니다.

데이터 플레인

들어오는 이벤트를 수신하고 Apache Kafka와 통신하고 이벤트를 이벤트 싱크에 전송하는 구성 요소의 컬렉션입니다. Apache Kafka 데이터 플레인에 대한 Knative 브로커 구현은 이벤트 흐름입니다. 구현은 **kafka-broker-receiver** 및 **kafka-broker-dispatcher** 배포로 구성됩니다.

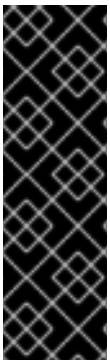
Kafka의 Broker 클래스를 구성할 때 Apache **Kafka** 의 Knative Broker 구현에서는 공유 데이터 플레인을 사용합니다. 즉, **knative-eventing** 네임스페이스의 **kafka-broker-receiver** 및 **kafka-broker-dispatcher** 배포는 클러스터의 모든 Apache Kafka 브로커에 사용됩니다.

그러나 **KafkaNamespaced** 의 Broker 클래스를 구성하면 Apache Kafka 브로커 컨트롤러는 브로커가 존재하는 각 네임스페이스에 대한 새 데이터 플레인을 생성합니다. 이 데이터 플레인은 해당 네임스페이스의 모든 **KafkaNamespaced** 브로커에서 사용합니다. 이를 통해 데이터 플레인을 격리하여 사용자 네임스페이스의 **kafka-broker-receiver** 및 **kafka-broker-dispatcher** 배포가 해당 네임스페이스의 브로커에만 사용됩니다.

중요

별도의 데이터 플레인을 보유하고 있기 때문에 이 보안 기능은 더 많은 배포를 생성하고 더 많은 리소스를 사용합니다. 이러한 격리 요구 사항이 없는 경우 **Kafka** 클래스와 함께 **일반** 브로커를 사용하십시오.

4.6.1.4. 격리된 데이터 플레인을 사용하는 Apache Kafka용 Knative 브로커 생성



중요

격리된 데이터 플레인이 있는 Apache Kafka에 대한 Knative 브로커 구현은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

KafkaNamespaced 브로커를 생성하려면 **eventing.knative.dev/broker.class** 주석을 **KafkaNamespaced** 로 설정해야 합니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing, **KnativeKafka** 사용자 정의 리소스가 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Red Hat AMQ Streams 와 같은 Apache Kafka 인스턴스에 액세스할 수 있으며 Kafka 주제를 생성했습니다.
- OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성할 수 있는 적절한 역할 및 권한으로 프로젝트를 생성하거나 프로젝트에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.

프로세스

1. YAML 파일을 사용하여 Apache Kafka 기반 브로커를 생성합니다.

```
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 격리된 데이터 플레인과 함께 Apache Kafka 브로커를 사용하려면 브로커 클래스 값이 **KafkaNamespaced** 여야 합니다.

2 3 참조된 **ConfigMap** 오브젝트 **my-config** 는 **Broker** 오브젝트와 동일한 네임스페이스에 있어야 합니다(이 경우 **my-namespace**).

2. Apache Kafka 기반 브로커 YAML 파일을 적용합니다.

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

중요

spec.config 의 **ConfigMap** 오브젝트는 **Broker** 오브젝트와 동일한 네임스페이스에 있어야 합니다.

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

KafkaNamespaced 클래스를 사용하여 첫 번째 **Broker** 오브젝트를 생성한 후 네임스페이스에 **kafka-broker-receiver** 및 **kafka-broker-dispatcher** 배포가 생성됩니다. 결과적으로 동일한 네임스페이스에 **KafkaNamespaced** 클래스가 있는 모든 브로커는 동일한 데이터 플레인을 사용합니다. **KafkaNamespaced** 클래스가 네임스페이스에 있는 브로커가 없으면 네임스페이스의 데이터 플레인이 삭제됩니다.

4.6.2. Apache Kafka 브로커 설정 구성

구성 맵을 생성하고 Kafka **Broker** 오브젝트에 이 구성 맵을 참조하여 Kafka 브로커의 복제 요소, 부트스트랩 서버 및 Kafka 브로커의 주제 파티션 수를 구성할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- OpenShift Serverless Operator, Knative Eventing 및 **KnativeKafka** CR(사용자 정의 리소스)이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성하거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성할 수 있는 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.

프로세스

1. **kafka-broker-config** 구성 맵을 수정하거나 다음 구성이 포함된 고유한 구성 맵을 생성합니다.

```
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. 구성 맵 이름입니다.
2. 구성 맵이 있는 네임스페이스입니다.
3. Kafka 브로커의 주제 파티션 수입니다. 이를 통해 브로커로 이벤트를 얼마나 빠르게 보낼 수 있는지 제어합니다. 파티션 수가 많을수록 더 많은 컴퓨팅 리소스가 필요합니다.
4. 주제 메시지의 복제 요소입니다. 이는 데이터 손실을 방지합니다. 복제 요소가 클수록 더 많은 컴퓨팅 리소스와 스토리지가 필요합니다.
5. 쉼표로 구분된 부트스트랩 서버 목록입니다. 이는 OpenShift Container Platform 클러스터 내부 또는 외부에 있을 수 있으며 브로커가 이벤트를 수신하고 이벤트를 보내는 Kafka 클러스터 목록입니다.

중요

default.topic.replication.factor 값은 클러스터의 Kafka 브로커 인스턴스 수보다 작거나 같아야 합니다. 예를 들어 Kafka 브로커가 하나만 있는 경우 **default.topic.replication.factor** 값은 **"1"** 을 초과해서는 안 됩니다.

Kafka 브로커 구성 맵의 예

```
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. 구성 맵을 적용합니다.

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

3. Kafka **Broker** 오브젝트의 구성 맵을 지정합니다.

Broker 오브젝트의 예

```
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. 브로커 이름입니다.
2. 브로커가 존재하는 네임스페이스입니다.
3. 브로커 클래스 주석입니다. 이 예에서 브로커는 클래스 값 Kafka를 사용하는 **Kafka** 브로커입니다.
4. 구성 맵 이름입니다.
5. 구성 맵이 있는 네임스페이스입니다.

4. 브로커를 적용합니다.

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

4.6.3. Apache Kafka에 대한 Knative 브로커 구현에 대한 보안 구성

Kafka 클러스터는 일반적으로 TLS 또는 SASL 인증 방법을 사용하여 보호됩니다. TLS 또는 SASL을 사용하여 보호된 Red Hat AMQ Streams 클러스터에 대해 작동하도록 Kafka 브로커 또는 채널을 구성할 수 있습니다.

참고

SASL과 TLS를 함께 활성화하는 것이 좋습니다.

4.6.3.1. Apache Kafka 브로커에 대한 TLS 인증 구성

TLS(*Transport Layer Security*)는 Apache Kafka 클라이언트와 서버에서 Knative와 Kafka 간의 트래픽을 암호화하고 인증을 위해 사용됩니다. TLS는 Apache Kafka에 대한 Knative 브로커 구현에 지원되는 유일한 트래픽 암호화 방법입니다.

사전 요구 사항

- OpenShift Container Platform에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- OpenShift Serverless Operator, Knative Eventing, **KnativeKafka** CR이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Kafka 클러스터 CA 인증서가 **.pem** 파일로 저장되어 있습니다.
- Kafka 클러스터 클라이언트 인증서와 **.pem** 파일로 저장된 키가 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. **knative-eventing** 네임스페이스에서 인증서 파일을 시크릿으로 생성합니다.

```
$ 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
```

중요

키 이름 **ca.crt**,**user.crt**, **user.key**를 사용합니다. 이 값은 변경하지 마십시오.

2. **KnativeKafka** CR을 편집하고 **broker** 사양의 보안에 대한 참조를 추가합니다.

```
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. Apache Kafka 브로커에 대한 SASL 인증 구성

SASL(*Simple Authentication and Security Layer*)은 Apache Kafka에서 인증에 사용됩니다. 클러스터에서 SASL 인증을 사용하는 경우 Kafka 클러스터와 통신하기 위해 Knative에 인증 정보를 제공해야 합니다. 그렇지 않으면 이벤트를 생성하거나 사용할 수 없습니다.

사전 요구 사항

- OpenShift Container Platform에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- OpenShift Serverless Operator, Knative Eventing, **KnativeKafka** CR이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Kafka 클러스터의 사용자 이름과 암호가 있습니다.
- **PLAIN**,**SCRAM-SHA-256** 또는 **SCRAM-SHA-512** 와 같이 사용할 SASL 메커니즘을 선택했습니다.
- TLS가 활성화된 경우 Kafka 클러스터의 **ca.crt** 인증서 파일도 필요합니다.
- OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. **knative-eventing** 네임스페이스에서 인증서 파일을 시크릿으로 생성합니다.

```
$ 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"
```

   - 키 이름 **ca.crt**,**password**, **sasl.mechanism** 을 사용합니다. 이 값은 변경하지 마십시오.
   - 공용 CA 인증서와 함께 SASL을 사용하려면 시크릿을 생성할 때 **ca.crt** 인수 대신 **tls.enabled=true** 플래그를 사용해야 합니다. 예를 들면 다음과 같습니다.

```
$ 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. **KnativeKafka** CR을 편집하고 **broker** 사양의 보안에 대한 참조를 추가합니다.

```
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. 추가 리소스

- Red Hat AMQ Streams 문서
- Kafka의 TLS 및 SASL

4.7. 브로커 관리

브로커를 생성한 후에는 Knative(**kn**) CLI 명령을 사용하거나 OpenShift Container Platform 웹 콘솔에서 수정하여 브로커를 관리할 수 있습니다.

4.7.1. CLI를 사용하여 브로커 관리

Knative(**kn**) CLI는 기존 브로커를 설명하고 나열하는 데 사용할 수 있는 명령을 제공합니다.

4.7.1.1. Knative CLI를 사용하여 기존 브로커 나열

Knative(**kn**) CLI를 사용하여 브로커를 나열하면 간소화되고 직관적인 사용자 인터페이스가 제공됩니다. **kn broker list** 명령을 사용하여 Knative CLI를 사용하여 클러스터의 기존 브로커를 나열할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

프로세스

- 기존 브로커를 모두 나열합니다.

```
$ kn broker list
```

출력 예

```
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. Knative CLI를 사용하여 기존 브로커 설명

Knative(**kn**) CLI를 사용하여 브로커를 설명하는 것은 간소화되고 직관적인 사용자 인터페이스를 제공합니다. **kn broker describe** 명령을 사용하여 Knative CLI를 사용하여 클러스터의 기존 브로커에 대한 정보를 출력할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

프로세스

- 기존 브로커를 설명합니다.

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

기본 브로커를 사용하는 명령의 예

```
$ kn broker describe default
```

출력 예

```
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. 개발자 화면을 사용하여 브로커를 싱크에 연결

트리거를 생성하여 OpenShift Container Platform **개발자** 화면의 이벤트 싱크에 브로커를 연결할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Serving, Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 **개발자** 화면으로 갑니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Knative 서비스 또는 채널과 같은 싱크를 생성했습니다.
- 브로커를 생성했습니다.

프로세스

1. **토폴로지** 보기에서 생성한 브로커를 가리킵니다. 화살표가 나타납니다. 브로커에 연결하려는 싱크로 화살표를 끕니다. 이 작업은 **트리거 추가** 대화 상자가 열립니다.

2. **트리거 추가** 대화 상자에서 트리거의 이름을 입력하고 **추가** 를 클릭합니다.

검증

토폴로지 페이지를 확인하여 브로커가 싱크에 연결되어 있는지 확인할 수 있습니다.

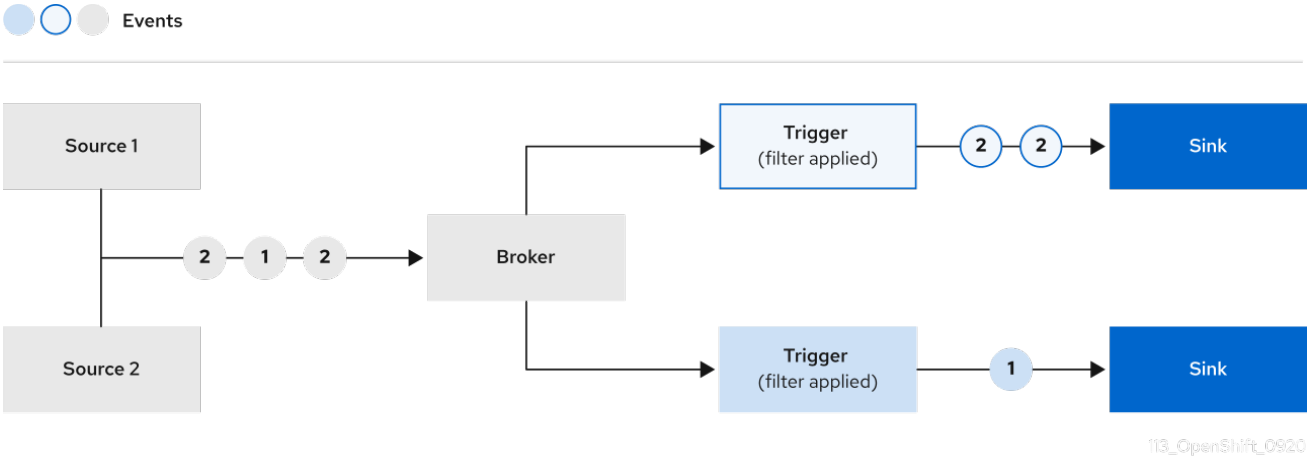
1. **개발자** 화면에서 **토폴로지**로 이동합니다.

2. 브로커를 싱크에 연결하는 행을 클릭하여 세부 **정보 패널에서 트리거에 대한 세부** 정보를 확인합니다.

5장. TRIGGER

5.1. 트리거 개요

브로커는 트리거와 함께 이벤트 소스에서 이벤트 싱크로 이벤트를 전달할 수 있습니다. 이벤트는 이벤트 소스에서 HTTP **POST** 요청으로 브로커로 전송됩니다. 이벤트가 브로커에 진입하면 트리거를 사용하여 CloudEvent 속성으로 필터링하고 이벤트 싱크에 HTTP **POST** 요청으로 보낼 수 있습니다.



Apache Kafka에 Knative 브로커를 사용하는 경우 트리거에서 이벤트 싱크로 이벤트 전달 순서를 구성할 수 있습니다. 트리거의 이벤트 전달 순서 구성을 참조하십시오.

5.1.1. 트리거에 대한 이벤트 전달 순서 구성

Kafka 브로커를 사용하는 경우 트리거에서 이벤트 싱크로 이벤트 전달 순서를 구성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing, Knative Kafka가 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Kafka 브로커는 클러스터에서 사용할 수 있으며 Kafka 브로커를 생성했습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- OpenShift(**oc**) CLI가 설치되어 있습니다.

프로세스

1. **Trigger** 오브젝트를 생성하거나 수정하고 **kafka.eventing.knative.dev/delivery.order** 주석을 설정합니다.

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

지원되는 소비자 제공 보장은 다음과 같습니다.

순서가 지정되지 않음

순서가 지정되지 않은 소비자는 적절한 오프셋 관리를 유지하면서 정렬되지 않은 메시지를 전달하는 비차단 소비자입니다.

주문됨

순서가 지정된 소비자는 파티션의 다음 메시지를 전달하기 전에 CloudEvent 구독자의 성공적인 응답을 기다리는 파티션별 차단 소비자입니다.
기본 주문 보증은 **순서가 지정되지 않습니다**.

2. **Trigger** 오브젝트를 적용합니다.

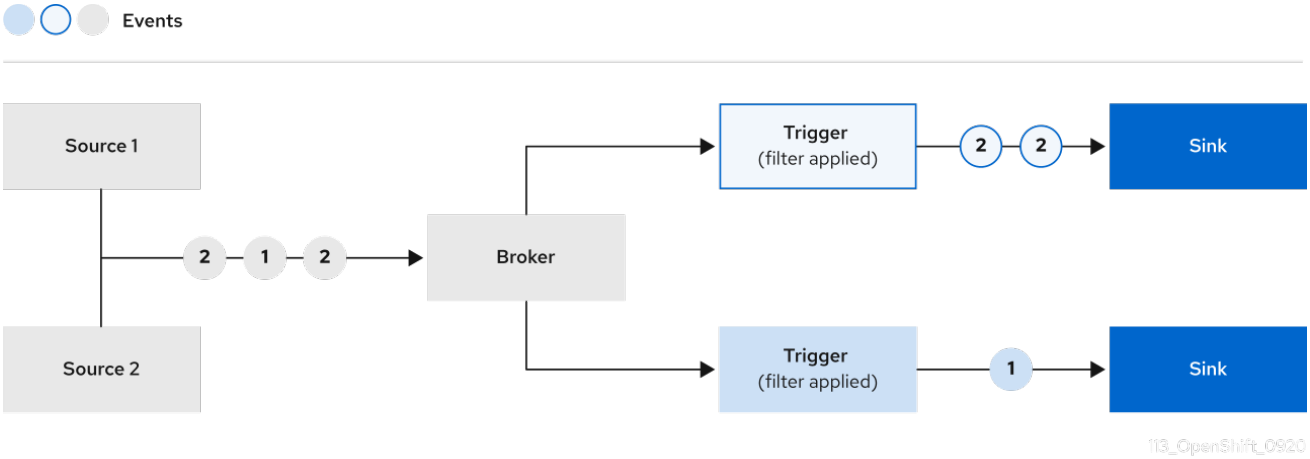
```
$ oc apply -f <filename>
```

5.1.2. 다음 단계

- 이벤트를 이벤트 싱크로 전달하지 못하는 경우 적용되는 이벤트 전달 매개변수를 구성합니다.

5.2. 트리거 생성

브로커는 트리거와 함께 이벤트 소스에서 이벤트 싱크로 이벤트를 전달할 수 있습니다. 이벤트는 이벤트 소스에서 HTTP **POST** 요청으로 브로커로 전송됩니다. 이벤트가 브로커에 진입하면 트리거를 사용하여 CloudEvent 속성으로 필터링하고 이벤트 싱크에 HTTP **POST** 요청으로 보낼 수 있습니다.



5.2.1. 관리자 화면을 사용하여 트리거 생성

OpenShift Container Platform 웹 콘솔을 사용하면 간소화되고 직관적인 사용자 인터페이스를 통해 트리거를 생성할 수 있습니다. Knative Eventing이 클러스터에 설치되고 브로커를 생성한 후 웹 콘솔을 사용하여 트리거를 생성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 **관리자** 화면에 있습니다.

- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- Knative 브로커를 생성했습니다.
- 구독자로 사용할 Knative 서비스를 생성했습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔의 **관리자** 화면에서 **Serverless → Eventing**으로 이동합니다.
2. **브로커** 탭에서 트리거를 추가할 브로커의 옵션 메뉴  를 선택합니다.
3. 목록에서 **트리거 추가**를 클릭합니다.
4. **트리거 추가** 대화 상자에서 트리거에 대한 **구독자**를 선택합니다. 구독자는 브로커에서 이벤트를 수신하는 Knative 서비스입니다.
5. **추가**를 클릭합니다.

5.2.2. 개발자 화면을 사용하여 트리거 생성

OpenShift Container Platform 웹 콘솔을 사용하면 간소화되고 직관적인 사용자 인터페이스를 통해 트리거를 생성할 수 있습니다. Knative Eventing이 클러스터에 설치되고 브로커를 생성한 후 웹 콘솔을 사용하여 트리거를 생성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Serving, Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인했습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 트리거에 연결할 브로커 및 Knative 서비스 또는 기타 이벤트 싱크를 생성했습니다.

프로세스

1. **개발자** 화면에서 **토폴로지** 페이지로 이동합니다.
2. 트리거를 생성할 브로커 위로 마우스 커서를 이동한 후 화살표를 끕니다. **트리거 추가** 옵션이 표시됩니다.
3. **트리거 추가**를 클릭합니다.
4. **구독자** 목록에서 싱크를 선택합니다.
5. **추가**를 클릭합니다.

검증

- 서브스크립션이 생성되면 **Topology** 페이지에서 브로커를 이벤트 싱크에 연결하는 선으로 표시됩니다.

트리거 삭제

1. **개발자** 화면에서 **토폴로지** 페이지로 이동합니다.
2. 삭제할 트리거를 클릭합니다.
3. **작업** 컨텍스트 메뉴에서 **트리거 삭제**를 선택합니다.

5.2.3. Knative CLI를 사용하여 트리거 생성

kn trigger create 명령을 사용하여 트리거를 생성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

- 트리거를 생성합니다.

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

또는 트리거를 생성하고 브로커 삽입을 사용하여 **default** 브로커를 동시에 생성할 수 있습니다.

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

기본적으로 트리거는 브로커에 전송된 모든 이벤트를 해당 브로커에 가입된 싱크로 전달합니다. 트리거에 **--filter** 특성을 사용하면 브로커의 이벤트를 필터링하여 구독자에게 정의된 기준에 따라 일부 이벤트만 제공할 수 있습니다.

5.3. 명령줄에서 트리거 나열

Knative(**kn**) CLI를 사용하여 트리거를 나열하면 간소화되고 직관적인 사용자 인터페이스가 제공됩니다.

5.3.1. Knative CLI를 사용하여 트리거 나열

kn trigger list 명령을 사용하여 클러스터의 기존 트리거를 나열할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

프로세스

1. 사용 가능한 트리거 목록을 인쇄합니다.

```
$ kn trigger list
```

출력 예

```
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. 선택 사항: JSON 형식으로 된 트리거 목록을 인쇄합니다.

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

5.4. 명령줄에서 트리거 설명

Knative(**kn**) CLI를 사용하여 트리거를 설명하면 간소화되고 직관적인 사용자 인터페이스가 제공됩니다.

5.4.1. Knative CLI를 사용하여 트리거 설명

kn trigger describe 명령을 사용하여 Knative CLI를 사용하여 클러스터의 기존 트리거에 대한 정보를 출력할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.
- 트리거를 생성했습니다.

프로세스

- 명령을 입력합니다.

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

출력 예

```
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. 트리거를 싱크에 연결

트리거를 싱크로 보내기 전에 브로커의 이벤트가 필터링되도록 트리거를 싱크에 연결할 수 있습니다. 트리거에 연결된 싱크는 **Trigger** 오브젝트의 리소스 사양에 **구독자** 로 구성됩니다.

Apache Kafka 싱크에 연결된 Trigger 오브젝트의 예

```
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 싱크에 연결되어 있는 트리거의 이름입니다.

2 **KafkaSink** 오브젝트의 이름입니다.

5.6. 명령줄에서 트리거 필터링

Knative(**kn**) CLI를 사용하여 트리거를 사용하여 이벤트를 필터링하면 간소화되고 직관적인 사용자 인터페이스가 제공됩니다. **kn trigger create** 명령을 적절한 플래그와 함께 사용하여 트리거를 사용하여 이벤트를 필터링할 수 있습니다.

5.6.1. Knative CLI를 사용하여 트리거로 이벤트 필터링

다음 트리거 예제에서는 **type: dev.knative.samples.helloworld** 속성이 있는 이벤트만 이벤트 싱크로 전송됩니다.

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

여러 특성을 사용하여 이벤트를 필터링할 수도 있습니다. 다음 예제에서는 유형, 소스 및 확장 특성을 사용하여 이벤트를 필터링하는 방법을 보여줍니다.

```
$ 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. 명령줄에서 트리거 업데이트

Knative(**kn**) CLI를 사용하여 트리거를 업데이트하면 간소화되고 직관적인 사용자 인터페이스가 제공됩니다.

5.7.1. Knative CLI를 사용하여 트리거 업데이트

kn trigger update 명령을 특정 플래그와 함께 사용하여 트리거의 특성을 업데이트할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

- 태그를 업데이트합니다.

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

  - 트리거를 업데이트하여 수신 이벤트와 정확히 일치하는 이벤트 특성을 필터링할 수 있습니다. 예를 들면 **type** 특성을 사용합니다.

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

  - 트리거에서 필터 특성을 제거할 수 있습니다. 예를 들어 **type** 키를 사용하여 필터 특성을 제거할 수 있습니다.

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

  - **--sink** 매개변수를 사용하여 트리거의 이벤트 싱크를 변경할 수 있습니다.

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

5.8. 명령줄에서 트리거 삭제

Knative(**kn**) CLI를 사용하여 트리거를 삭제하면 간소화되고 직관적인 사용자 인터페이스가 제공됩니다.

5.8.1. Knative CLI를 사용하여 트리거 삭제

kn trigger delete 명령을 사용하여 트리거를 삭제할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

- 트리거를 삭제합니다.

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

검증

1. 기존 트리거를 나열합니다.

```
$ kn trigger list
```

2. 트리거가 더 이상 존재하지 않는지 확인합니다.

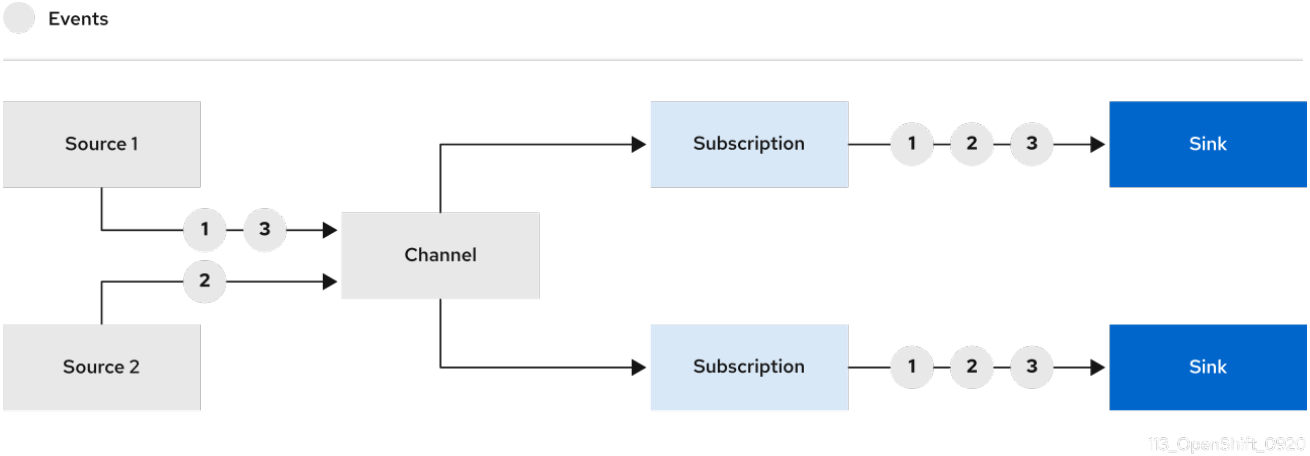
출력 예

```
No triggers found.
```

6장. 채널

6.1. 채널 및 서브스크립션

채널은 단일 이벤트 전달 및 지속성 계층을 정의하는 사용자 정의 리소스입니다. 이벤트 소스 또는 생산자에서 채널로 이벤트를 보낸 후에는 서브스크립션을 사용하여 이러한 이벤트를 여러 Knative 서비스 또는 기타 싱크로 보낼 수 있습니다.

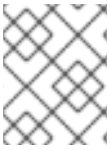


지원되는 **Channel** 오브젝트를 인스턴스화하여 채널을 생성하고 **Subscription** 오브젝트의 **delivery** 사양을 수정하여 재전송 시도를 구성할 수 있습니다.

Channel 오브젝트를 생성한 후에는 변경 승인 Webhook에서 기본 채널 구현을 기반으로 **Channel** 오브젝트에 일련의 **spec.channelTemplate** 속성을 추가합니다. 예를 들어 **InMemoryChannel** 기본 구현의 경우 **Channel** 오브젝트는 다음과 같습니다.

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

그런 다음 채널 컨트롤러에서 **spec.channelTemplate** 구성에 따라 지원 채널 인스턴스를 생성합니다.



참고

spec.channelTemplate 속성은 사용자가 아닌 기본 채널 메커니즘에 의해 설정되기 때문에 생성 후에는 변경할 수 없습니다.

이 메커니즘을 앞의 예제와 함께 사용하면 일반 지원 채널과 **InMemoryChannel** 채널이라는 두 개의 오브젝트가 생성됩니다. 다른 기본 채널 구현을 사용하는 경우 **InMemoryChannel**이 구현에 고유한 채널로 교체됩니다. 예를 들어 Apache Kafka의 Knative 브로커를 사용하면 **KafkaChannel** 채널이 생성됩니다.

지원 채널은 서브스크립션을 사용자가 생성한 채널 오브젝트에 복사하고 지원 채널의 상태를 반영하도록 사용자가 생성한 채널 오브젝트의 상태를 설정하는 프록시 역할을 합니다.

6.1.1. 채널 구현 유형

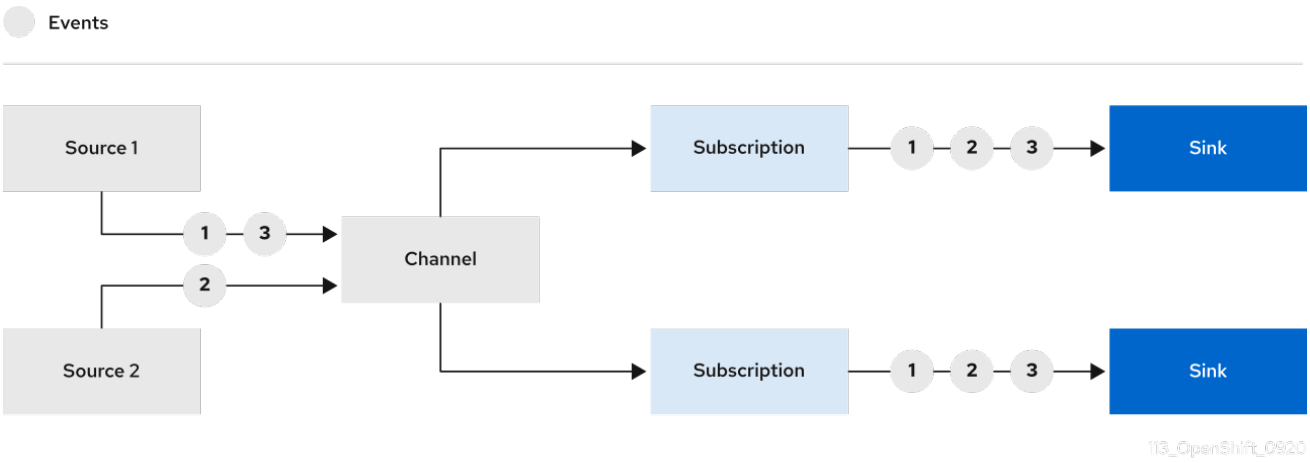
OpenShift Serverless에서는 **InMemoryChannel** 및 **KafkaChannel** 채널 구현을 지원합니다. **InMemoryChannel** 채널은 제한 사항으로만 개발 용도로 사용하는 것이 좋습니다. 프로덕션 환경에 **KafkaChannel** 채널을 사용할 수 있습니다.

다음은 **InMemoryChannel** 유형 채널에 대한 제한 사항입니다.

- 이벤트 지속성은 제공되지 않습니다. Pod가 다운되면 해당 Pod의 이벤트도 손실됩니다.
- **InMemoryChannel** 채널에서는 이벤트에 순서를 지정하지 않으므로 해당 채널에서 두 개의 이벤트를 동시에 수신하는 경우 이벤트를 순서와 관계없이 구독자에게 전달할 수 있습니다.
- 구독자가 이벤트를 거부하면 기본적으로 재전송을 시도하지 않습니다. **Subscription** 오브젝트의 **delivery** 사양을 수정하여 재전송 시도 횟수를 구성할 수 있습니다.

6.2. 채널 생성

채널은 단일 이벤트 전달 및 지속성 계층을 정의하는 사용자 정의 리소스입니다. 이벤트 소스 또는 생산자에서 채널로 이벤트를 보낸 후에는 서브스크립션을 사용하여 이러한 이벤트를 여러 Knative 서비스 또는 기타 싱크로 보낼 수 있습니다.



지원되는 **Channel** 오브젝트를 인스턴스화하여 채널을 생성하고 **Subscription** 오브젝트의 **delivery** 사양을 수정하여 재전송 시도를 구성할 수 있습니다.

6.2.1. 관리자 화면을 사용하여 채널 생성

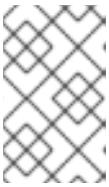
Knative Eventing이 클러스터에 설치되면 관리자 화면을 사용하여 채널을 생성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 **관리자** 화면에 있습니다.
- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 있습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔의 **관리자** 화면에서 **Serverless → Eventing**으로 이동합니다.
2. **생성** 목록에서 **채널**을 선택합니다. 그러면 **채널** 페이지로 이동합니다.
3. **유형** 목록에서 생성할 **Channel** 오브젝트 유형을 선택합니다.



참고

현재 **InMemoryChannel** 채널 오브젝트만 기본적으로 지원됩니다. OpenShift Serverless에 Apache Kafka에 Knative 브로커 구현을 설치한 경우 Apache Kafka의 Knative 채널을 사용할 수 있습니다.

4. **생성**을 클릭합니다.

6.2.2. 개발자 화면을 사용하여 채널 생성

OpenShift Container Platform 웹 콘솔을 사용하면 간소화되고 직관적인 사용자 인터페이스를 통해 채널을 생성할 수 있습니다. Knative Eventing이 클러스터에 설치되면 웹 콘솔을 사용하여 채널을 생성할 수 있습니다.

사전 요구 사항

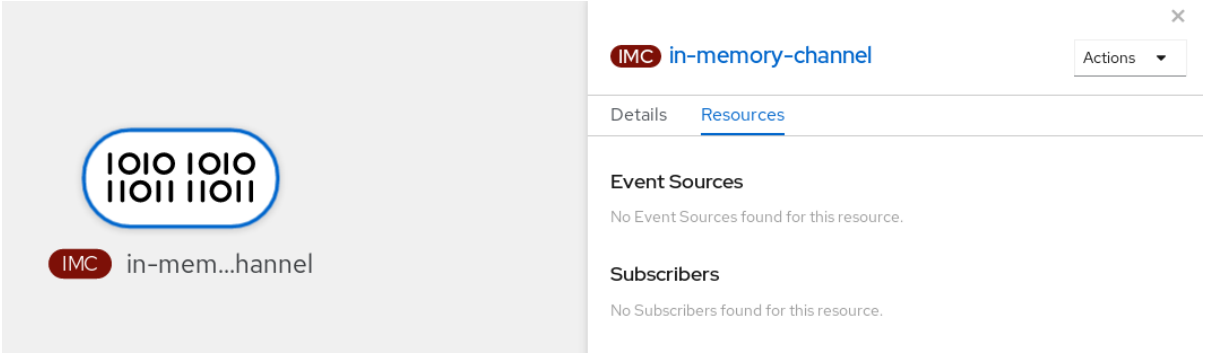
- OpenShift Container Platform 웹 콘솔에 로그인했습니다.
- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

1. **개발자** 화면에서 **+추가 → 채널**로 이동합니다.
2. **유형** 목록에서 생성할 **Channel** 오브젝트 유형을 선택합니다.
3. **생성**을 클릭합니다.

검증

- **토폴로지** 페이지로 이동하여 채널이 있는지 확인합니다.



6.2.3. Knative CLI를 사용하여 채널 생성

Knative(**kn**) CLI를 사용하여 채널을 생성하면 YAML 파일을 직접 수정하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다. **kn channel create** 명령을 사용하여 채널을 생성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

- 채널을 생성합니다.

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

채널 유형은 선택 사항이지만 지정된 위치는 **Group:Version:Kind** 형식으로 지정해야 합니다. 예를 들면 **InMemoryChannel** 오브젝트를 생성할 수 있습니다.

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

출력 예

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

검증

- 채널이 존재하는지 확인하려면 기존 채널을 나열하고 출력을 검사합니다.

```
$ kn channel list
```

출력 예

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

채널 삭제

- 채널을 삭제합니다.

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

6.2.4. YAML을 사용하여 기본 구현 채널 생성

YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 API를 사용하므로 선언적 및 재현 가능한 방식으로 채널을 설명할 수 있습니다. YAML을 사용하여 서버리스 채널을 생성하려면 **Channel** 오브젝트를 정의하는 YAML 파일을 생성한 다음 **oc apply** 명령을 사용하여 적용해야 합니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

1. **Channel** 오브젝트를 YAML 파일로 생성합니다.

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

2. YAML 파일을 적용합니다.

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

6.2.5. YAML을 사용하여 Apache Kafka의 채널 생성

YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 API를 사용하므로 선언적 및 재현 가능한 방식으로 채널을 설명할 수 있습니다. Kafka 채널을 생성하여 Kafka 주제에서 지원하는 Knative Eventing 채널을 생성할 수 있습니다. YAML을 사용하여 Kafka 채널을 생성하려면 **KafkaChannel** 오브젝트를 정의하는 YAML 파일을 생성한 다음 **oc apply** 명령을 사용하여 적용해야 합니다.

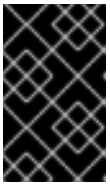
사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing, **KnativeKafka** 사용자 정의 리소스가 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

1. **KafkaChannel** 오브젝트를 YAML 파일로 생성합니다.

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



중요

OpenShift Serverless의 **KafkaChannel** 오브젝트에 대한 API의 **v1beta1** 버전만 지원됩니다. 이 버전은 더 이상 사용되지 않으므로 이 API의 **v1alpha1** 버전을 사용하지 마십시오.

2. **KafkaChannel** YAML 파일을 적용합니다.

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

6.2.6. 다음 단계

- 채널을 생성한 후 싱크 에서 이벤트를 수신할 수 있도록 채널을 싱크에 연결할 수 있습니다.
- 이벤트를 이벤트 싱크로 전달하지 못하는 경우 적용되는 이벤트 전달 매개변수를 구성합니다.

6.3. 싱크에 채널 연결

이벤트 소스 또는 생산자에서 채널로 전송된 이벤트는 *서브스크립션*을 사용하여 하나 이상의 싱크로 전달할 수 있습니다. **서브스크립션** 오브젝트를 구성하여 해당 채널로 전송된 이벤트를 사용하는 채널과 싱크(인터라고도 함)를 지정할 수 있습니다.

6.3.1. 개발자 화면을 사용하여 서브스크립션 생성

채널과 이벤트 싱크를 생성한 후에는 이벤트 전달을 활성화하는 서브스크립션을 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔을 사용하면 간소화되고 직관적인 사용자 인터페이스를 통해 서브스크립션을 생성할 수 있습니다.

사전 요구 사항

- *OpenShift Serverless Operator, Knative Serving, Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.*
- *웹 콘솔에 로그인했습니다.*
- *이벤트 싱크(예: Knative 서비스) 및 채널을 생성했습니다.*
- *프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.*

프로세스

1. ***개발자*** *화면에서* ***토폴로지*** *페이지로 이동합니다.*
2. *다음 방법 중 하나를 사용하여 서브스크립션을 생성합니다.*
   a. *서브스크립션을 생성할 채널 위로 마우스 커서를 이동하고 화살표를 끕니다.* ***서브스크립션 추가*** *옵션이 표시됩니다.*

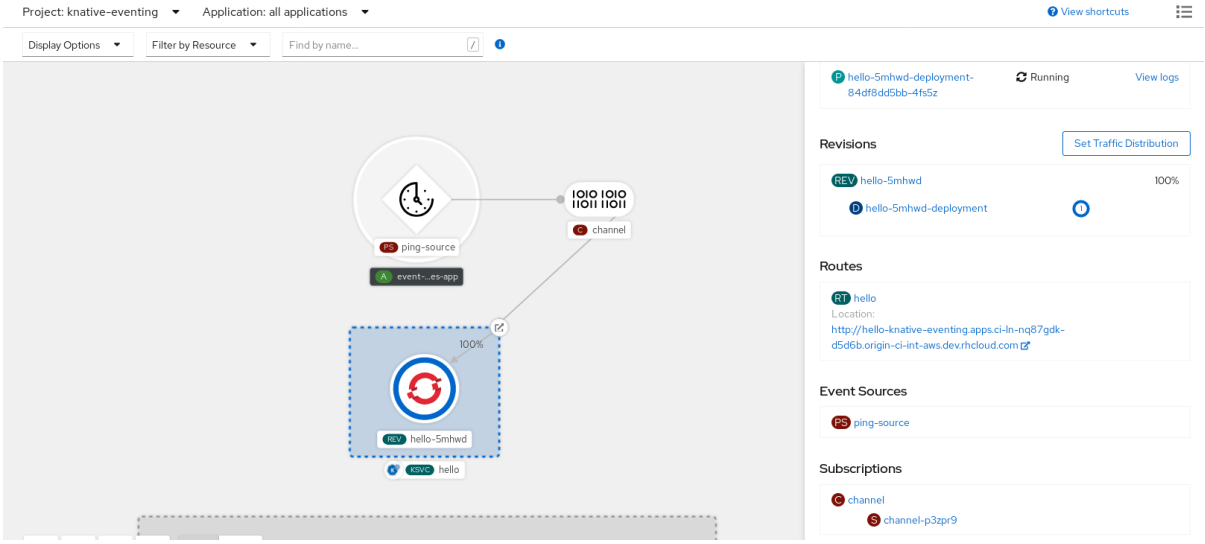
i. *구독자* 목록에서 싱크를 선택합니다.

ii. *추가*를 클릭합니다.

b. *채널과 동일한 네임스페이스 또는 프로젝트의 **토폴로지** 보기에서 서비스를 사용할 수 있는 경우, 서브스크립션을 생성할 채널을 클릭하고 화살표를 서비스로 직접 끌어 채널에서 해당 서비스로의 서브스크립션을 즉시 생성합니다.*

검증

- *서브스크립션이 생성되면 **토폴로지** 보기에 채널을 서비스에 연결하는 선이 표시되어 이를 확인할 수 있습니다.*



6.3.2. YAML 을 사용하여 서브스크립션 생성

*채널과 이벤트 싱크를 생성한 후에는 이벤트 전달을 활성화하는 서브스크립션을 생성할 수 있습니다. YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 API를 사용하므로 선언적 및 재현 가능한 방식으로 서브스크립션을 설명할 수 있습니다. YAML 을 사용하여 서브스크립션을 생성하려면 **Subscription** 오브젝트를 정의하는 YAML 파일을 생성한 다음 **oc apply** 명령을 사용하여 적용해야 합니다.*

사전 요구 사항

- *OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.*
- *OpenShift CLI(**oc**)를 설치합니다.*

- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

- **Subscription** 오브젝트를 생성합니다.
  - YAML 파일을 생성하고 다음 샘플 코드를 여기에 복사합니다.

```
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) 서브스크립션 이름입니다.

(2) 서브스크립션이 연결되는 채널의 구성 설정입니다.

(3) 이벤트 전달을 위한 구성 설정입니다. 이 설정은 구독자에게 전달할 수 없는 이벤트에 어떤 일이 발생하는지 스브스크립션에 알립니다. 이 값을 구성하면 사용할 수 없는 이벤트가 **deadLetterSink**로 전송됩니다. 이벤트가 삭제되고 이벤트 재전송이 시도되지 않으며 시스템에 오류가 기록됩니다. **deadLetterSink** 값은 대상이어야 합니다.

(4) 구독자에 대한 구성 설정입니다. 채널에서 이벤트가 전달되는 이벤트 싱크입니다.

  - YAML 파일을 적용합니다.

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

6.3.3. Knative CLI를 사용하여 서브스크립션 생성

채널과 이벤트 싱크를 생성한 후에는 이벤트 전달을 활성화하는 서브스크립션을 생성할 수 있습니다. Knative(**kn**) CLI를 사용하여 서브스크립션을 생성하면 YAML 파일을 직접 수정하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다. **kn subscription create** 명령을 적절한 플래그와 함께 사용하여 서브스크립션을 생성할 수 있습니다.

사전 요구 사항

- *OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.*
- *Knative(**kn**) CLI가 설치되어 있습니다.*
- *프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.*

프로세스

- *싱크를 채널에 연결하는 서브스크립션을 생성합니다.*

```
$ 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. ***--channel은** 처리해야 하는 클라우드 이벤트의 소스를 지정합니다. 채널 이름을 제공해야 합니다. 채널 사용자 정의 리소스에서 지원하는 기본 **InMemoryChannel** 채널을 사용하지 않는 경우 **Channel** 이름 앞에 지정된 채널 유형에 대해 **<group:version:kind>**를 추가해야 합니다. 예를 들어 Apache Kafka 백업 채널의 경우 **messaging.knative.dev:v1beta1:KafkaChannel** 이 됩니다.*

2. ***--sink는** 이벤트를 전달해야 하는 대상 대상을 지정합니다. 기본적으로 **<sink_name>**은 서브스크립션과 동일한 네임스페이스에서 이 이름의 Knative 서비스로 해석됩니다. 다음 접두사 중 하나를 사용하여 싱크 유형을 지정할 수 있습니다.*

   ksvc
   Knative 서비스입니다.
   channel
   대상으로 사용해야 하는 채널입니다. 여기에서는 기본 채널 유형만 참조할 수 있습니다.
   broker
   Eventing 브로커입니다.

3. *선택 사항: **--sink-dead-letter**는 이벤트를 전달하지 못하는 경우 이벤트를 전송해야 하는 싱크를 지정하는 데 사용할 선택적 플래그입니다. 자세한 내용은 OpenShift Serverless Event 제공 설명서를 참조하십시오.*

   명령 예

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

   출력 예

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

검증

- *채널이 서브스크립션을 통해 이벤트 싱크 또는 구독자에 연결되어 있는지 확인하려면 기존 서브스크립션을 나열하고 출력을 검사합니다.*

```
$ kn subscription list
```

출력 예

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

서브스크립션 삭제

- 서브스크립션을 삭제합니다.

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

6.3.4. 관리자 화면을 사용하여 서브스크립션 생성

구독자 라고도 하는 채널과 이벤트 싱크를 생성한 후 서브스크립션을 생성하여 이벤트 전달을 활성화할 수 있습니다. 서브스크립션은 이벤트를 전달할 채널과 구독자를 지정하는 **Subscription** 오브젝트를 구성하여 생성됩니다. 실패 처리 방법과 같은 구독자별 옵션을 지정할 수도 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 **관리자** 화면에 있습니다.
- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- Knative 채널을 생성했습니다.
- 구독자로 사용할 Knative 서비스를 생성했습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔의 **관리자** 화면에서 **Serverless → Eventing**으로 이동합니다.
2. **채널** 탭에서 서브스크립션을 추가할 채널의 옵션 메뉴  를 선택합니다.
3. 목록에서 **서브스크립션 추가**를 클릭합니다.
4. **서브스크립션 추가** 대화 상자에서 서브스크립션에 대한 **구독자**를 선택합니다. 구독자는 채널에서 이벤트를 수신하는 Knative 서비스입니다.
5. **추가**를 클릭합니다.

6.3.5. 다음 단계

- 이벤트를 이벤트 싱크로 전달하지 못하는 경우 적용되는 이벤트 전달 매개변수를 구성합니다.

6.4. 기본 채널 구현

default-ch-webhook 구성 맵을 사용하여 Knative Eventing의 기본 채널 구현을 지정할 수 있습니다. 전체 클러스터 또는 하나 이상의 네임스페이스에 대한 기본 채널 구현을 지정할 수 있습니다. 현재 **InMemoryChannel** 및 **KafkaChannel** 채널 유형이 지원됩니다.

6.4.1. 기본 채널 구현 구성

사전 요구 사항

- OpenShift Container Platform에 대한 관리자 권한이 있습니다.
- OpenShift Serverless Operator 및 Knative Eventing을 클러스터에 설치했습니다.
- Apache Kafka에 Knative 채널을 기본 채널 구현으로 사용하려면 클러스터에 **KnativeKafka** CR도 설치해야 합니다.

프로세스

- **KnativeEventing** 사용자 정의 리소스를 수정하여 **default-ch-webhook** 구성 맵에 대한 구성 세부 정보를 추가합니다.

```
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) **spec.config**에서 수정된 구성을 추가할 구성 맵을 지정할 수 있습니다.

(2) **default-ch-webhook** 구성 맵을 사용하여 클러스터 또는 하나 이상의 네임스페이스에 대한 기본 채널 구현을 지정할 수 있습니다.

(3) 클러스터 수준 기본 채널 유형 구성입니다. 이 예제에서 클러스터에 대한 기본 채널 구현은 **InMemoryChannel**입니다.

4 네임스페이스 범위의 기본 채널 유형 구성입니다. 이 예제에서 **my-namespace** 네임스페이스의 기본 채널 구현은 **KafkaChannel**입니다.

중요

네임스페이스별 기본값을 구성하면 클러스터 수준 설정을 덮어씁니다.

6.5. 채널의 보안 구성

6.5.1. Apache Kafka의 Knative 채널에 대한 TLS 인증 구성

TLS(Transport Layer Security)는 Apache Kafka 클라이언트와 서버에서 Knative와 Kafka 간의 트래픽을 암호화하고 인증을 위해 사용됩니다. TLS는 Apache Kafka에 대한 Knative 브로커 구현에 지원되는 유일한 트래픽 암호화 방법입니다.

사전 요구 사항

- OpenShift Container Platform에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- OpenShift Serverless Operator, Knative Eventing, **KnativeKafka** CR이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Kafka 클러스터 CA 인증서가 **.pem** 파일로 저장되어 있습니다.
- Kafka 클러스터 클라이언트 인증서와 **.pem** 파일로 저장된 키가 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. 선택한 네임스페이스에서 인증서 파일을 시크릿으로 생성합니다.

```
$ 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
```

중요

키 이름 **ca.crt**,**user.crt**, **user.key**를 사용합니다. 이 값은 변경하지 마십시오.

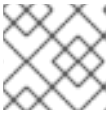
2. **KnativeKafka** 사용자 정의 리소스 편집을 시작합니다.

```
$ oc edit knativekafka
```

3. 시크릿 및 시크릿의 네임스페이스를 참조합니다.

```
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
```



참고

부트스트랩 서버에서 일치하는 포트를 지정해야 합니다.

예를 들면 다음과 같습니다.

```
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. Apache Kafka의 Knative 채널에 대한 SASL 인증 구성

SASL(Simple Authentication and Security Layer)은 Apache Kafka에서 인증에 사용됩니다. 클러스터에서 SASL 인증을 사용하는 경우 Kafka 클러스터와 통신하기 위해 Knative에 인증 정보를 제공해야 합니다. 그렇지 않으면 이벤트를 생성하거나 사용할 수 없습니다.

사전 요구 사항

- OpenShift Container Platform에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- OpenShift Serverless Operator, Knative Eventing, **KnativeKafka** CR이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Kafka 클러스터의 사용자 이름과 암호가 있습니다.
- **PLAIN**,**SCRAM-SHA-256** 또는 **SCRAM-SHA-512** 와 같이 사용할 SASL 메커니즘을 선택했습니다.
- TLS가 활성화된 경우 Kafka 클러스터의 **ca.crt** 인증서 파일도 필요합니다.

- OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. 선택한 네임스페이스에서 인증서 파일을 시크릿으로 생성합니다.

```
$ 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"
```

   - 키 이름 **ca.crt**,**password**, **sasl.mechanism** 을 사용합니다. 이 값은 변경하지 마십시오.
   - 공용 CA 인증서와 함께 SASL을 사용하려면 시크릿을 생성할 때 **ca.crt** 인수 대신 **tls.enabled=true** 플래그를 사용해야 합니다. 예를 들면 다음과 같습니다.

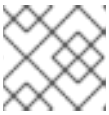
```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
  --from-literal=tls.enabled=true \
  --from-literal=password="SecretPassword" \
  --from-literal=saslType="SCRAM-SHA-512" \
  --from-literal=user="my-sasl-user"
```

2. **KnativeKafka** 사용자 정의 리소스 편집을 시작합니다.

```
$ oc edit knativekafka
```

3. 시크릿 및 시크릿의 네임스페이스를 참조합니다.

```
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
```



참고

부트스트랩 서버에서 일치하는 포트를 지정해야 합니다.

예를 들면 다음과 같습니다.

```
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
```

7장. 서브스크립션

7.1. 서브스크립션 생성

채널과 이벤트 싱크를 생성한 후에는 이벤트 전달을 활성화하는 서브스크립션을 생성할 수 있습니다. 서브스크립션은 이벤트를 전달할 채널과 싱크(인더라고도 함)를 지정하는 **Subscription** 오브젝트를 구성하여 생성됩니다.

7.1.1. 관리자 화면을 사용하여 서브스크립션 생성

구독자 라고도 하는 채널과 이벤트 싱크를 생성한 후 서브스크립션을 생성하여 이벤트 전달을 활성화할 수 있습니다. 서브스크립션은 이벤트를 전달할 채널과 구독자를 지정하는 **Subscription** 오브젝트를 구성하여 생성됩니다. 실패 처리 방법과 같은 구독자별 옵션을 지정할 수도 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 **관리자** 화면에 있습니다.
- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- Knative 채널을 생성했습니다.
- 구독자로 사용할 Knative 서비스를 생성했습니다.

프로세스

1. OpenShift Container Platform 웹 콘솔의 **관리자** 화면에서 **Serverless → Eventing**으로 이동합니다.
2. **채널** 탭에서 서브스크립션을 추가할 채널의 옵션 메뉴  를 선택합니다.
3. 목록에서 **서브스크립션 추가**를 클릭합니다.
4. **서브스크립션 추가** 대화 상자에서 서브스크립션에 대한 **구독자**를 선택합니다. 구독자는 채널에서 이벤트를 수신하는 Knative 서비스입니다.
5. **추가**를 클릭합니다.

7.1.2. 개발자 화면을 사용하여 서브스크립션 생성

채널과 이벤트 싱크를 생성한 후에는 이벤트 전달을 활성화하는 서브스크립션을 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔을 사용하면 간소화되고 직관적인 사용자 인터페이스를 통해 서브스크립션을 생성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Serving, Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.

- *웹 콘솔에 로그인했습니다.*
- *이벤트 싱크(예: Knative 서비스) 및 채널을 생성했습니다.*
- *프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.*

프로세스

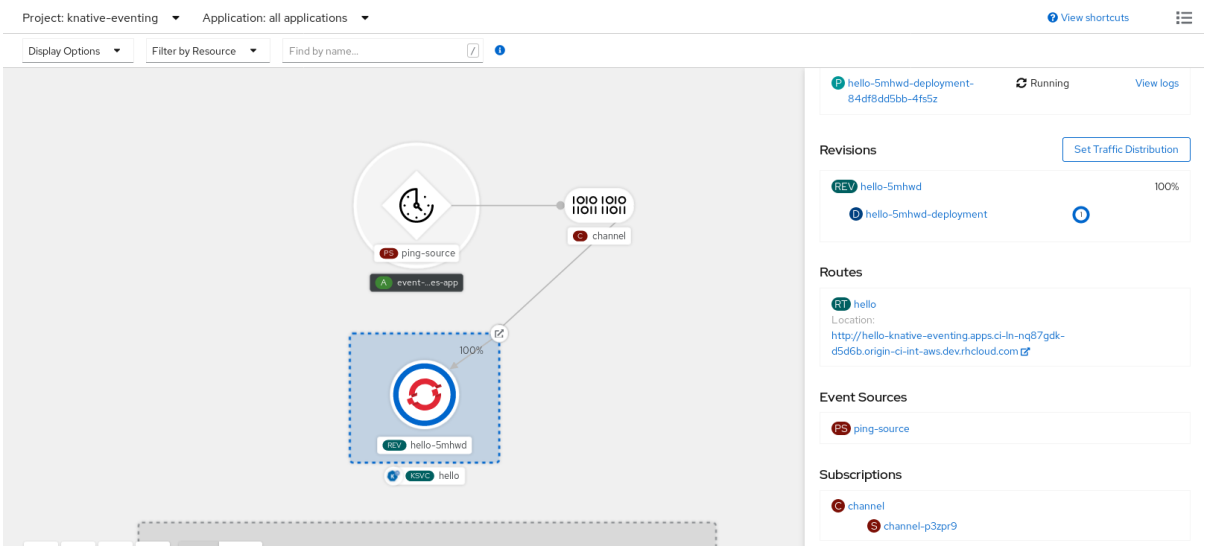
1. ***개발자*** *화면에서* ***토폴로지*** *페이지로 이동합니다.*
2. *다음 방법 중 하나를 사용하여 서브스크립션을 생성합니다.*
   a. *서브스크립션을 생성할 채널 위로 마우스 커서를 이동하고 화살표를 끕니다.* ***서브스크립션 추가*** *옵션이 표시됩니다.*

      i. ***구독자*** *목록에서 싱크를 선택합니다.*
      ii. ***추가****를 클릭합니다.*

   b. *채널과 동일한 네임스페이스 또는 프로젝트의* ***토폴로지*** *보기에서 서비스를 사용할 수 있는 경우, 서브스크립션을 생성할 채널을 클릭하고 화살표를 서비스로 직접 끌어 채널에서 해당 서비스로의 서브스크립션을 즉시 생성합니다.*

검증

- *서브스크립션이 생성되면* ***토폴로지*** *보기에 채널을 서비스에 연결하는 선이 표시되어 이를 확인할 수 있습니다.*



7.1.3. YAML 을 사용하여 서브스크립션 생성

*채널과 이벤트 싱크를 생성한 후에는 이벤트 전달을 활성화하는 서브스크립션을 생성할 수 있습니다. YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 API를 사용하므로 선언적 및 재현 가능한 방식으로 서브스크립션을 설명할 수 있습니다. YAML 을 사용하여 서브스크립션을 생성하려면 **Subscription** 오브젝트를 정의하는 YAML 파일을 생성한 다음 **oc apply** 명령을 사용하여 적용해야 합니다.*

사전 요구 사항

- *OpenShift Serverless Operator 및 Knative Eventing 이 클러스터에 설치되어 있습니다.*
- *OpenShift CLI(**oc**)를 설치합니다.*
- *프로젝트를 생성했거나 OpenShift Container Platform 에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.*

프로세스

- ***Subscription** 오브젝트를 생성합니다.*
  - *YAML 파일을 생성하고 다음 샘플 코드를 여기에 복사합니다.*

```
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. 서브스크립션 이름입니다.
2. 서브스크립션이 연결되는 채널의 구성 설정입니다.
3. 이벤트 전달을 위한 구성 설정입니다. 이 설정은 구독자에게 전달할 수 없는 이벤트에 어떤 일이 발생하는지 스브스크립션에 알립니다. 이 값을 구성하면 사용할 수 없는 이벤트가 **deadLetterSink**로 전송됩니다. 이벤트가 삭제되고 이벤트 재전송이 시도되지 않으며 시스템에 오류가 기록됩니다. **deadLetterSink** 값은 대상이어야 합니다.
4. 구독자에 대한 구성 설정입니다. 채널에서 이벤트가 전달되는 이벤트 싱크입니다.

- YAML 파일을 적용합니다.

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

7.1.4. Knative CLI를 사용하여 서브스크립션 생성

채널과 이벤트 싱크를 생성한 후에는 이벤트 전달을 활성화하는 서브스크립션을 생성할 수 있습니다. Knative(**kn**) CLI를 사용하여 서브스크립션을 생성하면 YAML 파일을 직접 수정하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다. **kn subscription create** 명령을 적절한 플래그와 함께 사용하여 서브스크립션을 생성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

- 싱크를 채널에 연결하는 서브스크립션을 생성합니다.

```
$ 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**은 처리해야 하는 클라우드 이벤트의 소스를 지정합니다. 채널 이름을 제공해야 합니다. 채널 사용자 정의 리소스에서 지원하는 기본 **InMemoryChannel** 채널을 사용하지 않는 경우 **Channel** 이름 앞에 지정된 채널 유형에 대해 **<group:version:kind>**를 추가해야 합니다. 예를 들어 Apache Kafka 백업 채널의 경우 **messaging.knative.dev:v1beta1:KafkaChannel** 이 됩니다.
2. **--sink**는 이벤트를 전달해야 하는 대상 대상을 지정합니다. 기본적으로 **<sink_name>**은 서브스크립션과 동일한 네임스페이스에서 이 이름의 Knative 서비스로 해석됩니다. 다음 접두

사 중 하나를 사용하여 싱크 유형을 지정할 수 있습니다.

ksvc
Knative 서비스입니다.
channel
대상으로 사용해야 하는 채널입니다. 여기에서는 기본 채널 유형만 참조할 수 있습니다.
broker
Eventing 브로커입니다.

3 선택 사항: **--sink-dead-letter**는 이벤트를 전달하지 못하는 경우 이벤트를 전송해야 하는 싱크를 지정하는 데 사용할 선택적 플래그입니다. 자세한 내용은 OpenShift Serverless Event 제공 설명서를 참조하십시오.

명령 예

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

출력 예

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

검증

- 채널이 서브스크립션을 통해 이벤트 싱크 또는 구독자에 연결되어 있는지 확인하려면 기존 서브스크립션을 나열하고 출력을 검사합니다.

```
$ kn subscription list
```

출력 예

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

서브스크립션 삭제

- 서브스크립션을 삭제합니다.

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

7.1.5. 다음 단계

- 이벤트를 이벤트 싱크로 전달하지 못하는 경우 적용되는 이벤트 전달 매개변수를 구성합니다.

7.2. 서브스크립션 관리

7.2.1. Knative CLI를 사용하여 서브스크립션 설명

kn subscription describe 명령을 사용하여 Knative(**kn**) CLI를 사용하여 터미널에서 서브스크립션에 대한 정보를 출력할 수 있습니다. Knative CLI를 사용하여 서브스크립션을 설명하면 YAML 파일을 직접 보는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다.

사전 요구 사항

- *Knative(**kn**) CLI가 설치되어 있습니다.*
- *클러스터에 서브스크립션이 생성되어 있습니다.*

프로세스

- *서브스크립션을 설명합니다.*

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

출력 예

```
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. Knative CLI를 사용하여 서브스크립션 나열

***kn subscription list** 명령을 사용하여 Knative(**kn**) CLI를 사용하여 클러스터의 기존 서브스크립션을 나열할 수 있습니다. Knative CLI를 사용하여 서브스크립션을 나열하면 간소화되고 직관적인 사용자 인터페이스가 제공됩니다.*

사전 요구 사항

- *Knative(**kn**) CLI가 설치되어 있습니다.*

프로세스

- *클러스터의 서브스크립션을 나열합니다.*

```
$ kn subscription list
```

출력 예

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

7.2.3. Knative CLI를 사용하여 서브스크립션 업데이트

kn subscription update 명령과 적절한 플래그를 사용하여 Knative(**kn**) CLI를 사용하여 터미널에서 서브스크립션을 업데이트할 수 있습니다. Knative CLI를 사용하여 서브스크립션을 업데이트하면 YAML 파일을 직접 업데이트하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다.

사전 요구 사항

- Knative(**kn**) CLI가 설치되어 있습니다.
- 서브스크립션이 생성되어 있습니다.

프로세스

- 서브스크립션을 업데이트합니다.

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

1 **--sink**는 이벤트를 전달해야 하는 업데이트된 대상을 지정합니다. 다음 접두사 중 하나를 사용하여 싱크 유형을 지정할 수 있습니다.

ksvc
: Knative 서비스입니다.

channel
: 대상으로 사용해야 하는 채널입니다. 여기에서는 기본 채널 유형만 참조할 수 있습니다.

broker
: Eventing 브로커입니다.

2 선택 사항: **--sink-dead-letter**는 이벤트를 전달하지 못하는 경우 이벤트를 전송해야 하는 싱크를 지정하는 데 사용할 선택적 플래그입니다. 자세한 내용은 OpenShift Serverless Event 제공 설명서를 참조하십시오.

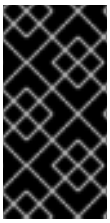
명령 예

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

8장. 이벤트 전달

이벤트를 이벤트 싱크로 전달하지 못하는 경우 적용되는 이벤트 전달 매개변수를 구성할 수 있습니다. 다양한 채널 및 브로커 유형에는 이벤트 전달을 위해 따르는 자체 동작 패턴이 있습니다.

dead letter sink를 포함한 이벤트 전달 매개변수를 구성하면 이벤트 싱크로 전달되지 않는 모든 이벤트를 재시도할 수 있습니다. 그러지 않으면 전달되지 않은 이벤트가 삭제됩니다.



중요

이벤트가 Apache Kafka의 채널 또는 브로커 수신자에게 성공적으로 전달되면 수신자는 **202** 상태 코드로 응답합니다. 즉, 이벤트가 Kafka 주제 내에 안전하게 저장되어 손실되지 않습니다. 수신자가 다른 상태 코드로 응답하는 경우 이벤트는 안전하게 저장되지 않으며 사용자가 문제를 해결하기 위해 단계를 수행해야 합니다.

8.1. 구성 가능한 이벤트 전달 매개변수

이벤트 전달을 위해 다음 매개변수를 구성할 수 있습니다.

dead letter sink

이벤트가 전달되지 않으면 지정된 이벤트 싱크에 저장되도록 **deadLetterSink** 전달 매개변수를 구성할 수 있습니다. dead letter sink에 저장되지 않는 전달되지 않은 이벤트는 삭제됩니다. dead letter sink는 Knative 서비스, Kubernetes 서비스 또는 URI와 같은 Knative Eventing 싱크 계약을 준수하는 모든 주소 지정 가능한 오브젝트입니다.

retries

재시도 전달 매개변수를 정수 값으로 구성하여 이벤트가 dead letter sink로 전송되기 전에 전달을 **retry**해야 하는 최소 횟수를 설정할 수 있습니다.

back off delay

backoffDelay 전달 매개변수를 설정하여 실패 후 이벤트 전달을 재시도하기 전에 시간 지연을 지정할 수 있습니다. **backoffDelay** 매개변수의 기간은 ISO 8601 형식을 사용하여 지정합니다. 예를 들어 **PT1S**는 1초 지연을 지정합니다.

back off policy

backoffPolicy 전달 매개 변수를 사용하여 재시도 정책을 지정할 수 있습니다. 정책을 **linear** 또는 **exponential**로 지정할 수 있습니다. **선형** 백오프 정책을 사용하는 경우 백오프 지연은 **backoffDelay * <numberOfRetries>와 동일합니다**. **exponential** 백오프 정책을 사용하는 경우 백오프 지연은 **backoffDelay*2^<numberOfRetries>**와 동일합니다.

8.2. 이벤트 전달 매개변수 구성 예

Broker, **Trigger**, **Channel**, **Subscription** 오브젝트에 대한 이벤트 전달 매개변수를 구성할 수 있습니다. 브로커 또는 채널에 대한 이벤트 전달 매개변수를 구성하는 경우 이러한 매개변수는 해당 오브젝트에 대해 생성된 트리거 또는 서브스크립션으로 전파됩니다. 트리거 또는 서브스크립션에 대한 이벤트 전달 매개변수를 설정하여 브로커 또는 채널 설정을 덮어쓸 수도 있습니다.

Broker 오브젝트의 예

```
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* 오브젝트의 예**

```
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* 오브젝트의 예**

```
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* 개체 예**

```
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. 트리거에 대한 이벤트 전달 순서 구성

Kafka 브로커를 사용하는 경우 트리거에서 이벤트 싱크로 이벤트 전달 순서를 구성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing, Knative Kafka가 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Kafka 브로커는 클러스터에서 사용할 수 있으며 Kafka 브로커를 생성했습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- OpenShift(**oc**) CLI가 설치되어 있습니다.

프로세스

1. ***Trigger*** 오브젝트를 생성하거나 수정하고 ***kafka.eventing.knative.dev/delivery.order*** 주석을 설정합니다.

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

지원되는 소비자 제공 보장은 다음과 같습니다.

순서가 지정되지 않음

순서가 지정되지 않은 소비자는 적절한 오프셋 관리를 유지하면서 정렬되지 않은 메시지를 전달하는 비차단 소비자입니다.

주문됨

순서가 지정된 소비자는 파티션의 다음 메시지를 전달하기 전에 CloudEvent 구독자의 성공적인 응답을 기다리는 파티션별 차단 소비자입니다.
기본 주문 보증은 ***순서가 지정되지 않습니다***.

2. ***Trigger*** *오브젝트를 적용합니다.*

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

9장. 이벤트 검색

9.1. 이벤트 소스 및 이벤트 소스 유형 나열

OpenShift Container Platform 클러스터에서 존재하거나 사용할 수 있는 모든 이벤트 소스 또는 이벤트 소스 유형의 목록을 볼 수 있습니다. OpenShift Container Platform 웹 콘솔의 Knative(**kn**) CLI 또는 **개발자** 화면을 사용하여 사용 가능한 이벤트 소스 또는 이벤트 소스 유형을 나열할 수 있습니다.

9.2. 명령줄에서 이벤트 소스 유형 나열

Knative(**kn**) CLI를 사용하면 클러스터에서 사용 가능한 이벤트 소스 유형을 볼 수 있는 간소화되고 직관적인 사용자 인터페이스가 제공됩니다.

9.2.1. Knative CLI를 사용하여 사용 가능한 이벤트 소스 유형 나열

kn source list-types CLI 명령을 사용하여 클러스터에서 생성하고 사용할 수 있는 이벤트 소스 유형을 나열할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

프로세스

1. 터미널에서 사용 가능한 이벤트 소스 유형을 나열합니다.

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

출력 예

```
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. 선택 사항: OpenShift Container Platform에서 사용 가능한 이벤트 소스 유형을 YAML 형식으로 나열할 수도 있습니다.

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

9.3. 개발자 관점에서 이벤트 소스 유형 나열

클러스터에서 사용 가능한 모든 이벤트 소스 유형 목록을 볼 수 있습니다. OpenShift Container Platform 웹 콘솔을 사용하면 사용 가능한 이벤트 소스 유형을 볼 수 있는 간소화되고 직관적인 사용자 인터페이스를 사용할 수 있습니다.

9.3.1. 개발자 화면 내에서 사용 가능한 이벤트 소스 유형 보기

사전 요구 사항

- OpenShift Container Platform 웹 콘솔에 로그인했습니다.
- OpenShift Serverless Operator 및 Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

1. **개발자** 화면에 액세스합니다.
2. **+추가**를 클릭합니다.
3. **이벤트 소스**를 클릭합니다.
4. 사용 가능한 이벤트 소스 유형을 확인합니다.

9.4. 명령줄에서 이벤트 소스 나열

Knative(**kn**) CLI를 사용하면 클러스터에서 기존 이벤트 소스를 볼 수 있는 간소화되고 직관적인 사용자 인터페이스가 제공됩니다.

9.4.1. Knative CLI를 사용하여 사용 가능한 이벤트 소스 나열

kn source list 명령을 사용하여 기존 이벤트 소스를 나열할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

프로세스

1. 터미널에서 기존 이벤트 소스를 나열합니다.

```
$ kn source list
```

출력 예

```
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. 선택 사항: **--type** 플래그를 사용하여 특정 유형의 이벤트 소스만 나열할 수 있습니다.

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

명령 예

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

출력 예

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

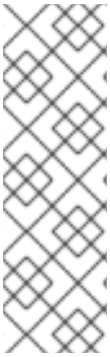
10장. 이벤트링 구성 튜닝

10.1. KNATIVE EVENTING 시스템 배포 구성 덮어쓰기

KnativeEventing CR(사용자 정의 리소스)의 **workloads** 사양을 수정하여 일부 특정 배포의 기본 구성을 덮어쓸 수 있습니다. 현재 **eventing-controller**,**eventing-webhook**, **imc-controller** 필드 및 프로브의 **준비** 및 **활성** 필드에 대해 기본 구성 설정을 재정의할 수 있습니다.

중요

replicas 사양은 HPA(horizontal Pod Autoscaler)를 사용하는 배포의 복제본 수를 재정의할 수 없으며 **eventing-webhook** 배포에는 작동하지 않습니다.



참고

기본적으로 배포에 정의된 프로브만 덮어쓸 수 있습니다.

모든 Knative Serving 배포는 기본적으로 준비 상태 및 활성 상태 프로브를 정의합니다.

- **net-kourier-controller** 및 **3scale-kourier-gateway** 는 준비 상태 프로브만 정의합니다.
- **net-istio-controller** 및 **net-istio-webhook** 는 프로브를 정의하지 않습니다.

10.1.1. 배포 구성 덮어쓰기

현재 **eventing-controller**,**eventing-webhook**, **imc-controller** 필드 및 프로브의 **준비** 및 **활성** 필드에 대해 기본 구성 설정을 재정의할 수 있습니다.

중요

replicas 사양은 HPA(horizontal Pod Autoscaler)를 사용하는 배포의 복제본 수를 재정의할 수 없으며 **eventing-webhook** 배포에는 작동하지 않습니다.

다음 예에서 **KnativeEventing** CR은 다음과 같이 **eventing-controller** 배포를 덮어씁니다.

- **준비 상태** 프로브 시간 초과 **eventing-controller** 가 10초로 설정됩니다.
- 배포에 CPU 및 메모리 리소스 제한이 지정되어 있습니다.
- 배포에는 3개의 복제본이 있습니다.
- **example-label:** 레이블이 추가되었습니다.
- **example-annotation: 주석** 주석이 추가되었습니다.
- **nodeSelector** 필드는 **disktype: hdd** 라벨이 있는 노드를 선택하도록 설정됩니다.

KnativeEventing CR 예

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

```
  name: knative-eventing
  namespace: knative-eventing
spec:
  workloads:
  - 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 *준비* 및 활성 상태 프로브 덮어쓰기를 사용하여 **프로브** 처리기와 관련된 필드를 제외하고 Kubernetes API에 지정된 대로 배포 컨테이너에서 프로브의 모든 필드를 덮어쓸 수 있습니다. **exec**,**grpc**,**httpGet**, **tcpSocket**.

참고

KnativeEventing CR 레이블 및 주석 설정은 배포 자체와 결과 Pod 모두에 대한 배포 레이블 및 주석을 재정의합니다.

10.1.2. 소비자 그룹 ID 및 주제 이름 수정

트리거, 브로커 및 채널에서 사용하는 소비자 그룹 ID 및 주제 이름을 생성하기 위한 템플릿을 변경할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- OpenShift Serverless Operator, Knative Eventing 및 **KnativeKafka** CR(사용자 정의 리소스)이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성하거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성할 수 있는 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.

프로세스

1. 트리거, 브로커 및 채널에서 사용하는 소비자 그룹 ID 및 주제 이름을 생성하기 위한 템플릿을 변경하려면 **KnativeKafka** 리소스를 수정합니다.

```
apiVersion: v1
kind: KnativeKafka
metadata:
  name: knative-kafka
  namespace: knative-eventing
# ...
spec:
  config:
    config-kafka-features:
      triggers.consumergroup.template: <template> 1
      brokers.topic.template: <template> 2
      channels.topic.template: <template> 3
```

1 *트리거에서 사용하는 소비자 그룹 ID를 생성하기 위한 템플릿입니다. 유효한 Go* ***텍스트/템플릿*** *값을 사용합니다. 기본값은* ***{% raw %}"knative-trigger-{{ .Namespace }}-{{ .Name }}"{% endraw %}*** *입니다.*

2 *브로커가 사용하는 Kafka 주제 이름을 생성하기 위한 템플릿입니다. 유효한 Go* ***텍스트/템플릿*** *값을 사용합니다. 기본값은* ***{% raw %}"knative-broker-{{ .Namespace }}-{{ .Name }}"{% endraw %}*** *입니다.*

3 *채널에서 사용하는 Kafka 주제 이름을 생성하기 위한 템플릿입니다. 유효한 Go* ***텍스트/템플릿*** *값을 사용합니다. 기본값은* ***{% raw %}"messaging-kafka.{{ .Namespace }}.{{ .Name }}"{% endraw %}*** *입니다.*

템플릿 구성 예

```
apiVersion: v1
kind: KnativeKafka
metadata:
  name: knative-kafka
  namespace: knative-eventing
# ...
spec:
  config:
    config-kafka-features:
      triggers.consumergroup.template: "{% raw %}"knative-trigger-{{ .Namespace }}-{{ .Name }}-{{ .annotations.my-annotation }}"{% endraw %}"
      brokers.topic.template: "{% raw %}"knative-broker-{{ .Namespace }}-{{ .Name }}-{{ .annotations.my-annotation }}"{% endraw %}"
      channels.topic.template: "{% raw %}"messaging-kafka.{{ .Namespace }}.{{ .Name }}-{{ .annotations.my-annotation }}"{% endraw %}"
```

2. ***KnativeKafka*** *YAML 파일을 적용합니다.*

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

추가 리소스

- *Kubernetes API 문서의 프로브 구성 섹션*

10.2. 고가용성

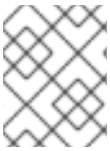
고가용성(HA)은 중단이 발생하는 경우 API가 작동하도록 하는 데 도움이 되는 Kubernetes API의 표준 기능입니다. HA 배포에서 활성 컨트롤러가 충돌하거나 삭제되면 다른 컨트롤러를 쉽게 사용할 수 있습니다. 이 컨트롤러는 현재 사용할 수 없는 컨트롤러에서 서비스 중인 API의 처리를 대신합니다.

OpenShift Serverless의 HA는 Knative Serving 또는 Eventing 컨트롤 플레인을 설치하면 기본적으로 활성화되는 리더 선택을 통해 사용할 수 있습니다. 리더 선택 HA 패턴을 사용하는 경우에는 요구하기 전에 컨트롤러의 인스턴스가 이미 예약되어 클러스터 내에서 실행됩니다. 이러한 컨트롤러 인스턴스는 리더 선택 잠금이라는 공유 리소스를 사용하기 위해 경쟁합니다. 지정된 시간에 리더 선택 잠금 리소스에 액세스할 수 있는 컨트롤러의 인스턴스를 리더라고 합니다.

OpenShift Serverless의 HA는 Knative Serving 또는 Eventing 컨트롤 플레인을 설치하면 기본적으로 활성화되는 리더 선택을 통해 사용할 수 있습니다. 리더 선택 HA 패턴을 사용하는 경우에는 요구하기 전에 컨트롤러의 인스턴스가 이미 예약되어 클러스터 내에서 실행됩니다. 이러한 컨트롤러 인스턴스는 리더 선택 잠금이라는 공유 리소스를 사용하기 위해 경쟁합니다. 지정된 시간에 리더 선택 잠금 리소스에 액세스할 수 있는 컨트롤러의 인스턴스를 리더라고 합니다.

10.2.1. Knative Eventing의 고가용성 복제본 구성

HA(고가용성)는 기본적으로 Knative Eventing ***이벤트ing-controller,eventing- webhook,imc-controller,imc-dispatcher, mt-broker-controller*** *구성 요소에 기본적으로 사용할 수 있으며, 기본적으로 두 개의 복제본을 사용하도록 구성됩니다.* ***KnativeEventing*** *CR(사용자 정의 리소스)의* ***spec.high-availability.replicas*** *값을 수정하여 이러한 구성 요소의 복제본 수를 변경할 수 있습니다.*



참고

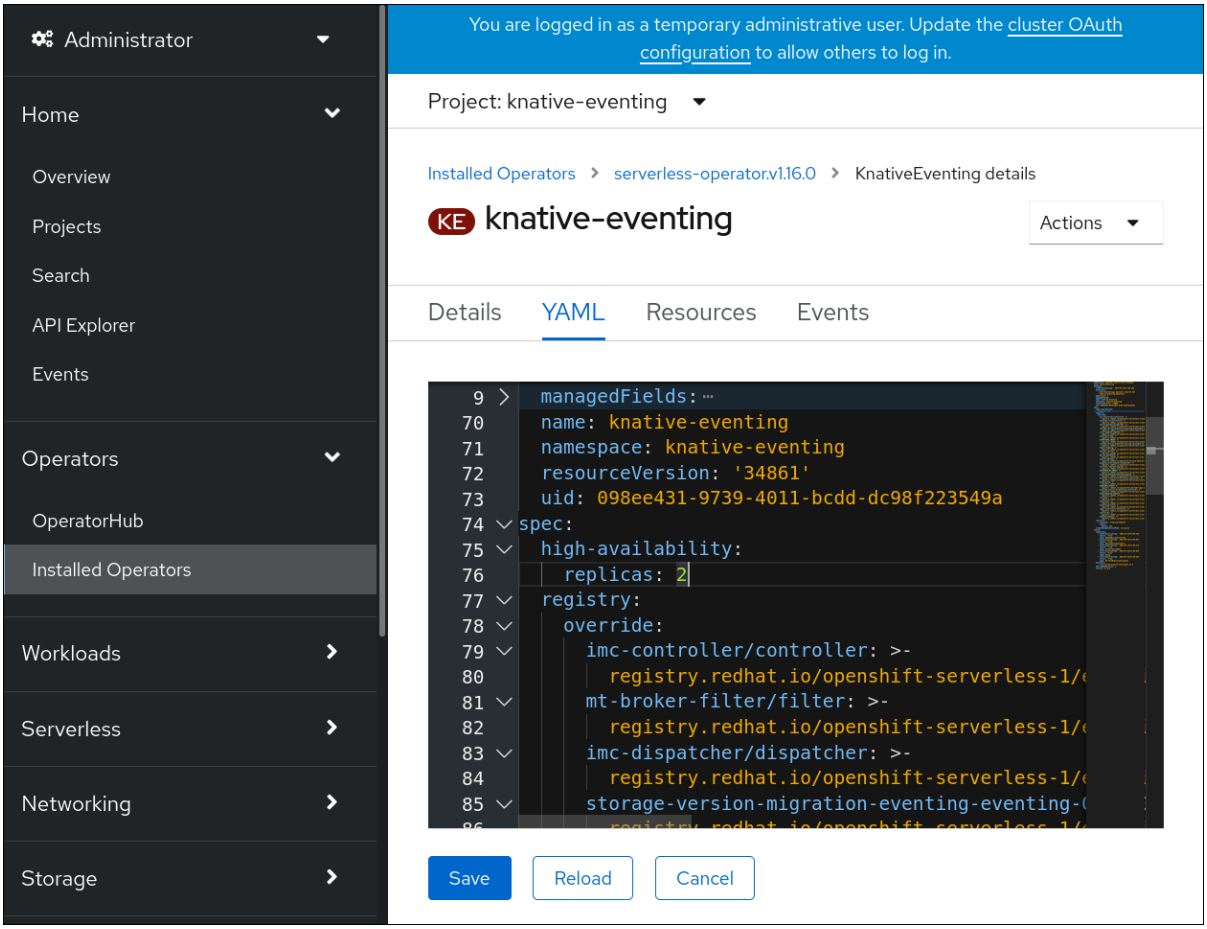
Knative Eventing의 경우 ***mt-broker-filter*** *및* ***mt-broker-ingress*** *배포는 HA에 의해 확장되지 않습니다. 여러 배포가 필요한 경우 이러한 구성 요소를 수동으로 스케일링합니다.*

사전 요구 사항

- *OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 있습니다.*
- *OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.*

프로세스

1. *OpenShift Container Platform 웹 콘솔의* ***관리자*** *화면에서* ***OperatorHub → 설치된 Operator****로 이동합니다.*
2. ***knative-serving*** *네임스페이스를 선택합니다.*
3. *OpenShift Serverless Operator의* ***제공되는 API*** *목록에서* ***Knative Eventing****을 클릭하여* ***Knative Eventing*** *탭으로 이동합니다.*
4. ***knative-eventing****을 클릭한 다음* ***knative-eventing*** *페이지의* ***YAML*** *탭으로 이동합니다.*



5. ***KnativeEventing*** *CR의 복제본 수를 수정합니다.*

   YAML의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  high-availability:
    replicas: 3
```

10.2.2. Apache Kafka의 Knative 브로커 구현을 위한 고가용성 복제본 구성

HA(고가용성)는 기본적으로 Apache Kafka 구성 요소 ***kafka-controller*** *및* ***kafka-webhook-eventing*** *의 Knative 브로커 구현에 사용할 수 있으며 기본적으로 두 개의 복제본이 있도록 구성됩니다.* ***KnativeKafka*** *CR(사용자 정의 리소스)의* ***spec.high-availability.replicas*** *값을 수정하여 이러한 구성 요소의 복제본 수를 변경할 수 있습니다.*

사전 요구 사항

- *OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 있습니다.*
- *Apache Kafka용 OpenShift Serverless Operator 및 Knative 브로커가 클러스터에 설치되어 있습니다.*

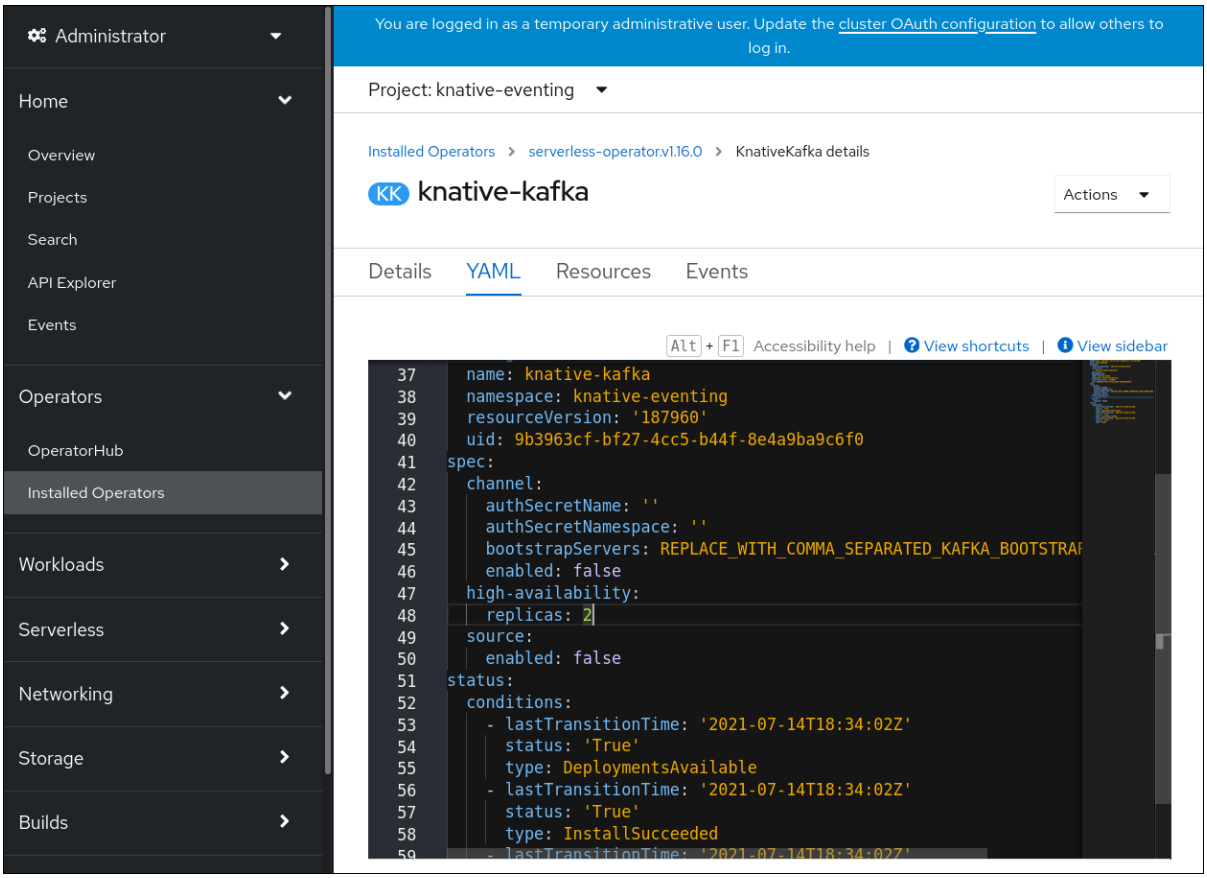
프로세스

1. *OpenShift Container Platform 웹 콘솔의 **관리자** 화면에서 **OperatorHub → 설치된 Operator**로 이동합니다.*

2. ***knative-serving** 네임스페이스를 선택합니다.*

3. *OpenShift Serverless Operator의 **제공되는 API** 목록에서 **Knative Kafka**를 클릭하여 **Knative Kafka** 탭으로 이동합니다.*

4. ***knative-kafka**를 클릭한 다음 **knative-kafka** 페이지의 **YAML** 탭으로 이동합니다.*



5. ***KnativeKafka** CR의 복제본 수를 수정합니다.*

YAML의 예

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  name: knative-kafka
  namespace: knative-eventing
spec:
  high-availability:
    replicas: 3
```

11장. EVENTING에 대한 KUBE-RBAC-PROXY 구성

kube-rbac-proxy 구성 요소는 Knative Eventing에 대한 내부 인증 및 권한 부여 기능을 제공합니다.

11.1. EVENTING에 대한 KUBE-RBAC-PROXY 리소스 구성

OpenShift Serverless Operator CR을 사용하여 **kube-rbac-proxy** 컨테이너의 리소스 할당을 전역적으로 덮어쓸 수 있습니다.

참고

특정 배포에 대한 리소스 할당을 덮어쓸 수도 있습니다.

다음 구성은 Knative Eventing **kube-rbac-proxy** 최소 및 최대 CPU 및 메모리 할당을 설정합니다.

KnativeEventing CR 예

```
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. 최소 CPU 할당을 설정합니다.
2. 최소 RAM 할당을 설정합니다.
3. 최대 CPU 할당을 설정합니다.
4. 최대 RAM 할당을 설정합니다.

12 장. SERVICE MESH 에서 CONTAINERSOURCE 사용

서비스 메시에서 컨테이너 소스를 사용할 수 있습니다.

12.1. 서비스 메시를 사용하여 CONTAINERSOURCE 구성

다음 절차에서는 Service Mesh를 사용하여 컨테이너 소스를 구성하는 방법을 설명합니다.

사전 요구 사항

- Service Mesh 및 Serverless의 통합을 설정했습니다.

프로세스

1. **Service MeshMemberRoll** 의 멤버인 네임스페이스에 서비스를 생성합니다.

   event-display-service.yaml 구성 파일의 예

```
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) **ServiceMeshMemberRoll** 의 멤버인 네임스페이스입니다.

   (2) 이 주석은 Service Mesh 사이드카를 Knative 서비스 Pod에 삽입합니다.

2. **Service** 리소스를 적용합니다.

```
$ oc apply -f event-display-service.yaml
```

3. **ServiceMeshMemberRoll** 및 sink의 멤버인 네임스페이스에 **event-display** 로 설정된 **ContainerSource** 오브젝트를 생성합니다.

   test-heartbeats-containersource.yaml 구성 파일의 예

```
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:
        # This corresponds to a heartbeats image URI that you have built and published
        - image: quay.io/openshift-knative/heartbeats
          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-service
```

1 ***ServiceMeshMemberRoll*** *의 일부인 네임스페이스입니다.*

2 *이러한 주석은 Service Mesh 와* ***ContainerSource*** *오브젝트를 통합할 수 있습니다.*

4. ***ContainerSource*** *리소스를 적용합니다.*

```
$ oc apply -f test-heartbeats-containersource.yaml
```

5. *선택 사항: 메시지 덤퍼 기능 로그를 확인하여 이벤트가 Knative 이벤트 싱크로 전송되었는지 확인합니다.*

명령 예

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

출력 예

```
☁  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": ""
}
```

13장. 서비스 메시에서 싱크 바인딩 사용

Service Mesh와 함께 싱크 바인딩을 사용할 수 있습니다.

13.1. 서비스 메시를 사용하여 싱크 바인딩 구성

다음 절차에서는 Service Mesh를 사용하여 싱크 바인딩을 구성하는 방법을 설명합니다.

사전 요구 사항

- Service Mesh 및 Serverless의 통합을 설정했습니다.

프로세스

1. **Service MeshMemberRoll** 의 멤버인 네임스페이스에 Service 오브젝트를 생성합니다.

   event-display-service.yaml 구성 파일의 예

```
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 **ServiceMeshMemberRoll** 의 멤버인 네임스페이스입니다.

   2 이 주석은 Service Mesh 사이드카를 Knative 서비스 Pod에 삽입합니다.

2. **Service** 오브젝트를 적용합니다.

```
$ oc apply -f event-display-service.yaml
```

3. **SinkBinding** 오브젝트를 생성합니다.

   heartbeat-sinkbinding.yaml 구성 파일의 예

```
apiVersion: sources.knative.dev/v1alpha1
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 ***ServiceMeshMemberRoll*** *의 일부인 네임스페이스입니다.*

2 ***app: heartbeat-cron*** *라벨을 이벤트 싱크에 모든 작업을 바인딩합니다.*

4. ***SinkBinding*** *오브젝트를 적용합니다.*

```
$ oc apply -f heartbeat-sinkbinding.yaml
```

5. ***CronJob*** *오브젝트를 생성합니다.*

heartbeat-cronjob.yaml *구성 파일의 예*

```
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. **ServiceMeshMemberRoll** 의 일부인 네임스페이스입니다.
2. 서비스 메시 사이드카를 **CronJob** Pod에 삽입합니다.

6. **CronJob** 오브젝트를 적용합니다.

```
$ oc apply -f heartbeat-cronjob.yaml
```

7. 선택 사항: 메시지 덤퍼 기능 로그를 확인하여 이벤트가 Knative 이벤트 싱크로 전송되었는지 확인합니다.

명령 예

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

출력 예

```
☁  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": ""
  }
```