Red Hat OpenShift Serverless 1.32

Eventing

OpenShift Serverless でのイベント駆動型アーキテクチャーの使用

Last Updated: 2024-06-03

Red Hat OpenShift Serverless 1.32 Eventing

OpenShift Serverless でのイベント駆動型アーキテクチャーの使用

法律上の通知

概要

このドキュメントは、イベントソースおよびシンク、ブローカー、トリガー、チャネル、サブスクリプションなどの Eventing 機能に関する情報を提供します。

目次

第1章 KNATIVE EVENTING

OpenShift Container Platform 上の Knative Eventing を使用すると、開発者は Serverless アプリケーションと共に イベント駆動型のアーキテクチャー を使用できます。イベント駆動型のアーキテクチャーは、イベントプロデューサーとイベントコンシューマー間の関係を切り離すという概念に基づいています。

イベントプロデューサーはイベントを作成し、イベント**シンク**またはコンシューマーはイベントを受信します。Knative Eventing は、標準の HTTP POST リクエストを使用してイベントプロデューサーとシンク間でイベントを送受信します。これらのイベントは CloudEvents 仕様 に準拠しており、すべてのプログラミング言語でのイベントの作成、解析、および送受信を可能にします。

1.1. KNATIVE EVENTING ユースケース

Knative Eventing は以下のユースケースをサポートします。

コンシューマーを作成せずにイベントを公開する

イベントを HTTP POST としてブローカーに送信し、バインディングを使用してイベントを生成するアプリケーションから宛先設定を分離できます。

パブリッシャーを作成せずにイベントを消費

Trigger を使用して、イベント属性に基づいて Broker からイベントを消費できます。アプリケーションはイベントを HTTP POST として受信します。

複数のタイプのシンクへの配信を有効にするために、Knative Eventing は複数の Kubernetes リソースで実装できる以下の汎用インターフェイスを定義します。

アドレス指定可能なリソース

HTTP 経由でイベントの **status.address.url** フィールドに定義されるアドレスに配信されるイベントを受信し、確認することができます。Kubernetes **Service** リソースはアドレス指定可能なインターフェイスにも対応します。

呼び出し可能なリソース

HTTP 経由で配信されるイベントを受信し、これを変換できます。HTTP 応答ペイロードで **0** または **1** の新規イベントを返します。返されるイベントは、外部イベントソースからのイベントが処理されるのと同じ方法で処理できます。

第2章 イベントソース

2.1. イベントソース

Knative **イベントソース** には、クラウドイベントの生成またはインポート、これらのイベントの別のエンドポイントへのリレー (sink とも呼ばれる) を行う Kubernetes オブジェクトを指定できます。イベントに対応する分散システムを開発するには、イベントのソースが重要になります。

OpenShift Container Platform Web コンソールの **Developer** パースペクティブ、Knative (**kn**) CLI を使用するか、YAML ファイルを適用することで、Knative イベントソースを作成および管理できます。

現時点で、OpenShift Serverless は以下のイベントソースタイプをサポートします。

API サーバーソース

Kubernetes API サーバーイベントを Knative に送ります。API サーバーソースは、Kubernetes リソースが作成、更新、または削除されるたびに新規イベントを送信します。

Ping ソース

指定された cron スケジュールに、固定ペイロードを使用してイベントを生成します。

Kafka イベントソース

Apache Kafka クラスターをイベントソースとしてシンクに接続します。

カスタムイベントソース を作成することもできます。

2.2. ADMINISTRATOR パースペクティブのイベントソース

イベントに対応する分散システムを開発するには、イベントのソースが重要になります。

2.2.1. Administrator パースペクティブを使用したイベントソースの作成

Knative **イベントソース** には、クラウドイベントの生成またはインポート、これらのイベントの別のエンドポイントへのリレー (**sink** とも呼ばれる) を行う Kubernetes オブジェクトを指定できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Web コンソールにログインしており、**Administrator** パースペクティブを使用している。
- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Serverless → Eventing** に移動します。
2. **Create** リストで、**Event Source** を選択します。**Event Sources** ページに移動します。
3. 作成するイベントソースタイプを選択します。

2.3. API サーバーソースの作成

API サーバーソースは、Knative サービスなどのイベントシンクを Kubernetes API サーバーに接続するために使用できるイベントソースです。API サーバーソースは Kubernetes イベントを監視し、それらを Knative Eventing ブローカーに転送します。

2.3.1. Web コンソールを使用した API サーバーソースの作成

Knative Eventing がクラスターにインストールされると、Web コンソールを使用して API サーバーソースを作成できます。OpenShift Container Platform Web コンソールを使用すると、イベントソースを作成するための合理的で直感的なユーザーインターフェイスが提供されます。

前提条件

- OpenShift Container Platform Web コンソールにログインしている。
- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

既存のサービスアカウントを再利用する必要がある場合は、既存の **ServiceAccount** リソースを変更して、新規リソースを作成せずに、必要なパーミッションを含めることができます。

1. イベントソースのサービスアカウント、ロールおよびロールバインディングを YAML ファイルとして作成します。

```
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 この namespace を、イベントソースのインストールに選択した namespace に変更します。

2. YAML ファイルを適用します。

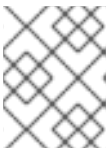
```
$ oc apply -f <filename>
```

3. **Developer** パースペクティブで、**+Add → Event Source** に移動します。**Event Sources** ページが表示されます。

4. オプション: イベントソースに複数のプロバイダーがある場合は、**Providers** 一覧から必要なプロバイダーを選択し、プロバイダーから利用可能なイベントソースをフィルターします。

5. **ApiServerSource** を選択してから **Create Event Source**をクリックします。**Create Event Source** ページが表示されます。

6. **Form view** または **YAML view** を使用して、**ApiServerSource** 設定を設定します。



注記

Form view と **YAML view** 間で切り換えることができます。ビューの切り替え時に、データは永続化されます。

   a. **APIVERSION** に **v1** を、**KIND** に **Event** を入力します。

   b. 作成したサービスアカウントの **Service Account Name**を選択します。

   c. **Target** セクションで、イベントシンクを選択します。これは **Resource** または **URI** のいずれかです。

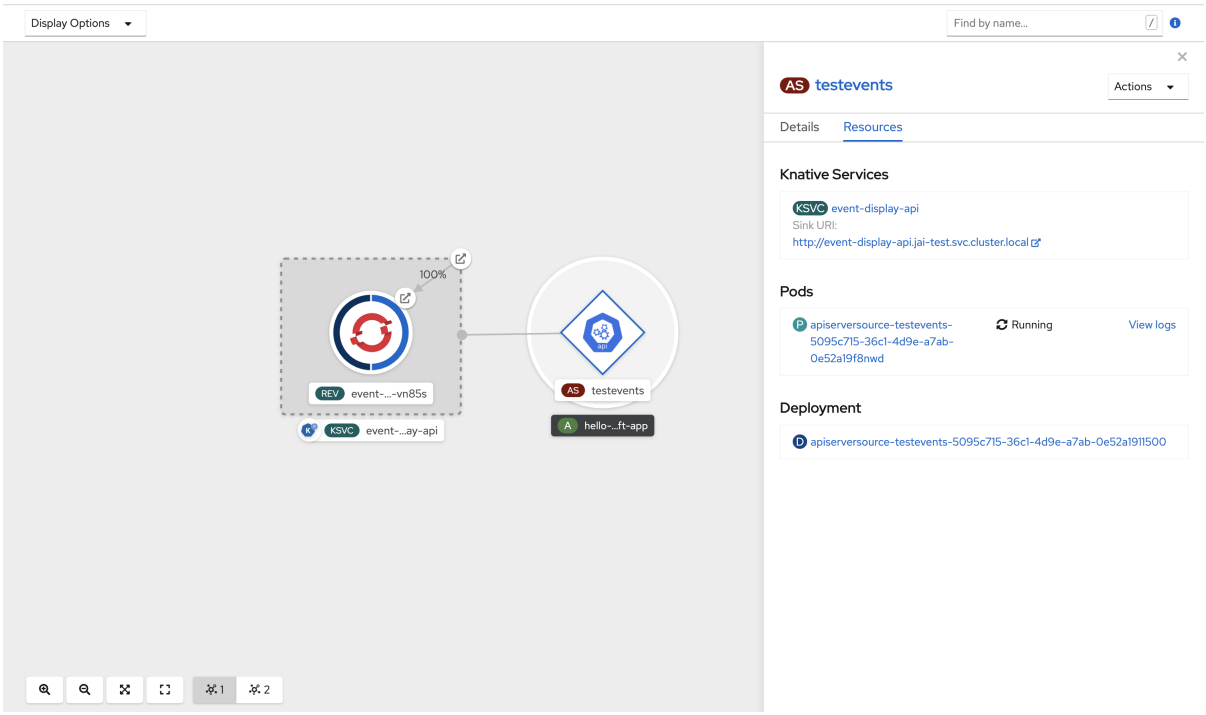
      i. **Resource** を選択して、チャネル、ブローカー、またはサービスをイベントソースのシンクとして使用します。

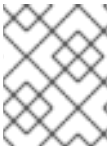
      ii. **URI** を選択して、イベントのルーティング先となる URI (Uniform Resource Identifier) を指定します。

7. **Create** をクリックします。

検証

- API サーバーソースを作成したら、それを **トポロジー** ビューで表示して、イベントシンクに接続されていることを確認します。



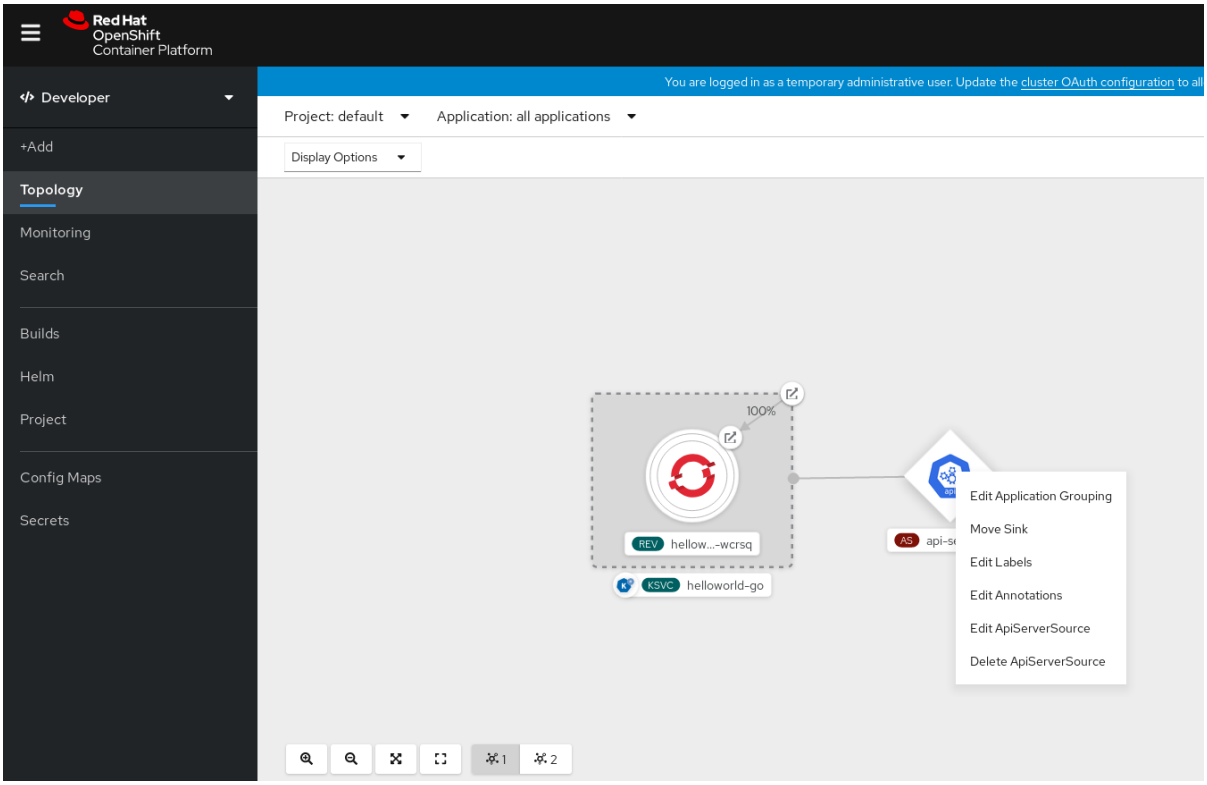


注記

URI シンクが使用される場合は、**URI sink → Edit URI** を右クリックして URI を変更します。

API サーバーソースの削除

1. **Topology** ビューに移動します。
2. API サーバーソースを右クリックし、**Delete ApiServerSource** を選択します。



2.3.2. Knative CLI を使用した API サーバーソースの作成

kn source apiserver create コマンドを使用し、**kn** CLI を使用して API サーバーソースを作成できます。API サーバーソースを作成するために **kn** CLI を使用すると、YAML ファイルを直接修正するよりも合理的で直感的なユーザーインターフェイスが得られます。

前提条件

- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- Knative (**kn**) CLI がインストールされている。

手順

既存のサービスアカウントを再利用する必要がある場合は、既存の **ServiceAccount** リソースを変更して、新規リソースを作成せずに、必要なパーミッションを含めることができます。

1. イベントソースのサービスアカウント、ロールおよびロールバインディングを YAML ファイルとして作成します。

```
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) この namespace を、イベントソースのインストールに選択した namespace に変更します。

2. YAML ファイルを適用します。

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

3. イベントシンクを持つ API サーバーソースを作成します。次の例では、シンクはブローカーです。

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

4. API サーバーソースが正しく設定されていることを確認するには、受信メッセージをログにダンプする Knative サービスを作成します。

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

5. ブローカーをイベントシンクとして使用した場合は、トリガーを作成して、**default** のブローカーからサービスへのイベントをフィルタリングします。

-

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

6. デフォルト namespace で Pod を起動してイベントを作成します。

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

7. 以下のコマンドを入力し、生成される出力を検査して、コントローラーが正しくマップされていることを確認します。

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

出力例

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

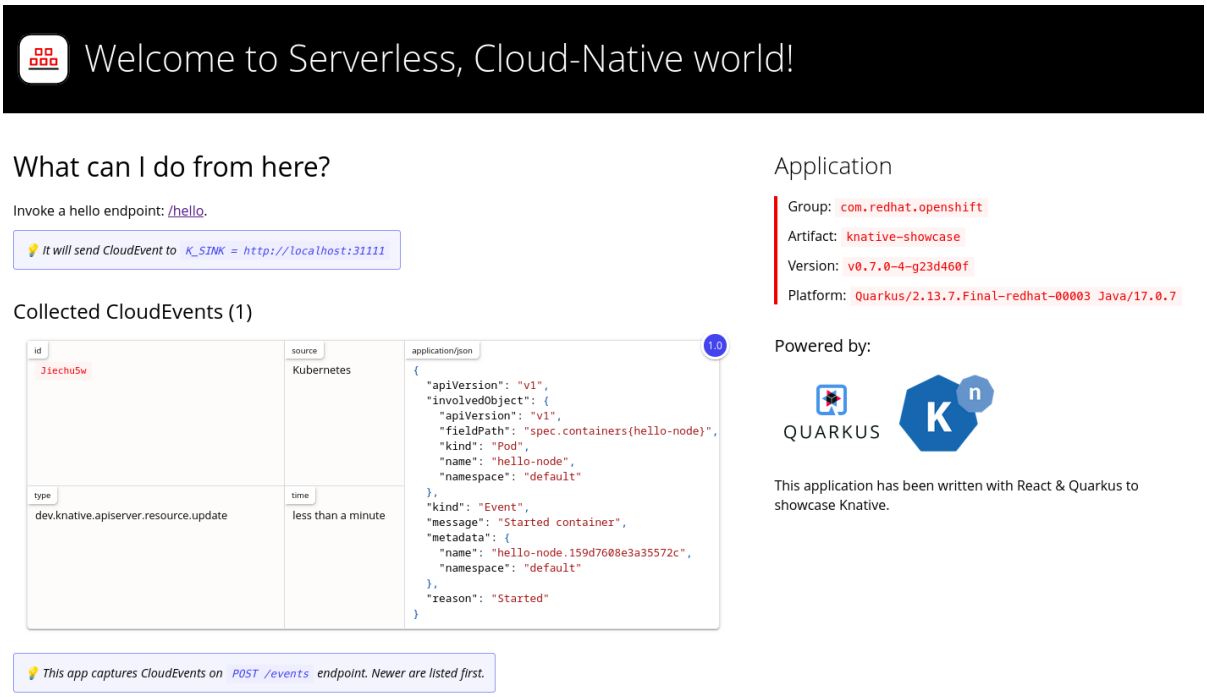
検証

Kubernetes イベントが Knative に送信されたことを確認するには、イベント表示ログを確認するか、Web ブラウザーを使用してイベントを確認します。

- Web ブラウザーでイベントを表示するには、次のコマンドで返されたリンクを開きます。

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

図2.1 ブラウザーページの例



- あるいは、ターミナルでログを確認するには、次のコマンドを入力して Pod のイベント表示ログを表示します。

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

出力例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.apiserver.resource.update
  datacontenttype: application/json
  ...
Data,
  {
    "apiVersion": "v1",
    "involvedObject": {
      "apiVersion": "v1",
      "fieldPath": "spec.containers{event-origin}",
      "kind": "Pod",
      "name": "event-origin",
      "namespace": "default",
       .....
    },
    "kind": "Event",
    "message": "Started container",
    "metadata": {
      "name": "event-origin.159d7608e3a3572c",
      "namespace": "default",
      ....
    },
```

```
    "reason": "Started",
    ...
  }
```

API サーバーソースの削除

1. トリガーを削除します。

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

2. イベントソースを削除します。

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

3. サービスアカウント、クラスターロール、およびクラスターバインディングを削除します。

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

2.3.2.1. Knative CLI シンクフラグ

Knative (**kn**) CLI を使用してイベントソースを作成する場合は、**--sink** フラグを使用して、そのリソースからイベントが送信されるシンクを指定できます。シンクは、他のリソースから受信イベントを受信できる、アドレス指定可能または呼び出し可能な任意のリソースです。

以下の例では、サービスの **http://event-display.svc.cluster.local** をシンクとして使用するシンクバインディングを作成します。

シンクフラグを使用したコマンドの例

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

1 **http://event-display.svc.cluster.local** の **svc** は、シンクが Knative サービスであることを判別します。他のデフォルトのシンクの接頭辞には、**channel** および **broker** が含まれます。

2.3.3. YAML ファイルを使用した API サーバーソースの作成

YAML ファイルを使用して Knative リソースを作成する場合は、宣言的 API を使用するため、再現性の高い方法でイベントソースを宣言的に記述できます。YAML を使用して API サーバーソースを作成するには、**ApiServerSource** オブジェクトを定義する YAML ファイルを作成し、**oc apply** コマンドを使用してそれを適用する必要があります。

前提条件

- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。

- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- API サーバーソース YAML ファイルで定義されるものと同じ namespace に **default** ブローカーを作成している。
- OpenShift CLI (**oc**) がインストールされている。

手順

既存のサービスアカウントを再利用する必要がある場合は、既存の **ServiceAccount** リソースを変更して、新規リソースを作成せずに、必要なパーミッションを含めることができます。

1. イベントソースのサービスアカウント、ロールおよびロールバインディングを YAML ファイルとして作成します。

```
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 この namespace を、イベントソースのインストールに選択した namespace に変更します。

2. YAML ファイルを適用します。

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

3. API サーバーソースを YAML ファイルとして作成します。

```
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 ファイルを適用します。

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

5. API サーバーソースが正しく設定されていることを確認するには、受信メッセージをログにダンプする Knative サービスを YAML ファイルとして作成します。

```
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 ファイルを適用します。

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

7. 直接の手順で作成下サービスに、**default** ブローカーからイベントをフィルターする **Trigger** オブジェクトを YAML ファイルとして作成します。

```
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 ファイルを適用します。

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

9. デフォルト namespace で Pod を起動してイベントを作成します。

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

10. 以下のコマンドを入力し、出力を検査して、コントローラーが正しくマップされていることを確認します。

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

出力例

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

```
    apiVersion: eventing.knative.dev/v1
    kind: Broker
    name: default
```

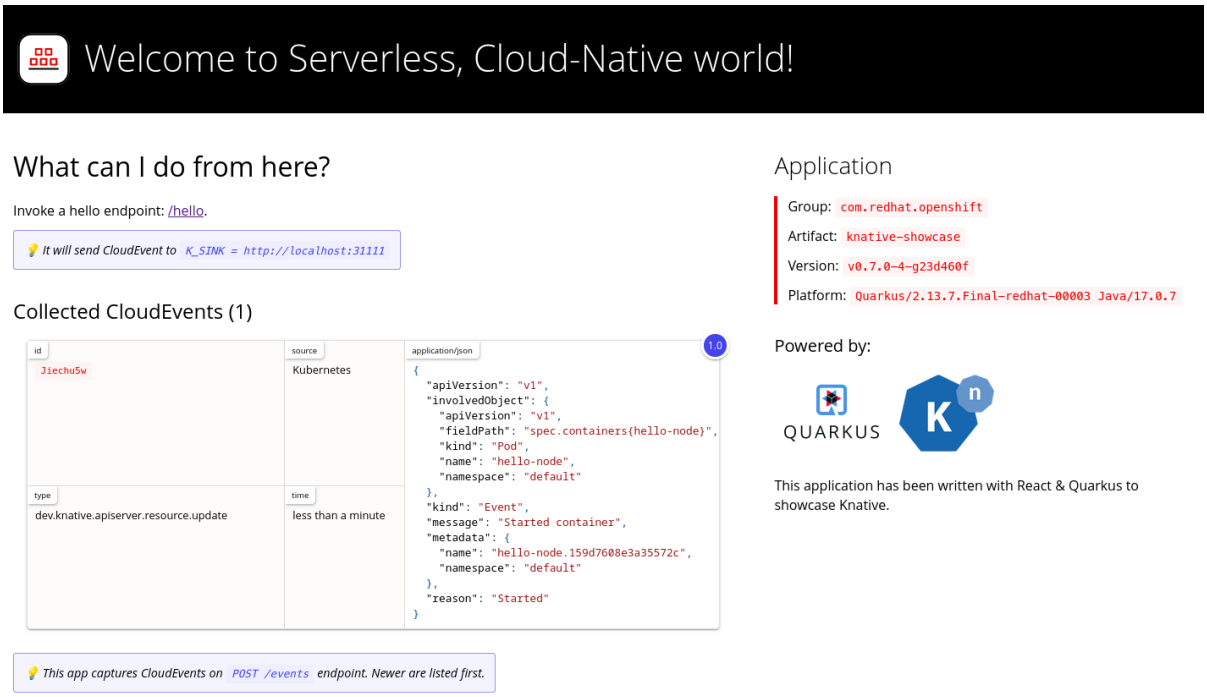
検証

Kubernetes イベントが Knative に送信されたことを確認するには、イベント表示ログを確認するか、Web ブラウザーを使用してイベントを確認してください。

- Web ブラウザーでイベントを表示するには、次のコマンドで返されたリンクを開きます。

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

図2.2 ブラウザーページの例



- ターミナルでログを確認するには、次のコマンドを入力して Pod のイベント表示ログを表示します。

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

出力例

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

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

API サーバーソースの削除

1. トリガーを削除します。

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

2. イベントソースを削除します。

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

3. サービスアカウント、クラスターロール、およびクラスターバインディングを削除します。

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

2.4. PING ソースの作成

ping ソースは、一定のペイロードを使用して ping イベントをイベントコンシューマーに定期的に送信するために使用されるイベントソースです。ping ソースを使用すると、タイマーと同様にイベントの送信をスケジュールできます。

2.4.1. Web コンソールを使用した ping ソースの作成

Knative Eventing がクラスターにインストールされると、Web コンソールを使用して ping ソースを作成できます。OpenShift Container Platform Web コンソールを使用すると、イベントソースを作成するための合理的で直感的なユーザーインターフェイスが提供されます。

前提条件

- OpenShift Container Platform Web コンソールにログインしている。
- OpenShift Serverless Operator、Knative Serving、および Knative Eventing がクラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. PingSource が機能していることを確認するには、受信メッセージをサービスのログにダンプする単純な Knative サービスを作成します。

   a. **Developer** パースペクティブで、**+Add → YAML** に移動します。

   b. サンプル YAML をコピーします。

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

   c. **Create** をクリックします。

2. 直前の手順で作成したサービスと同じ namespace、またはイベントの送信先となる他のシンクと同じ namespace に ping ソースを作成します。

   a. **Developer** パースペクティブで、**+Add → Event Source** に移動します。**Event Sources** ページが表示されます。

   b. オプション: イベントソースに複数のプロバイダーがある場合は、**Providers** 一覧から必要なプロバイダーを選択し、プロバイダーから利用可能なイベントソースをフィルターします。

   c. **Ping Source** を選択してから **Create Event Source** をクリックします。**Create Event Source** ページが表示されます。

注記

Form view または **YAML view** を使用して **PingSource** 設定を設定し、これらのビューを切り換えることができます。ビューの切り替え時に、データは永続化されます。

   d. **Schedule** の値を入力します。この例では、値は ***/2 * * * *** であり、2 分ごとにメッセージを送信する PingSource を作成します。

   e. オプション: **Data** の値を入力できます。これはメッセージのペイロードです。

   f. **Target** セクションで、イベントシンクを選択します。これは **Resource** または **URI** のいずれかです。

      i. **Resource** を選択して、チャネル、ブローカー、またはサービスをイベントソースのシンクとして使用します。この例では、前の手順で作成した **event-display** サービスをターゲット **Resource** として使用します。

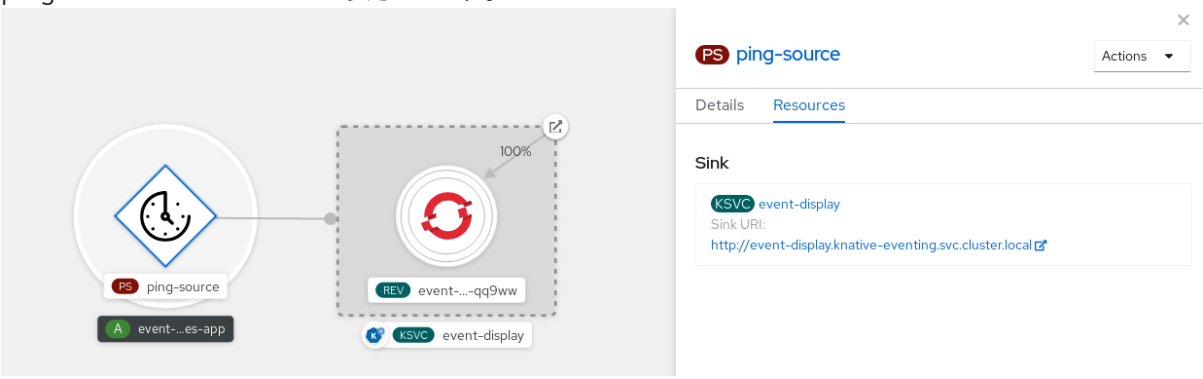
      ii. **URI** を選択して、イベントのルーティング先となる URI (Uniform Resource Identifier) を指定します。

   g. **Create** をクリックします。

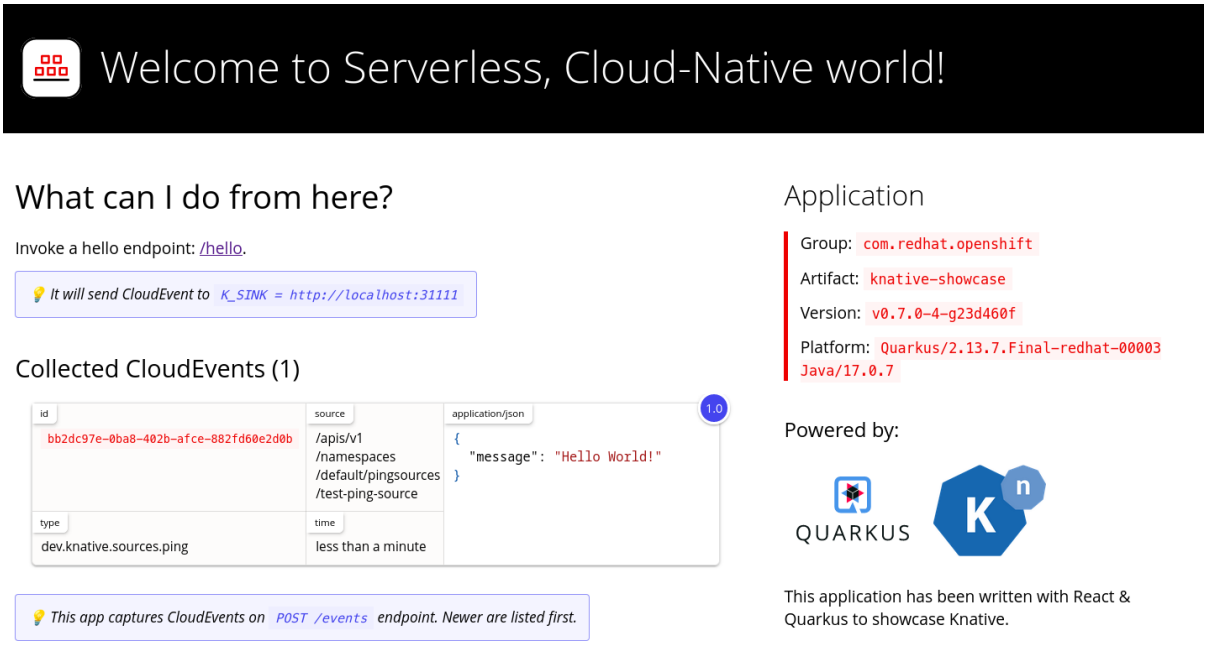
検証

Topology ページを表示して、ping ソースが作成され、シンクに接続されていることを確認できます。

1. **Developer** パースペクティブで、**Topology** に移動します。
2. ping ソースおよびシンクを表示します。



3. イベント表示サービスを Web ブラウザーで表示します。Web UI に ping ソースイベントが表示されるはずです。



ping ソースの削除

1. **Topology** ビューに移動します。
2. API サーバーソースを右クリックし、**Delete Ping Source** を選択します。

2.4.2. Knative CLI を使用した ping ソースの作成

kn source ping create コマンドを使用し、Knative (**kn**) CLI を使用して ping ソースを作成できます。Knative CLI を使用してイベントソースを作成すると、YAML ファイルを直接変更するよりも合理化された直感的なユーザーインターフェイスが提供されます。

前提条件

- OpenShift Serverless Operator、Knative Serving、および Knative Eventing がクラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- オプション: この手順の検証手順を使用する場合は、OpenShift CLI (**oc**) がインストールされている。

手順

1. ping ソースが機能していることを確認するには、受信メッセージをサービスのログにダンプする単純な Knative サービスを作成します。

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

2. 要求する必要のある ping イベントのセットごとに、PingSource をイベントコンシューマーと同じ namespace に作成します。

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

3. 以下のコマンドを入力し、出力を検査して、コントローラーが正しくマップされていることを確認します。

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

出力例

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

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

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

```
++ ValidSchedule        15s
++ EventTypeProvided    15s
++ ResourcesCorrect     15s
```

検証

シンク Pod のログを確認して、Kubernetes イベントが Knative イベントに送信されていることを確認できます。

デフォルトでは、Knative サービスは、60 秒以内にトラフィックを受信しないと Pod を終了します。このガイドの例では、新たに作成される Pod で各メッセージが確認されるように 2 分ごとにメッセージを送信する ping ソースを作成します。

1. 作成された新規 Pod を監視します。

```
$ watch oc get pods
```

2. Ctrl+C を使用して Pod の監視をキャンセルし、作成された Pod のログを確認します。

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

出力例

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

ping ソースの削除

- ping ソースを削除します。

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

2.4.2.1. Knative CLI シンクフラグ

Knative (**kn**) CLI を使用してイベントソースを作成する場合は、**--sink** フラグを使用して、そのリソースからイベントが送信されるシンクを指定できます。シンクは、他のリソースから受信イベントを受信できる、アドレス指定可能または呼び出し可能な任意のリソースです。

以下の例では、サービスの **http://event-display.svc.cluster.local** をシンクとして使用するシンクバインディングを作成します。

シンクフラグを使用したコマンドの例

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

1 **http://event-display.svc.cluster.local** の **svc** は、シンクが Knative サービスであることを判別します。他のデフォルトのシンクの接頭辞には、**channel** および **broker** が含まれます。

2.4.3. YAML を使用した ping ソースの作成

YAML ファイルを使用して Knative リソースを作成する場合は、宣言的 API を使用するため、再現性の高い方法でイベントソースを宣言的に記述できます。YAML を使用して Serverless ping を作成するには、**PingSource** オブジェクトを定義する YAML ファイルを作成し、**oc apply** を使用してこれを適用する必要があります。

PingSource オブジェクトの例

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

1 CRON 式 を使用して指定されるイベントのスケジュール。

2 JSON でエンコードされたデータ文字列として表現されるイベントメッセージの本体。

3 これらはイベントコンシューマーの詳細です。この例では、**event-display** という名前の Knative サービスを使用しています。

前提条件

- OpenShift Serverless Operator、Knative Serving、および Knative Eventing がクラスターにインストールされている。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. ping ソースが機能していることを確認するには、受信メッセージをサービスのログにダンプする単純な Knative サービスを作成します。

   a. サービス YAML ファイルを作成します。

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

   b. サービスを作成します。

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

2. 要求する必要のある ping イベントのセットごとに、ping ソースをイベントコンシューマーと同じ namespace に作成します。

   a. ping ソースの YAML ファイルを作成します。

```
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 ソースを作成します。

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

3. 以下のコマンドを入力し、コントローラーが正しくマップされていることを確認します。

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

出力例

```
apiVersion: sources.knative.dev/v1
kind: PingSource
metadata:
  annotations:
    sources.knative.dev/creator: developer
    sources.knative.dev/lastModifier: developer
  creationTimestamp: "2020-04-07T16:11:14Z"
```

```
  generation: 1
  name: test-ping-source
  namespace: default
  resourceVersion: "55257"
  selfLink: /apis/sources.knative.dev/v1/namespaces/default/pingsources/test-ping-source
  uid: 3d80d50b-f8c7-4c1b-99f7-3ec00e0a8164
spec:
  data: '{ value: "hello" }'
  schedule: '*/2 * * * *'
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
      namespace: default
```

検証

シンク Pod のログを確認して、Kubernetes イベントが Knative イベントに送信されていることを確認できます。

デフォルトでは、Knative サービスは、60 秒以内にトラフィックを受信しないと Pod を終了します。このガイドの例では、新たに作成される Pod で各メッセージが確認されるように 2 分ごとにメッセージを送信する PingSource を作成します。

1. 作成された新規 Pod を監視します。

```
$ watch oc get pods
```

2. Ctrl+C を使用して Pod の監視をキャンセルし、作成された Pod のログを確認します。

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

出力例

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

ping ソースの削除

- ping ソースを削除します。

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

コマンドの例

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

2.5. APACHE KAFKA のソース

Apache Kafka クラスターからイベントを読み取り、これらのイベントをシンクに渡す Apache Kafka ソースを作成できます。Kafka ソースを作成するには、OpenShift Container Platform Web コンソールの Knative (**kn**) CLI を使用するか、**KafkaSource** オブジェクトを YAML ファイルとして直接作成し、OpenShift CLI (**oc**) を使用して適用します。

注記

Apache Kafka の Knative ブローカーのインストール を参照してください。

2.5.1. Web コンソールを使用した Apache Kafka イベントソースの作成

Apache Kafka の Knative ブローカー実装がクラスターにインストールされたら、Web コンソールを使用して Apache Kafka ソースを作成できます。OpenShift Container Platform Web コンソールを使用すると、Kafka ソースを作成するための合理的で直感的なユーザーインターフェイスが提供されます。

前提条件

- OpenShift Serverless Operator、Knative Serving、および **KnativeKafka** カスタムリソースがクラスターにインストールされている。
- Web コンソールにログインしている。
- インポートする Kafka メッセージを生成する Red Hat AMQ Streams (Kafka) クラスターにアクセスできる。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. **Developer** パースペクティブで、**+Add** ページに移動し、**Event Source** を選択します。
2. **Event Sources** ページで、**Type** セクションの **Kafka Source** を選択します。
3. **Kafka Source** 設定を設定します。
   a. **ブートストラップサーバー** のコンマ区切りのリストを追加します。
   b. **トピック** のコンマ区切りのリストを追加します。
   c. **コンシューマーグループ** を追加します。
   d. 作成したサービスアカウントの **Service Account Name** を選択します。
   e. **Target** セクションで、イベントシンクを選択します。これは **Resource** または **URI** のいずれかです。

i. **Resource** を選択して、チャネル、ブローカー、またはサービスをイベントソースのシンクとして使用します。

ii. **URI** を選択して、イベントのルーティング先となる URI (Uniform Resource Identifier) を指定します。

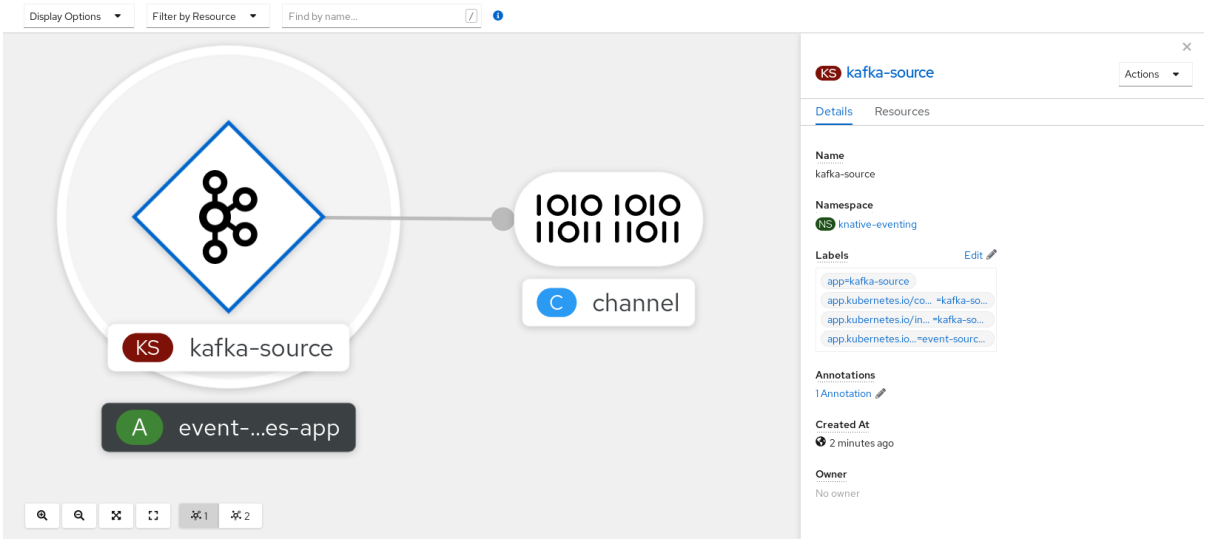
f. Kafka イベントソースの **Name** を入力します。

4. **Create** をクリックします。

検証

Topology ページを表示して、Kafka イベントソースが作成され、シンクに接続されていることを確認できます。

1. **Developer** パースペクティブで、**Topology** に移動します。
2. Kafka イベントソースおよびシンクを表示します。



2.5.2. Knative CLI を使用した Apache Kafka イベントソースの作成

kn source kafka create コマンドを使用し、Knative (**kn**) CLI を使用して Kafka ソースを作成できます。Knative CLI を使用してイベントソースを作成すると、YAML ファイルを直接変更するよりも合理化された直感的なユーザーインターフェイスが提供されます。

前提条件

- OpenShift Serverless Operator、Knative Eventing、Knative Serving、および **KnativeKafka** カスタムリソース (CR) がクラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- インポートする Kafka メッセージを生成する Red Hat AMQ Streams (Kafka) クラスターにアクセスできる。
- Knative (**kn**) CLI がインストールされている。
- オプション: この手順で検証ステップを使用する場合は、OpenShift CLI (**oc**) をインストールしている。

手順

1. Kafka イベントソースが機能していることを確認するには、受信メッセージをサービスのログにダンプする Knative サービスを作成します。

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

2. **KafkaSource** CR を作成します。

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

注記

このコマンドのプレースホルダー値は、ソース名、ブートストラップサーバー、およびトピックの値に置き換えます。

--servers、**--topics**、および **--consumergroup** オプションは、Kafka クラスターへの接続パラメーターを指定します。**--consumergroup** オプションは任意です。

3. オプション: 作成した **KafkaSource** CR の詳細を表示します。

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

出力例

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

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

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

検証手順

1. Kafka インスタンスをトリガーし、メッセージをトピックに送信します。

```
$ 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
```

プロンプトにメッセージを入力します。このコマンドは、以下を前提とします。

- Kafka クラスターが **kafka** namespace にインストールされている。
- **KafkaSource** オブジェクトが **my-topic** トピックを使用するように設定されている。

2. ログを表示して、メッセージが到達していることを確認します。

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

出力例

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

2.5.2.1. Knative CLI シンクフラグ

Knative (**kn**) CLI を使用してイベントソースを作成する場合は、**--sink** フラグを使用して、そのリソースからイベントが送信されるシンクを指定できます。シンクは、他のリソースから受信イベントを受信できる、アドレス指定可能または呼び出し可能な任意のリソースです。

以下の例では、サービスの **http://event-display.svc.cluster.local** をシンクとして使用するシンクバインディングを作成します。

シンクフラグを使用したコマンドの例

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

1 **http://event-display.svc.cluster.local** の **svc** は、シンクが Knative サービスであることを判別します。他のデフォルトのシンクの接頭辞には、**channel** および **broker** が含まれます。

2.5.3. YAML を使用した Apache Kafka イベントソースの作成

YAML ファイルを使用して Knative リソースを作成する場合は、宣言的 API を使用するため、再現性の高い方法でアプリケーションを宣言的に記述できます。YAML を使用して Kafka ソースを作成するに

は、**KafkaSource** オブジェクトを定義する YAML ファイルを作成し、**oc apply** コマンドを使用してそれを適用する必要があります。

前提条件

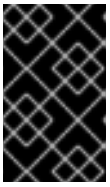
- OpenShift Serverless Operator、Knative Serving、および **KnativeKafka** カスタムリソースがクラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- インポートする Kafka メッセージを生成する Red Hat AMQ Streams (Kafka) クラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **KafkaSource** オブジェクトを YAML ファイルとして作成します。

```
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. コンシューマーグループは、同じグループ ID を使用し、トピックからデータを消費するコンシューマーのグループです。
2. トピックは、データの保存先を提供します。各トピックは、1 つまたは複数のパーティションに分割されます。
3. シンクは、イベントがソースから送信される場所を指定します。



重要

OpenShift Serverless 上の **KafkaSource** オブジェクトの API の **v1beta1** バージョンのみがサポートされます。非推奨となった **v1alpha1** バージョンの API は使用しないでください。

KafkaSource オブジェクトの例

```
apiVersion: sources.knative.dev/v1beta1
kind: KafkaSource
metadata:
```

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

2. **KafkaSource** YAML ファイルを適用します。

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

検証

- 以下のコマンドを入力して、Kafka イベントソースが作成されたことを確認します。

```
$ oc get pods
```

出力例

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

2.5.4. Apache Kafka ソースの SASL 認証の設定

Simple Authentication and Security Layer(SASL) は、Apache Kafka が認証に使用します。クラスターで SASL 認証を使用する場合、ユーザーは Kafka クラスターと通信するために Knative に認証情報を提供する必要があります。そうしないと、イベントを生成または消費できません。

前提条件

- OpenShift Container Platform でクラスターまたは専用の管理者パーミッションを持っている。
- OpenShift Serverless Operator、Knative Eventing、および **KnativeKafka** CR は、OpenShift Container Platform クラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- Kafka クラスターのユーザー名およびパスワードがある。
- 使用する SASL メカニズムを選択している (例: **PLAIN**、**SCRAM-SHA-256**、または **SCRAM-SHA-512**)。
- TLS が有効になっている場合は、Kafka クラスターの **ca.crt** 証明書ファイルがある。
- OpenShift (**oc**) CLI がインストールされている。

手順

1. 選択された namespace にシークレットとして証明書ファイルを作成します。

```
$ 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 タイプは **PLAIN**、**SCRAM-SHA-256**、または **SCRAM-SHA-512** です。

2. Kafka ソースを作成または変更して、次の **spec** 設定が含まれるようにします。

```
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 パブリッククラウドの Kafka サービスを使用している場合は、**caCert** 仕様は必要ありません。

2.6. カスタムイベントソース

Knative に含まれていないイベントプロデューサーや、**CloudEvent** 形式ではないイベントを生成するプロデューサーからイベントを Ingress する必要がある場合は、カスタムイベントソースを使用してこれを実行できます。カスタムイベントソースは、次のいずれかの方法で作成できます。

- シンクバインディングを作成して、**PodSpecable** オブジェクトをイベントソースとして使用します。
- コンテナーソースを作成して、コンテナーをイベントソースとして使用します。

2.6.1. シンクバインディング

SinkBinding オブジェクトは、イベント生成を配信アドレス指定から切り離すことをサポートします。シンクバインディングは、**イベントプロデューサー** をイベントコンシューマーまたは **シンク** に接続するために使用されます。イベントプロデューサーは、**PodSpec** テンプレートを組み込む Kubernetes リソースであり、イベントを生成します。シンクは、イベントを受信できるアドレス指定可能な Kubernetes オブジェクトです。

SinkBinding オブジェクトは、環境変数をシンクの **PodTemplateSpec** に挿入します。つまり、アプリケーションコードが Kubernetes API と直接対話してイベントの宛先を見つける必要はありません。これらの環境変数は以下のとおりです。

K_SINK

解決されたシンクの URL。

K_CE_OVERRIDES

アウトバウンドイベントの上書きを指定する JSON オブジェクト。

注記

現在、**SinkBinding** オブジェクトはサービスのカスタムリビジョン名をサポートしません。

2.6.1.1. YAML を使用したシンクバインディングの作成

YAML ファイルを使用して Knative リソースを作成する場合は、宣言的 API を使用するため、再現性の高い方法でイベントソースを宣言的に記述できます。YAML を使用してシンクバインディングを作成するには、**SinkBinding** オブジェクトを定義する YAML ファイルを作成し、**oc apply** コマンドを使用してそれを適用する必要があります。

前提条件

- OpenShift Serverless Operator、Knative Serving、および Knative Eventing がクラスターにインストールされている。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. シンクバインディングが正しく設定されていることを確認するには、受信メッセージをダンプする Knative イベント表示サービスまたはイベントシンクを作成します。
   a. サービス YAML ファイルを作成します。

      サービス YAML ファイルの例

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

b. サービスを作成します。

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

2. イベントをサービスに転送するシンクバインディングインスタンスを作成します。

a. シンクバインディング YAML ファイルを作成します。

サービス YAML ファイルの例

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

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

(1) この例では、ラベル **app: heartbeat-cron** を指定したジョブがイベントシンクにバインドされます。

b. シンクバインディングを作成します。

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

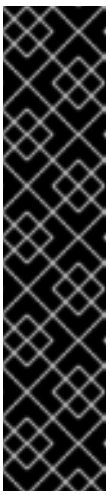
3. **CronJob** オブジェクトを作成します。

a. cron ジョブの YAML ファイルを作成します。

cron ジョブの YAML ファイルの例

```
apiVersion: batch/v1
kind: CronJob
```

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



重要

シンクバインディングを使用するには、**bindings.knative.dev/include=true** ラベルを Knative リソースに手動で追加する必要があります。

たとえば、このラベルを **CronJob** インスタンスに追加するには、以下の行を **Job** リソースの YAML 定義に追加します。

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

b. cron ジョブを作成します。

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

4. 以下のコマンドを入力し、出力を検査して、コントローラーが正しくマップされていることを確認します。

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

出力例

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

検証

メッセージダンパー機能ログを確認して、Kubernetes イベントが Knative イベントシンクに送信されていることを確認できます。

1. コマンドを入力します。

```
$ oc get pods
```

2. コマンドを入力します。

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

出力例

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

2.6.1.2. Knative CLI を使用したシンクバインディングの作成

kn source binding create コマンドを使用し、Knative (**kn**) を使用してシンクバインディングを作成できます。Knative CLI を使用してイベントソースを作成すると、YAML ファイルを直接変更するよりも合理化された直感的なユーザーインターフェイスが提供されます。

前提条件

- OpenShift Serverless Operator、Knative Serving、および Knative Eventing がクラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- Knative (**kn**) CLI をインストールしている。
- OpenShift CLI (**oc**) がインストールされている。

注記

以下の手順では、YAML ファイルを作成する必要があります。

サンプルで使用されたもので YAML ファイルの名前を変更する場合は、必ず対応する CLI コマンドを更新する必要があります。

手順

1. シンクバインディングが正しく設定されていることを確認するには、受信メッセージをダンプする Knative イベント表示サービスまたはイベントシンクを作成します。

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

2. イベントをサービスに転送するシンクバインディングインスタンスを作成します。

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

3. **CronJob** オブジェクトを作成します。
   a. cron ジョブの YAML ファイルを作成します。

cron ジョブの YAML ファイルの例

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: heartbeat-cron
spec:
  # Run every minute
  schedule: "* * * * *"
  jobTemplate:
    metadata:
      labels:
        app: heartbeat-cron
        bindings.knative.dev/include: "true"
    spec:
```

```
template:
  spec:
    restartPolicy: Never
    containers:
      - name: single-heartbeat
        image: quay.io/openshift-knative/heartbeats:latest
        args:
          - --period=1
        env:
          - name: ONE_SHOT
            value: "true"
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
```

重要

シンクバインディングを使用するには、**bindings.knative.dev/include=true** ラベルを Knative CR に手動で追加する必要があります。

たとえば、このラベルを **CronJob** CR に追加するには、以下の行を **Job** CR の YAML 定義に追加します。

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

b. cron ジョブを作成します。

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

4. 以下のコマンドを入力し、出力を検査して、コントローラーが正しくマップされていることを確認します。

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

出力例

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

```
    app:      heartbeat-cron
Sink:
  Name:       event-display
  Resource:   Service (serving.knative.dev/v1)

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

検証

メッセージダンパー機能ログを確認して、Kubernetes イベントが Knative イベントシンクに送信されていることを確認できます。

- 以下のコマンドを入力して、メッセージダンパー機能ログを表示します。

```
$ oc get pods
```

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

出力例

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

2.6.1.2.1. Knative CLI シンクフラグ

Knative (**kn**) CLI を使用してイベントソースを作成する場合は、**--sink** フラグを使用して、そのリソースからイベントが送信されるシンクを指定できます。シンクは、他のリソースから受信イベントを受信できる、アドレス指定可能または呼び出し可能な任意のリソースです。

以下の例では、サービスの **http://event-display.svc.cluster.local** をシンクとして使用するシンクバインディングを作成します。

シンクフラグを使用したコマンドの例

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
```

```
--subject "Job:batch/v1:app=heartbeat-cron" \
--sink http://event-display.svc.cluster.local \ 1
--ce-override "sink=bound"
```

1 **http://event-display.svc.cluster.local** の **svc** は、シンクが Knative サービスであることを判別します。他のデフォルトのシンクの接頭辞には、**channel** および **broker** が含まれます。

2.6.1.3. Web コンソールを使用したシンクバインディングの作成

Knative Eventing がクラスターにインストールされると、Web コンソールを使用して シンクバインディングを作成できます。OpenShift Container Platform Web コンソールを使用すると、イベントソースを作成するための合理的で直感的なユーザーインターフェイスが提供されます。

前提条件

- OpenShift Container Platform Web コンソールにログインしている。
- OpenShift Serverless Operator、Knative Serving、および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. シンクとして使用する Knative サービスを作成します。
   a. **Developer** パースペクティブで、**+Add → YAML** に移動します。
   b. サンプル YAML をコピーします。

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

   c. **Create** をクリックします。
2. イベントソースとして使用される **CronJob** リソースを作成し、1 分ごとにイベントを送信します。
   a. **Developer** パースペクティブで、**+Add → YAML** に移動します。
   b. サンプル YAML をコピーします。

```
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** ラベルを含めるようにしてください。OpenShift Serverless のデフォルトの namespace 選択動作は包含モードを使用します。

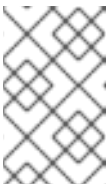
c. **Create** をクリックします。

3. 直前の手順で作成したサービスと同じ namespace、またはイベントの送信先となる他のシンクと同じ namespace にシンクバインディングを作成します。

a. **Developer** パースペクティブで、**+Add → Event Source** に移動します。**Event Sources** ページが表示されます。

b. オプション: イベントソースに複数のプロバイダーがある場合は、**Providers** 一覧から必要なプロバイダーを選択し、プロバイダーから利用可能なイベントソースをフィルターします。

c. **Sink Binding** を選択し、**Create Event Source** をクリックします。**Create Event Source** ページが表示されます。



注記

Form view または **YAML view** を使用して **Sink Binding** 設定を設定し、ビューを切り替えることができます。ビューの切り替え時に、データは永続化されます。

d. **apiVersion** フィールドに **batch/v1** を入力します。

e. **Kind** フィールドに **Job** と入力します。

注記

CronJob の種類は OpenShift Serverless シンクバインディングで直接サポートされていないため、**Kind** フィールドは cron ジョブオブジェクト自体ではなく、cron ジョブで作成される **Job** オブジェクトをターゲットにする必要があります。

f. **Target** セクションで、イベントシンクを選択します。これは **Resource** または **URI** のいずれかです。

i. **Resource** を選択して、チャネル、ブローカー、またはサービスをイベントソースのシンクとして使用します。この例では、前の手順で作成した **event-display** サービスをターゲット **Resource** として使用します。

ii. **URI** を選択して、イベントのルーティング先となる URI (Uniform Resource Identifier) を指定します。

g. **Match labels** セクションで以下を実行します。

i. **Name** フィールドに **app** と入力します。

ii. **Value** フィールドに **heartbeat-cron** と入力します。

注記

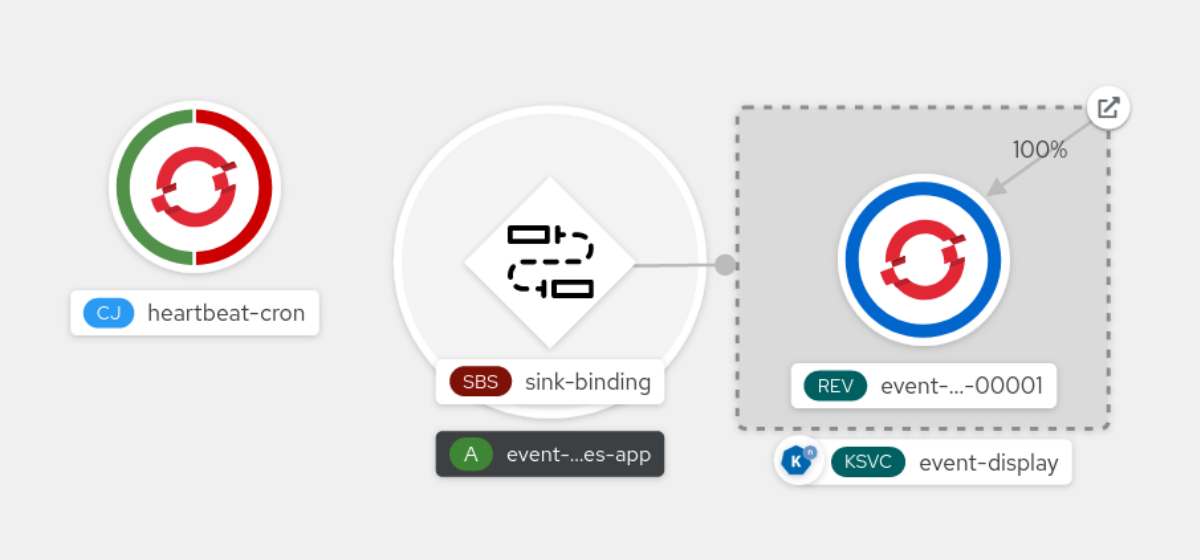
ラベルセレクターは、リソース名ではなくシンクバインディングで cron ジョブを使用する場合に必要になります。これは、cron ジョブで作成されたジョブには予測可能な名前がなく、名前に無作為に生成される文字列が含まれているためです。たとえば、**hearthbeat-cron-1cc23f** になります。

h. **Create** をクリックします。

検証

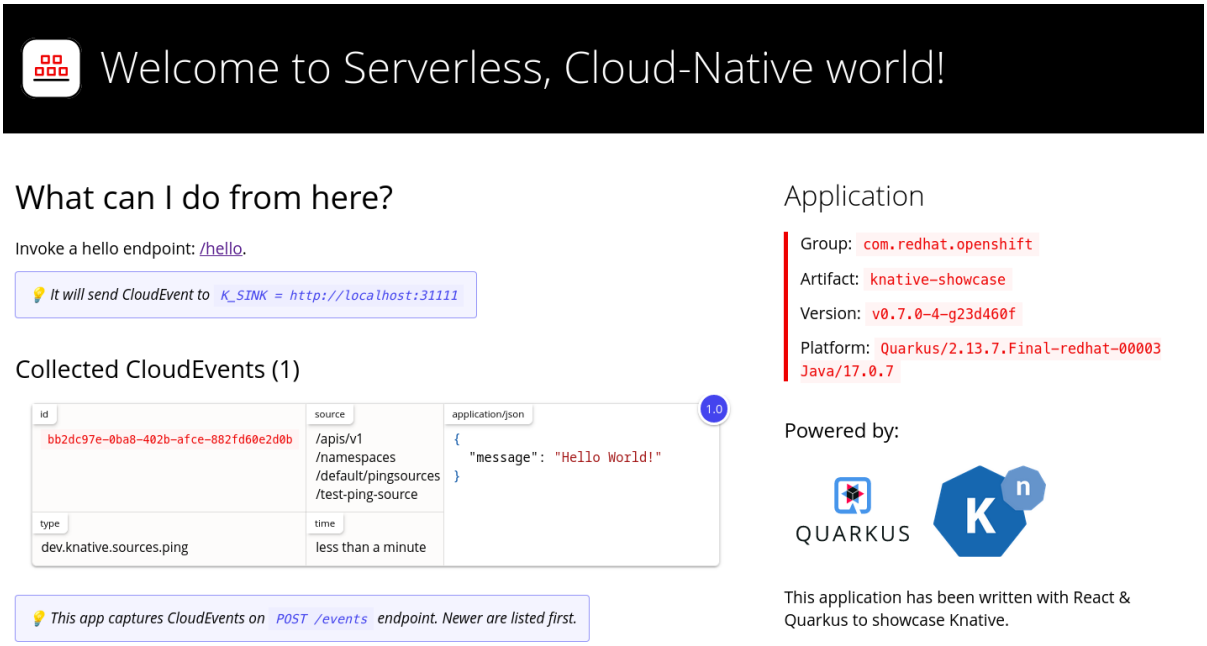
Topology ページおよび Pod ログを表示して、シンクバインディング、シンク、および cron ジョブが正常に作成され、機能していることを確認できます。

1. **Developer** パースペクティブで、**Topology** に移動します。
2. シンクバインディング、シンク、およびハートビートの cron ジョブを表示します。



3. シンクバインディングが追加されると、正常なジョブが cron ジョブによって登録されていることを確認します。つまり、シンクバインディングは cron ジョブで作成されたジョブが正常に再設定されることを意味します。

4. **イベント表示** サービスを参照して、ハートビート cron ジョブによって生成されたイベントを確認します。



2.6.1.4. シンクバインディング参照

シンクバインディングを作成して、**PodSpecable** オブジェクトをイベントソースとして使用できます。**SinkBinding** オブジェクトを作成するときに、複数のパラメーターを設定できます。

SinkBinding オブジェクトは以下のパラメーターをサポートします。

フィールド	説明	必須またはオプション
apiVersion	API バージョンを指定します (例: **sources.knative.dev/v1**)。	必須

フィールド	説明	必須またはオプション
kind	このリソースオブジェクトを **SinkBinding** オブジェクトとして特定します。	必須
metadata	**SinkBinding** オブジェクトを一意に識別するメタデータを指定します。たとえば、**name** です。	必須
spec	この **SinkBinding** オブジェクトの設定情報を指定します。	必須
spec.sink	シンクとして使用する URI に解決するオブジェクトへの参照。	必須
spec.subject	ランタイムコントラクトがバインディング実装によって拡張されるリソースを参照します。	必須
spec.ceOverrides	上書きを定義して、シンクに送信されたイベントへの出力形式および変更を制御します。	オプション

2.6.1.4.1. Subject パラメーター

Subject パラメーターは、ランタイムコントラクトがバインディング実装によって拡張されるリソースを参照します。**Subject** 定義に複数のフィールドを設定できます。

Subject 定義は、以下のフィールドをサポートします。

フィールド	説明	必須またはオプション
apiVersion	参照先の API バージョン。	必須
kind	参照先の種類。	必須
namespace	参照先の namespace。省略されている場合、デフォルトはオブジェクトの namespace に設定されます。	オプション
name	参照先の名前。	**selector** を設定する場合は、使用しないでください。
selector	参照先のセレクター。	**name** を設定する場合は、使用しないでください。

フィールド	説明	必須またはオプション
selector.matchExpressions	ラベルセレクターの要件のリストです。	**matchExpressions** または **matchLabels** のいずれかのみを使用します。
selector.matchExpressions.key	セレクターが適用されるラベルキー。	**matchExpressions** を使用する場合に必須です。
selector.matchExpressions.operator	キーと値のセットの関係を表します。有効な演算子は **In**、**NotIn**、**Exists**、および **DoesNotExist** です。	**matchExpressions** を使用する場合に必須です。
selector.matchExpressions.values	文字列値の配列。**operator** パラメーターの値が **In** または **NotIn** の場合、値配列が空でないようにする必要があります。**operator** パラメーターの値が **Exists** または **DoesNotExist** の場合、値の配列は空である必要があります。この配列は、ストラテジーに基づいたマージパッチの適用中に置き換えられます。	**matchExpressions** を使用する場合に必須です。
selector.matchLabels	キーと値のペアのマップ。**matchLabels** マップの各キーと値のペアは **matchExpressions** の要素と同じです。ここで、キーフィールドは **matchLabels.<key>** で、**operator** は **In** で、**values** の配列には **matchLabels.<value>** のみが含まれます。	**matchExpressions** または **matchLabels** のいずれかのみを使用します。

サブジェクトパラメーターの例

以下の YAML の場合は、**default** namespace の **mysubject** という名前の **Deployment** オブジェクトが選択されます。

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

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

以下の YAML の場合は、**default** namespace にラベル **working=example** が設定された **Job** オブジェクトが選択されます。

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

以下の YAML の場合は、**default** namespace にラベル **working=example** または **working=sample** が含まれる **Pod** オブジェクトが選択されます。

```
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
  ...
```

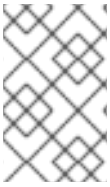
2.6.1.4.2. CloudEvent オーバーライド

ceOverrides 定義は、シンクに送信される CloudEvent の出力形式および変更を制御するオーバーライドを提供します。**ceOverrides** 定義に複数のフィールドを設定できます。

ceOverrides の定義は、以下のフィールドをサポートします。

フィールド	説明	必須またはオプション

フィールド	説明	必須またはオプション
extensions	アウトバウンドイベントで追加または上書きされる属性を指定します。各 **extensions** のキーと値のペアは、属性拡張機能としてイベントに個別に設定されます。	オプション



注記

拡張子として許可されるのは、有効な **CloudEvent** 属性名のみです。拡張機能オーバーライド設定から仕様定義属性を設定することはできません。たとえば、**type** 属性を変更することはできません。

CloudEvent オーバーライドの例

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

これにより、**subject** に **K_CE_OVERRIDES** 環境変数が設定されます。

出力例

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

2.6.1.4.3. include ラベル

シンクバインディングを使用するには、**bindings.knative.dev/include: "true"** ラベルをリソースまたはリソースが含まれる namespace のいずれかに割り当てる必要があります。リソース定義にラベルが含まれていない場合、クラスター管理者は以下を実行してこれを namespace に割り当てることができます。

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

2.6.1.5. Service Mesh とシンクバインディングの統合

前提条件

- Service Mesh を OpenShift Serverless と統合しました。

手順

1. **ServiceMeshMemberRoll** のメンバーである namespace に **Service** を作成します。

```
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** のメンバーである namespace。
2. Service Mesh サイドカーは Knative サービス Pod に挿入します。

2. **Service** リソースを適用します。

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

3. **SinkBinding** リソースを作成します。

```
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** のメンバーである namespace。
2. この例では、ラベル **app: heartbeat-cron** を指定したジョブがイベントシンクにバインドされます。

4. **SinkBinding** リソースを適用します。

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

5. **CronJob** を作成します。

```
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** のメンバーである namespace。

2 Service Mesh サイドカーを **CronJob** Pod に挿入します。

6. **CronJob** リソースを適用します。

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

検証

イベントが Knative イベントシンクに送信されたことを確認するには、メッセージダンパー機能のログを調べます。

1. 以下のコマンドを入力します。

```
$ oc get pods
```

2. 以下のコマンドを入力します。

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

出力例

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

関連情報

- Service Mesh と OpenShift Serverless の統合

2.6.2. コンテナーソース

コンテナーソースは、イベントを生成し、イベントをシンクに送信するコンテナーイメージを作成します。コンテナーソースを使用して、イメージ URI を使用するコンテナーイメージおよび **ContainerSource** オブジェクトを作成して、カスタムイベントソースを作成できます。

2.6.2.1. コンテナーイメージを作成するためのガイドライン

コンテナーソースコントローラーには、**K_SINK** および **K_CE_OVERRIDES** の 2 つの環境変数が注入されます。これらの変数は、それぞれ **sink** および **ceOverrides** 仕様から解決されます。イベントは、**K_SINK** 環境変数で指定されたシンク URI に送信されます。メッセージは、**CloudEvent** HTTP 形式を使用して **POST** として送信する必要があります。

コンテナーイメージの例

以下は、ハートビートコンテナーイメージの例になります。

```
package main

import (
 "context"
```

```
 "encoding/json"
 "flag"
 "fmt"
 "log"
 "os"
 "strconv"
 "time"

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

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

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

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

func init() {
 flag.StringVar(&eventSource, "eventSource", "", "the event-source (CloudEvents)")
 flag.StringVar(&eventType, "eventType", "dev.knative.eventing.samples.heartbeat", "the event-type
(CloudEvents)")
 flag.StringVar(&sink, "sink", "", "the host url to heartbeat to")
 flag.StringVar(&label, "label", "", "a special label")
 flag.StringVar(&periodStr, "period", "5", "the number of seconds between heartbeats")
}

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

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

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

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

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

func main() {
 flag.Parse()
```

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

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

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

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

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

var period time.Duration
if p, err := strconv.Atoi(periodStr); err != nil {
 period = time.Duration(5) * time.Second
} else {
 period = time.Duration(p) * time.Second
}

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

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

 event := cloudevents.NewEvent("1.0")
```

```
  event.SetType(eventType)
  event.SetSource(eventSource)
  event.SetExtension("the", 42)
  event.SetExtension("heart", "yes")
  event.SetExtension("beats", true)

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

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

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

  if env.OneShot {
   return
  }

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

以下は、以前のハートビートコンテナーイメージを参照するコンテナーソースの例です。

```
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 CLI を使用したコンテナーソースの作成および管理

kn source container コマンドを使用し、Knative (**kn**) CLI を使用してコンテナーソースを作成および管理できます。Knative CLI を使用してイベントソースを作成すると、YAML ファイルを直接変更するよりも合理化された直感的なユーザーインターフェイスが提供されます。

コンテナーソースの作成

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

コンテナーソースの削除

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

コンテナーソースの記述

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

既存のコンテナーソースをリスト表示

```
$ kn source container list
```

既存のコンテナーソースを YAML 形式でリスト表示

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

コンテナーソースの更新

このコマンドにより、既存のコンテナーソースのイメージ URI が更新されます。

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

2.6.2.3. Web コンソールを使用したコンテナーソースの作成

Knative Eventing がクラスターにインストールされると、Web コンソールを使用してコンテナーソースを作成できます。OpenShift Container Platform Web コンソールを使用すると、イベントソースを作成するための合理的で直感的なユーザーインターフェイスが提供されます。

前提条件

- OpenShift Container Platform Web コンソールにログインしている。
- OpenShift Serverless Operator、Knative Serving、および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. Developer パースペクティブで、+Add → Event Source に移動します。Event Sources ページが表示されます。
2. Container Source を選択してから Create Event Sourceをクリックします。Create Event Source ページが表示されます。
3. Form view または YAML view を使用して、Container Source 設定を設定します。

注記

Form view と YAML view 間で切り換えることができます。ビューの切り替え時に、データは永続化されます。

   a. Image フィールドに、コンテナーソースが作成したコンテナーで実行するイメージの URI を入力します。
   b. Name フィールドにイメージの名前を入力します。
   c. オプション: Arguments フィールドで、コンテナーに渡す引数を入力します。
   d. オプション: Environment variables フィールドで、コンテナーに設定する環境変数を追加します。
   e. Target セクションで、イベントシンクを選択します。これは Resource または URI のいずれかです。
      i. Resource を選択して、チャネル、ブローカー、またはサービスをイベントソースのシンクとして使用します。
      ii. URI を選択して、イベントのルーティング先となる URI (Uniform Resource Identifier) を指定します。
4. コンテナーソースの設定が完了したら、Create をクリックします。

2.6.2.4. コンテナーソースのリファレンス

ContainerSource オブジェクトを作成することにより、コンテナーをイベントソースとして使用できます。**ContainerSource** オブジェクトを作成するときに、複数のパラメーターを設定できます。

ContainerSource オブジェクトは以下のフィールドをサポートします。

フィールド	説明	必須またはオプション
apiVersion	API バージョンを指定します (例: **sources.knative.dev/v1**)。	必須
kind	このリソースオブジェクトを **ContainerSource** オブジェクトとして特定します。	必須
metadata	**ContainerSource** オブジェクトを一意に識別するメタデータを指定します。たとえば、**name** です。	必須

フィールド	説明	必須またはオプション
spec	この **ContainerSource** オブジェクトの設定情報を指定します。	必須
spec.sink	シンクとして使用する URI に解決するオブジェクトへの参照。	必須
spec.template	**ContainerSource** オブジェクトの **template** 仕様。	必須
spec.ceOverrides	上書きを定義して、シンクに送信されたイベントへの出力形式および変更を制御します。	オプション

テンプレートパラメーターの例

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

2.6.2.4.1. CloudEvent オーバーライド

ceOverrides 定義は、シンクに送信される CloudEvent の出力形式および変更を制御するオーバーライドを提供します。**ceOverrides** 定義に複数のフィールドを設定できます。

ceOverrides の定義は、以下のフィールドをサポートします。

フィールド	説明	必須またはオプション
extensions	アウトバウンドイベントで追加または上書きされる属性を指定します。各 **extensions** のキーと値のペアは、属性拡張機能としてイベントに個別に設定されます。	オプション

注記

拡張子として許可されるのは、有効な **CloudEvent** 属性名のみです。拡張機能オーバーライド設定から仕様定義属性を設定することはできません。たとえば、**type** 属性を変更することはできません。

CloudEvent オーバーライドの例

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

これにより、**subject** に **K_CE_OVERRIDES** 環境変数が設定されます。

出力例

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

2.6.2.5. Service Mesh と ContainerSource の統合

前提条件

- Service Mesh を OpenShift Serverless と統合しました。

手順

1. **ServiceMeshMemberRoll** のメンバーである namespace に **Service** を作成します。

```
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** のメンバーである namespace。

(2) Service Mesh サイドカーは Knative サービス Pod に挿入します。

2. **Service** リソースを適用します。

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

3. **ServiceMeshMemberRoll** のメンバーである namespace に **ContainerSource** オブジェクトを作成し、**event-display** に設定されたシンクを作成します。

```
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. namespace は **ServiceMeshMemberRoll** の一部です。
2. Service Mesh と **ContainerSource** オブジェクトの統合を有効にします

4. **ContainerSource** リソースを適用します。

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

検証

イベントが Knative イベントシンクに送信されたことを確認するには、メッセージダンパー機能のログを調べます。

1. 以下のコマンドを入力します。

```
$ oc get pods
```

2. 以下のコマンドを入力します。

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

出力例

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

関連情報

- Service Mesh と OpenShift Serverless の統合

2.7. 開発者パースペクティブを使用してイベントソースをイベントシンクに接続する

OpenShift Container Platform Web コンソールを使用してイベントソースを作成する場合は、イベントがソースから送信されるターゲットイベントシンクを指定できます。このイベントシンクは、他のリソースから受信イベントを受信できる、アドレス指定可能または呼び出し可能な任意のリソースです。

2.7.1. 開発者パースペクティブを使用してイベントソースをイベントシンクに接続する

前提条件

- OpenShift Serverless Operator、Knative Serving、および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Web コンソールにログインしており、**Developer** パースペクティブを使用している。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- Knative サービス、チャネル、ブローカーなどのイベントシンクを作成している。

手順

1. **+Add → Event Source** に移動して任意のタイプのイベントソースを作成し、作成するイベントソースを選択します。

2. **Create Event Source**フォームビューの **Target** セクションで、イベントシンクを選択します。これは **Resource** または **URI** のいずれかです。

    a. **Resource** を選択して、チャネル、ブローカー、またはサービスをイベントソースのシンクとして使用します。

    b. **URI** を選択して、イベントのルーティング先となる URI (Uniform Resource Identifier) を指定します。

3. **Create** をクリックします。

検証

Topology ページを表示して、イベントソースが作成され、シンクに接続されていることを確認できます。

1. **Developer** パースペクティブで、**Topology** に移動します。

2. イベントソースを表示し、接続されたイベントシンクをクリックして、右側のパネルにシンクの詳細を表示します。

第3章 イベントシンク

3.1. イベントシンク

イベントソースの作成時に、イベントをソースに対して送信するイベントシンクを指定できます。イベントシンクは、他のリソースから受信イベントを受信できる、アドレス指定可能なリソースまたは呼び出し可能なリソースです。Knative サービス、チャネル、ブローカーはすべてイベントシンクの例です。また、特定の Apache Kafka シンクタイプも利用できます。

アドレス指定可能なオブジェクトは、HTTP 経由で **status.address.url** フィールドに定義されるアドレスに配信されるイベントを受信し、確認することができます。特別な場合として、コア Kubernetes **Service** オブジェクトはアドレス指定可能なインターフェイスにも対応します。

呼び出し可能なオブジェクトは、HTTP 経由で配信されるイベントを受信し、そのイベントを変換できます。HTTP 応答で **0** または **1** の新規イベントを返します。返されるイベントは、外部イベントソースからのイベントが処理されるのと同じ方法で処理できます。

3.1.1. Knative CLI シンクフラグ

Knative (**kn**) CLI を使用してイベントソースを作成する場合は、**--sink** フラグを使用して、そのリソースからイベントが送信されるシンクを指定できます。シンクは、他のリソースから受信イベントを受信できる、アドレス指定可能または呼び出し可能な任意のリソースです。

以下の例では、サービスの **http://event-display.svc.cluster.local** をシンクとして使用するシンクバインディングを作成します。

シンクフラグを使用したコマンドの例

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

1 **http://event-display.svc.cluster.local** の **svc** は、シンクが Knative サービスであることを判別します。他のデフォルトのシンクの接頭辞には、**channel** および **broker** が含まれます。

ヒント

kn のカスタマイズ により、どの CR が Knative (**kn**) CLI コマンドの **--sink** フラグと併用できるかを設定できます。

3.2. イベントシンクの作成

イベントソースの作成時に、イベントをソースに対して送信するイベントシンクを指定できます。イベントシンクは、他のリソースから受信イベントを受信できる、アドレス指定可能なリソースまたは呼び出し可能なリソースです。Knative サービス、チャネル、ブローカーはすべてイベントシンクの例です。また、特定の Apache Kafka シンクタイプも利用できます。

イベントシンクとして使用できるリソースを作成する方法は、次のドキュメントを参照してください。

- Serverless アプリケーション

- ブローカーの作成
- チャネルの作成
- Kafka シンク

3.3. APACHE KAFKA のシンク

Apache Kafka シンクは、クラスター管理者がクラスターで Apache Kafka を有効にした場合に使用できるイベントシンクの一種です。Kafka シンクを使用して、イベントソースから Kafka トピックにイベントを直接送信できます。

3.3.1. YAML を使用した Apache Kafka シンクの作成

イベントを Kafka トピックに送信する Kafka シンクを作成できます。デフォルトでは、Kafka シンクはバイナリーコンテンツモードを使用します。これは、構造化モードよりも効率的です。YAML を使用して Kafka シンクを作成するには、**KafkaSink** オブジェクトを定義する YAML ファイルを作成してから、**ocapply** コマンドを使用してそれを適用する必要があります。

前提条件

- OpenShift Serverless Operator、Knative Serving、および **KnativeKafka** カスタムリソース (CR) がクラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- インポートする Kafka メッセージを生成する Red Hat AMQ Streams (Kafka) クラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **KafkaSink** オブジェクト定義を YAML ファイルとして作成します。

   Kafka シンク YAML

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

2. Kafka シンクを作成するには、**KafkaSink** YAML ファイルを適用します。

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

3. シンクが仕様で指定されるようにイベントソースを設定します。

API サーバーソースに接続された Kafka シンクの例

```
apiVersion: sources.knative.dev/v1alpha2
kind: ApiServerSource
metadata:
  name: <source-name> 1
  namespace: <namespace> 2
spec:
  serviceAccountName: <service-account-name> 3
  mode: Resource
  resources:
  - apiVersion: v1
    kind: Event
  sink:
    ref:
      apiVersion: eventing.knative.dev/v1alpha1
      kind: KafkaSink
      name: <sink-name> 4
```

1. イベントソースの名前。
2. イベントソースの namespace。
3. イベントソースのサービスアカウント。
4. Kafka シンクの名前。

3.3.2. OpenShift Container Platform Web コンソールを使用した Apache Kafka のイベントシンクの作成

OpenShift Container Platform Web コンソールの **Developer** パースペクティブを使用して、イベントを Kafka トピックに送信する Kafka シンクを作成できます。デフォルトでは、Kafka シンクはバイナリーコンテンツモードを使用します。これは、構造化モードよりも効率的です。

開発者は、イベントシンクを作成して、特定のソースからイベントを受信し、それを Kafka トピックに送信できます。

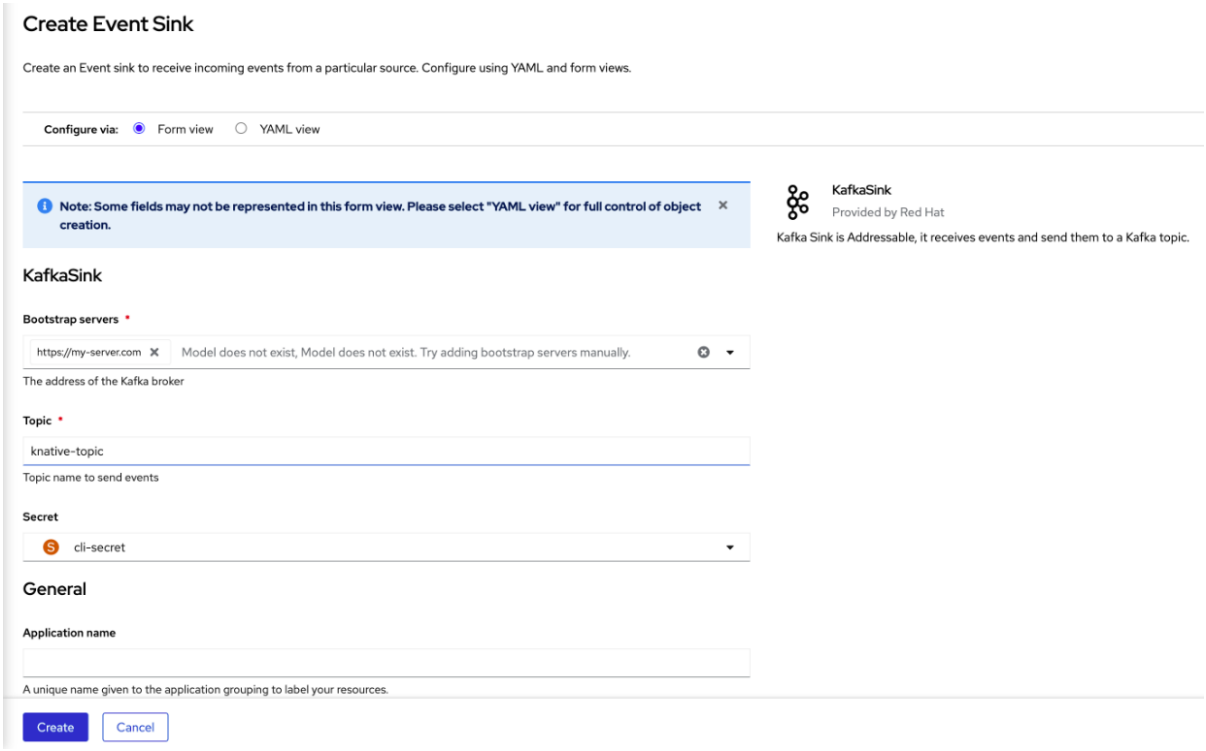
前提条件

- OperatorHub から、Knative Serving、Knative Eventing、および Apache Kafka API 用の Knative ブローカーを使用して OpenShift Serverless Operator をインストールしている。
- Kafka 環境で Kafka トピックを作成しました。

手順

1. **Developer** パースペクティブで、**+Add** ビューに移動します。
2. **Eventing カタログ** で **Event Sink** をクリックします。
3. カタログ項目で **KafkaSink** を検索してクリックします。
4. **イベントシンクの作成** をクリックします。

5. フォームビューで、ホスト名とポートの組み合わせであるブートストラップサーバーの URL を入力します。



6. イベントデータを送信するトピックの名前を入力します。
7. イベントシンクの名前を入力します。
8. **Create** をクリックします。

検証

1. **Developer** パースペクティブで、**Topology** ビューに移動します。
2. 作成したイベントシンクをクリックして、右側のパネルに詳細を表示します。

3.3.3. Apache Kafka シンクのセキュリティーの設定

Transport Layer Security(TLS) は、Apache Kafka クライアントおよびサーバーによって、Knative と Kafka 間のトラフィックを暗号化するため、および認証のために使用されます。TLS は、Apache Kafka の Knative ブローカー実装でサポートされている唯一のトラフィック暗号化方式です。

Simple Authentication and Security Layer(SASL) は、Apache Kafka が認証に使用します。クラスターで SASL 認証を使用する場合、ユーザーは Kafka クラスターと通信するために Knative に認証情報を提供する必要があります。そうしないと、イベントを生成または消費できません。

前提条件

- OpenShift Serverless Operator、Knative Eventing、および **KnativeKafka** カスタムリソース (CR) が OpenShift Container Platform クラスターにインストールされている。
- Kafka シンクが **KnativeKafka** CR で有効になっている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

- **.pem** ファイルとして Kafka クラスター CA 証明書が保存されている。
- Kafka クラスタークライアント証明書とキーが **.pem** ファイルとして保存されている。
- OpenShift (**oc**) CLI がインストールされている。
- 使用する SASL メカニズムを選択している (例: **PLAIN**、**SCRAM-SHA-256**、または **SCRAM-SHA-512**)。

手順

1. **KafkaSink** オブジェクトと同じ namespace に証明書ファイルをシークレットとして作成します。

重要

証明書とキーは PEM 形式である必要があります。

- 暗号化なしで SASL を使用した認証の場合:

```
$ 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>
```

- SASL を使用した認証と TLS を使用した暗号化の場合:

```
$ 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 パブリッククラウドで管理される Kafka サービスを使用している場合は、**ca.crt** を省略してシステムのルート CA セットを使用できます。

- TLS を使用した認証と暗号化の場合:

```
$ 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 パブリッククラウドで管理される Kafka サービスを使用している場合は、**ca.crt** を省略してシステムのルート CA セットを使用できます。

2. **KafkaSink** オブジェクトを作成または変更し、**auth** 仕様にシークレットへの参照を追加します。

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

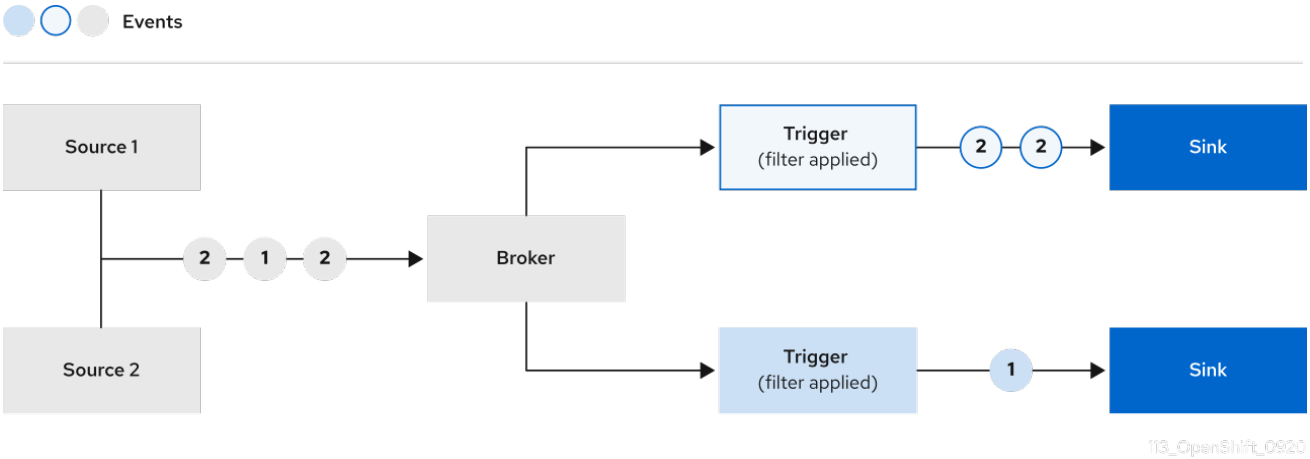
3. **KafkaSink** オブジェクトを適用します。

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

第4章 ブローカー

4.1. ブローカー

ブローカーはトリガーと組み合わせて、イベントをイベントソースからイベントシンクに配信できます。イベントは、HTTP **POST** リクエストとしてイベントソースからブローカーに送信されます。イベントがブローカーに送信された後に、それらはトリガーを使用して CloudEvent 属性 でフィルターされ、HTTP **POST** リクエストとしてイベントシンクに送信できます。



4.2. ブローカータイプ

クラスター管理者は、クラスターのデフォルトブローカー実装を設定できます。ブローカーを作成する場合は、**Broker** オブジェクトで設定を指定しない限り、デフォルトのブローカー実装が使用されます。

4.2.1. 開発目的でのデフォルトブローカーの実装

Knative は、デフォルトのチャネルベースのブローカー実装を提供します。このチャネルベースのブローカーは、開発およびテストの目的で使用できますが、実稼働環境での適切なイベント配信の保証は提供しません。デフォルトのブローカーは、デフォルトで **InMemoryChannel** チャネル実装によってサポートされています。

Apache Kafka を使用してネットワークホップを削減する場合は、Apache Kafka の Knative ブローカー実装を使用します。チャネルベースのブローカーが **KafkaChannel** チャネル実装によってサポートされるように設定しないでください。

4.2.2. Apache Kafka の実稼働環境対応の Knative ブローカー実装

実稼働環境対応の Knative Eventing デプロイメントの場合、Red Hat は Apache Kafka に Knative ブローカー実装を使用することを推奨します。ブローカーは、Knative ブローカーの Apache Kafka ネイティブ実装であり、CloudEvents を Kafka インスタンスに直接送信します。

Kafka ブローカーは、イベントを保存してルーティングできるように Kafka とネイティブに統合されています。これにより、他のブローカータイプよりもブローカーとトリガーモデルの Kafka との統合性が向上し、ネットワークホップを削減することができます。Knative ブローカー実装のその他の利点は次のとおりです。

- 少なくとも 1 回の配信保証
- CloudEvents パーティショニング拡張機能に基づくイベントの順序付き配信

- コントロールプレーンの高可用性
- 水平方向にスケーラブルなデータプレーン

Apache Kafka の Knative ブローカー実装は、バイナリーコンテンツモードを使用して、受信した CloudEvent を Kafka レコードとして保存します。これは、CloudEvent のすべての属性と拡張機能が Kafka レコードのヘッダーとしてマップされ、CloudEvent の **data** 仕様が Kafka レコードの値に対応することを意味します。

4.3. ブローカーの作成

Knative は、デフォルトのチャネルベースのブローカー実装を提供します。このチャネルベースのブローカーは、開発およびテストの目的で使用できますが、実稼働環境での適切なイベント配信の保証は提供しません。

クラスター管理者がデフォルトのブローカータイプとして Apache Kafka を使用するように OpenShift Serverless デプロイメントを設定している場合は、デフォルト設定を使用してブローカーを作成すると、Apache Kafka の Knative ブローカーが作成されます。

OpenShift Serverless デプロイメントが Apache Kafka の Kafka ブローカーをデフォルトのブローカータイプとして使用するように設定されていない場合は、以下の手順でデフォルト設定を使用すると、チャネルベースのブローカーが作成されます。

4.3.1. Knative CLI を使用したブローカーの作成

ブローカーはトリガーと組み合わせて、イベントをイベントソースからイベントシンクに配信できます。ブローカーを作成するために Knative (**kn**) CLI を使用すると、YAML ファイルを直接修正するよりも合理的で直感的なユーザーインターフェイスが得られます。**kn broker create** コマンドを使用して、ブローカーを作成できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

- ブローカーを作成します。

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

検証

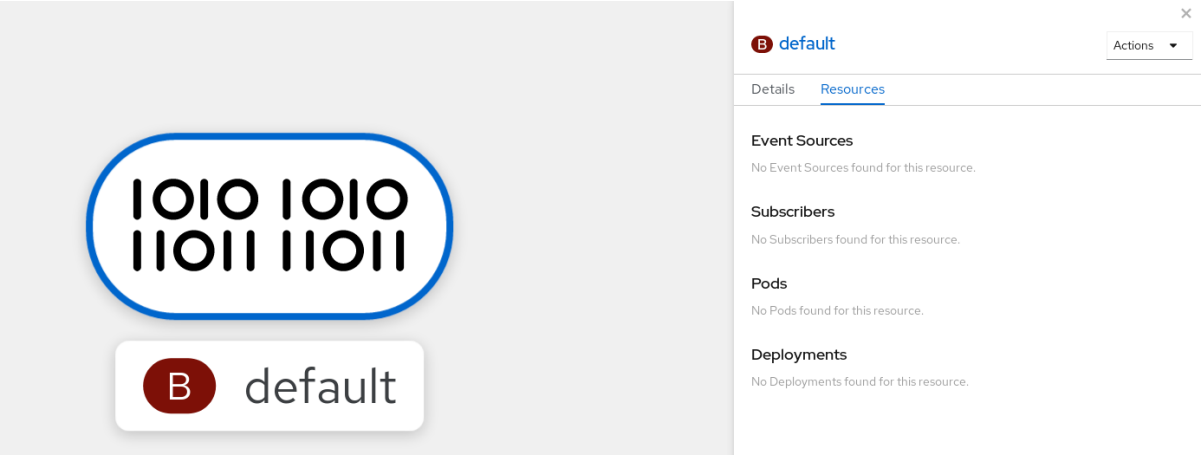
1. **kn** コマンドを使用して、既存のブローカーをリスト表示します。

```
$ kn broker list
```

出力例

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

2. オプション: OpenShift Container Platform Web コンソールを使用している場合は、**Developer** パースペクティブの **Topology** ビューに移動し、ブローカーが存在することを確認できます。



4.3.2. トリガーのアノテーションによるブローカーの作成

ブローカーはトリガーと組み合わせて、イベントをイベントソースからイベントシンクに配信できます。**eventing.knative.dev/injection: enabled** アノテーションを **Trigger** オブジェクトに追加してブローカーを作成できます。

重要

knative-eventing-injection: enabled アノテーションを使用してブローカーを作成する場合は、クラスター管理者パーミッションがなければこのブローカーを削除することができません。クラスター管理者が最初にこのアノテーションを削除せずにブローカーを削除すると、削除後にブローカーが再び作成されます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. **Trigger** オブジェクトを、**eventing.knative.dev/injection: enabled** アノテーションを付けて YAML ファイルとして作成します。

```
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 トリガーがイベントを送信するイベントシンクまたは **サブスクライバー** の詳細を指定します。

2. **Trigger** YAML ファイルを適用します。

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

検証

oc CLI を使用してブローカーが正常に作成されていることを確認するか、または Web コンソールの **Topology** ビューでこれを確認できます。

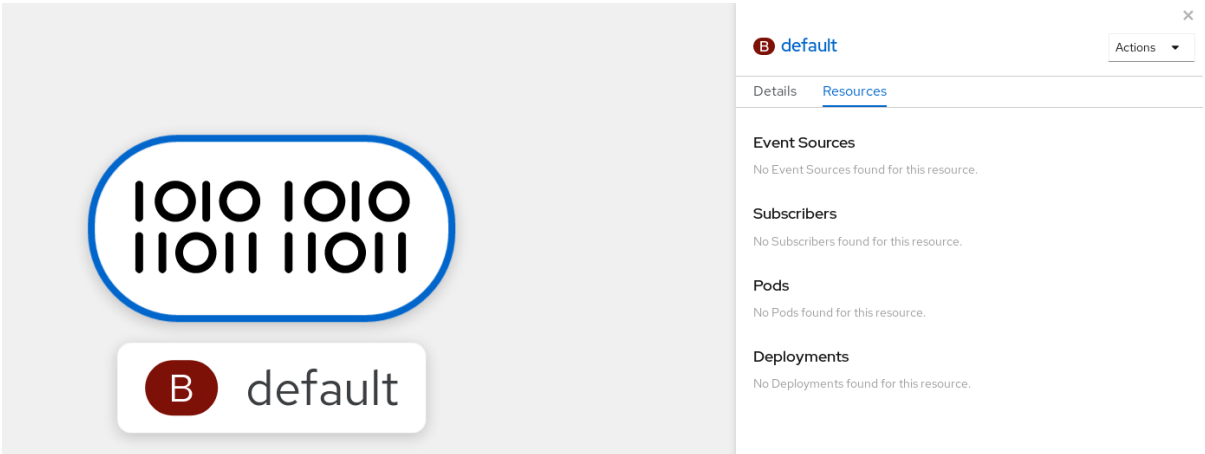
1. 以下の **oc** コマンドを入力してブローカーを取得します。

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

出力例

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

2. オプション: OpenShift Container Platform Web コンソールを使用している場合は、**Developer** パースペクティブの **Topology** ビューに移動し、ブローカーが存在することを確認できます。



4.3.3. namespace へのラベル付けによるブローカーの作成

ブローカーはトリガーと組み合わせて、イベントをイベントソースからイベントシンクに配信できます。所有しているか、または書き込みパーミッションのある namespace にラベルを付けて **default** ブローカーを自動的に作成できます。

注記

この方法を使用して作成されたブローカーは、ラベルを削除すると削除されません。これらは手動で削除する必要があります。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- Red Hat OpenShift Service on AWS または OpenShift Dedicated を使用している場合は、クラスターまたは Dedicated 管理者権限が割り当てられている。

手順

- **eventing.knative.dev/injection=enabled** で namespace にラベルを付ける。

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

検証

oc CLI を使用してブローカーが正常に作成されていることを確認するか、または Web コンソールの **Topology** ビューでこれを確認できます。

1. **oc** コマンドを使用してブローカーを取得します。

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

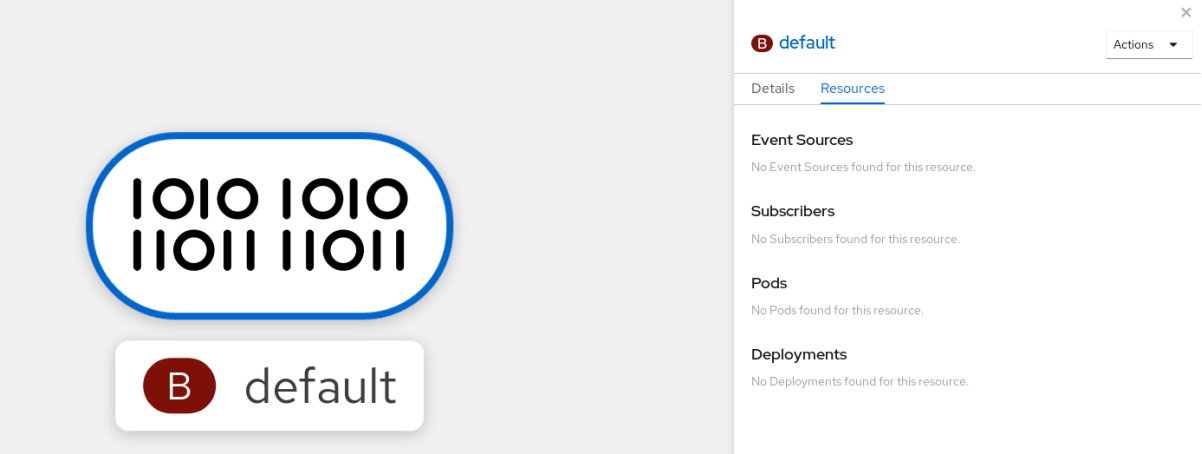
コマンドの例

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

出力例

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

2. オプション: OpenShift Container Platform Web コンソールを使用している場合は、**Developer** パースペクティブの **Topology** ビューに移動し、ブローカーが存在することを確認できます。



4.3.4. 挿入 (injection) によって作成されたブローカーの削除

挿入によりブローカーを作成し、後でそれを削除する必要がある場合は、手動で削除する必要があります。namespace ラベルまたはトリガーアノテーションを使用して作成されたブローカーは、ラベルまたはアノテーションを削除した場合に永続的に削除されません。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. **eventing.knative.dev/injection=enabled** ラベルを namespace から削除します。

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

   アノテーションを削除すると、Knative では削除後にブローカーを再作成できなくなります。

2. 選択された namespace からブローカーを削除します。

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

検証

- **oc** コマンドを使用してブローカーを取得します。

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

コマンドの例

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

出力例

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

4.3.5. Web コンソールを使用してブローカーを作成する

Knative Eventing がクラスターにインストールされた後、Web コンソールを使用してブローカーを作成できます。OpenShift Container Platform Web コンソールを使用すると、ブローカーを作成するための合理的で直感的なユーザーインターフェイスが提供されます。

前提条件

- OpenShift Container Platform Web コンソールにログインしている。
- OpenShift Serverless Operator、Knative Serving、および Knative Eventing がクラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

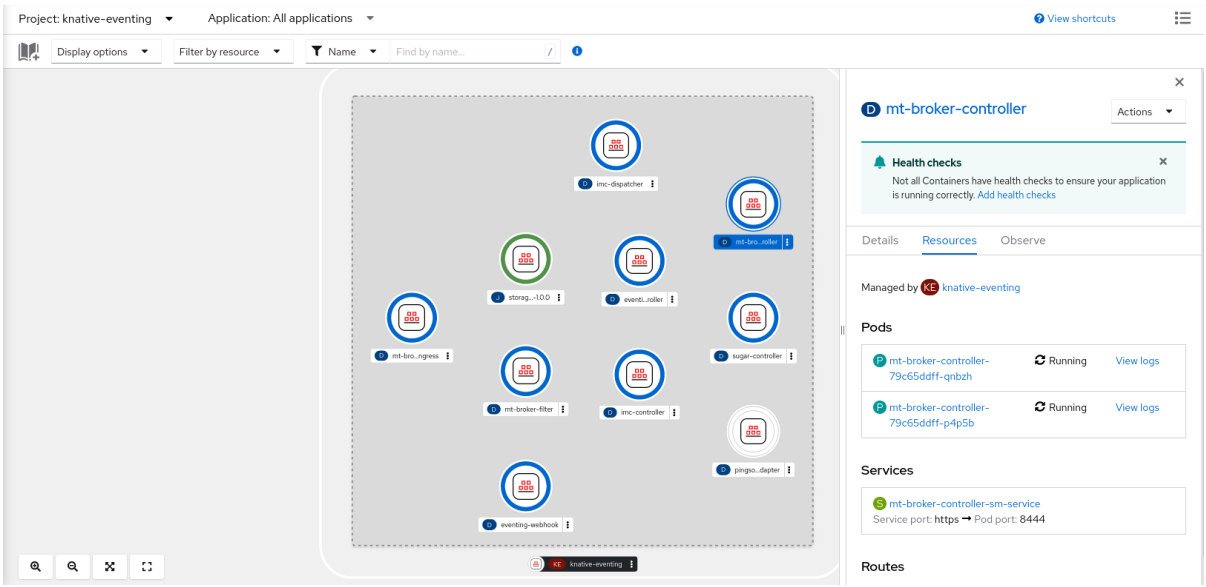
手順

1. **Developer** パースペクティブで、**+Add → Broker** に移動します。**Broker** ページが表示されます。
2. オプション:ブローカーの **Name** を更新します。名前を更新しないと、生成されたブローカーの名前は **default** になります。
3. **Create** をクリックします。

検証

トポロジー ページでブローカーコンポーネントを表示することにより、ブローカーが作成されたことを確認できます。

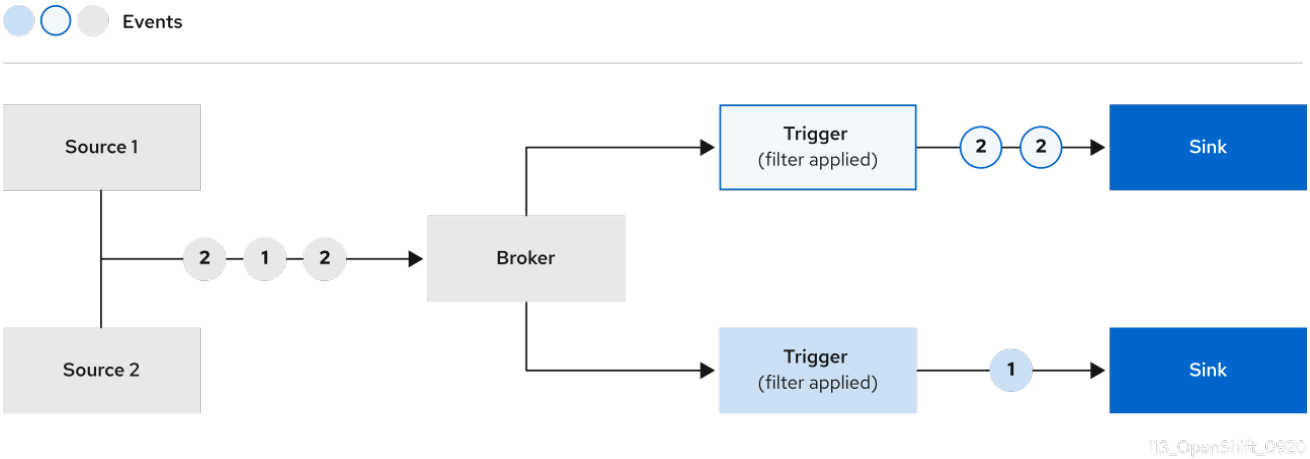
1. **Developer** パースペクティブで、**Topology** に移動します。
2. **mt-broker-ingress**、**mt-broker-filter**、および **mt-broker-controller** コンポーネントを表示します。



4.3.6. Administrator パースペクティブを使用したブローカーの作成

ブローカーはトリガーと組み合わせて、イベントをイベントソースからイベントシンクに配信できます。イベントは、HTTP **POST** リクエストとしてイベントソースからブローカーに送信されます。イベントがブローカーに送信された後に、それらはトリガーを使用して CloudEvent 属性 でフィルターさ

れ、HTTP **POST** リクエストとしてイベントシンクに送信できます。



前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Web コンソールにログインしており、**Administrator** パースペクティブを使用している。
- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Serverless → Eventing** に移動します。
2. **Create** リストで、**Broker** を選択します。**Create Broker** ページに移動します。
3. オプション: ブローカーの YAML 設定を変更します。
4. **Create** をクリックします。

4.3.7. 次のステップ

- イベントがイベントシンクに配信されなかった場合に適用される イベント配信パラメーター を設定します。

4.3.8. 関連情報

- デフォルトブローカークラスの設定
- トリガー
- 開発者パースペクティブを使用してブローカーをシンクに接続する

4.4. デフォルトのブローカーバッキングチャネルの設定

チャネルベースのブローカーを使用している場合は、ブローカーのデフォルトのバッキングチャネルタイプを **InMemoryChannel** または **KafkaChannel** に設定できます。

前提条件

- OpenShift Container Platform に対する管理者権限を持っている。
- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。
- OpenShift (**oc**) CLI がインストールされている。
- Apache Kafka チャネルをデフォルトのバッキングチャネルタイプとして使用する場合は、クラスターに **KnativeKafka** CR もインストールしている。

手順

1. **KnativeEventing** カスタムリソース (CR) を変更して、 **config-br-default-channel** Config Map の設定の詳細を追加します。

```
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** で、変更した設定を追加する Config Map を指定できます。

(2) デフォルトのバッキングチャネルタイプの設定。この例では、クラスターのデフォルトのチャネル実装は **KafkaChannel** です。

(3) ブローカーをサポートする Kafka チャネルのパーティションの数。

(4) ブローカーをサポートする Kafka チャネルのレプリケーションファクター。

2. 更新された **KnativeEventing** CR を適用します。

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

4.5. デフォルトブローカークラスの設定

config-br-defaults Config Map を使用して、Knative Eventing のデフォルトのブローカークラス設定を指定できます。クラスター全体または 1 つ以上の namespace に対して、デフォルトのブローカークラスを指定できます。現在、**MTChannelBasedBroker** および **Kafka** ブローカータイプがサポートされて

います。

前提条件

- OpenShift Container Platform に対する管理者権限を持っている。
- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。
- Apache Kafka の Knative ブローカーをデフォルトのブローカー実装として使用する場合は、クラスターに **KnativeKafka** CR もインストールしている。

手順

- **KnativeEventing** カスタムリソースを変更して、**config-br-defaults** Config Map の設定の詳細を追加します。

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

(1) Knative Eventing のデフォルトのブローカークラス。

(2) **spec.config** で、変更した設定を追加する Config Map を指定できます。

(3) **config-br-defaults** Config Map は、**spec.config** 設定またはブローカークラスを指定しないブローカーのデフォルト設定を指定します。

(4) クラスター全体のデフォルトのブローカークラス設定。この例では、クラスターのデフォルトのブローカークラスの実装は **Kafka** です。

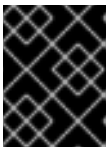
(5) **kafka-broker-config** Config Map は、Kafka ブローカーのデフォルト設定を指定します。関連情報セクションの Apache Kafka 設定用の Knative ブローカーの設定を参照してください。

6 **kafka-broker-config** Config Map が存在する namespace。

7 namespace スコープのデフォルトブローカクラス設定。この例では、**my-namespace** namespace のデフォルトのブローカークラスの実装は **MTChannelBasedBroker** です。複数の namespace に対してデフォルトのブローカークラスの実装を指定できます。

8 **config-br-default-channel** Config Map は、ブローカーのデフォルトのバッキングチャネルを指定します。「関連情報」セクションの「デフォルトのブローカーバッキングチャネルの設定」を参照してください。

9 **config-br-default-channel** Config Map が存在する namespace。



重要

namespace 固有のデフォルトを設定すると、クラスター全体の設定が上書きされます。

4.6. APACHE KAFKA の KNATIVE ブローカー実装

実稼働環境対応の Knative Eventing デプロイメントの場合、Red Hat は Apache Kafka に Knative ブローカー実装を使用することを推奨します。ブローカーは、Knative ブローカーの Apache Kafka ネイティブ実装であり、CloudEvents を Kafka インスタンスに直接送信します。

Kafka ブローカーは、イベントを保存してルーティングできるように Kafka とネイティブに統合されています。これにより、他のブローカータイプよりもブローカーとトリガーモデルの Kafka との統合性が向上し、ネットワークホップを削減することができます。Knative ブローカー実装のその他の利点は次のとおりです。

- 少なくとも 1 回の配信保証
- CloudEvents パーティショニング拡張機能に基づくイベントの順序付き配信
- コントロールプレーンの高可用性
- 水平方向にスケーラブルなデータプレーン

Apache Kafka の Knative ブローカー実装は、バイナリーコンテンツモードを使用して、受信した CloudEvent を Kafka レコードとして保存します。これは、CloudEvent のすべての属性と拡張機能が Kafka レコードのヘッダーとしてマップされ、CloudEvent の **data** 仕様が Kafka レコードの値に対応することを意味します。

4.6.1. デフォルトのブローカータイプとして設定されていない場合の Apache Kafka ブローカーの作成

OpenShift Serverless デプロイメントがデフォルトのブローカータイプとして Kafka ブローカーを使用するように設定されていない場合は、以下の手順のいずれかを使用して、Kafka ベースのブローカーを作成できます。

4.6.1.1. YAML を使用した Apache Kafka ブローカーの作成

YAML ファイルを使用して Knative リソースを作成する場合は、宣言的 API を使用するため、再現性の高い方法でアプリケーションを宣言的に記述できます。YAML を使用して Kafka ブローカーを作成するには、**Broker** オブジェクトを定義する YAML ファイルを作成し、**oc apply** コマンドを使用してそれを適用する必要があります。

前提条件

- OpenShift Serverless Operator、Knative Eventing、および **KnativeKafka** カスタムリソースが OpenShift Container Platform クラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. Kafka ベースのブローカーを YAML ファイルとして作成します。

```
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) ブローカークラス。指定されていないと、ブローカーはクラスター管理者の設定に従ってデフォルトクラスを使用します。Kafka ブローカーを使用するには、この値を **Kafka** にする必要があります。

(2) Apache Kafka の Knative ブローカーのデフォルトの Config Map 。この Config Map は、クラスター管理者がクラスター上で Kafka ブローカー機能を有効にした場合に作成されます。

2. Kafka ベースのブローカー YAML ファイルを適用します。

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

4.6.1.2. 外部で管理される Kafka トピックを使用する Apache Kafka ブローカーの作成

独自の内部トピックの作成を許可せずに Kafka ブローカーを使用する場合は、代わりに外部で管理される Kafka トピックを使用できます。これを実行するには、**kafka.eventing.knative.dev/external.topic** アノテーションを使用する Kafka **Broker** オブジェクトを作成する必要があります。

前提条件

- OpenShift Serverless Operator、Knative Eventing、および **KnativeKafka** カスタムリソースが OpenShift Container Platform クラスターにインストールされている。
- Red Hat AMQ Streams などの Kafka インスタンスにアクセスでき、Kafka トピックを作成している。

- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. Kafka ベースのブローカーを YAML ファイルとして作成します。

```
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) ブローカークラス。指定されていないと、ブローカーはクラスター管理者の設定に従ってデフォルトクラスを使用します。Kafka ブローカーを使用するには、この値を **Kafka** にする必要があります。

(2) 使用する Kafka トピックの名前。

2. Kafka ベースのブローカー YAML ファイルを適用します。

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

4.6.1.3. 分離されたデータプレーンのある Apache Kafka の Knative Broker 実装

重要

分離されたデータプレーンを使用した Apache Kafka の Knative Broker 実装は、テクノロジープレビュー機能としてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

Apache Kafka の Knative Broker 実装には 2 つのプレーンがあります。

コントロールプレーン

Kubernetes API と通信し、カスタムオブジェクトを監視し、データプレーンを管理するコントローラーで設定されます。

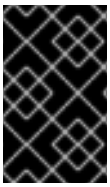
データプレーン

受信イベントをリッスンし、Apache Kafka と通信し、イベントをイベントシンクに送信するコンポーネントのコレクション。Apache Kafka データプレーンの Knative Broker 実装は、イベントが送信される場所です。この実装は、**kafka-broker-receiver** および **kafka-broker-dispatcher** デプロイ

メントで設定されます。

Kafka の Broker クラスを設定する場合、Apache **Kafka** の Knative Broker 実装は共有データプレーンを使用します。つまり、**knative-eventing** namespace の **kafka-broker-receiver** および **kafka-broker-dispatcher** デプロイメントがクラスター内のすべての Apache Kafka Broker に使用されます。

ただし、**KafkaNamespaced** の Broker クラスを設定すると、Apache Kafka ブローカーコントローラーは、ブローカーが存在する namespace ごとに新しいデータプレーンを作成します。このデータプレーンは、その namespace のすべての **KafkaNamespaced** ブローカーによって使用されます。これにより、データプレーンが分離されるため、ユーザーの namespace の **kafka-broker-receiver** および **kafka-broker-dispatcher** デプロイメントは、その namespace のブローカーに対してのみ使用されます。



重要

データプレーンを分離した結果、このセキュリティー機能はより多くのデプロイメントを作成し、より多くのリソースを使用します。このような分離要件がない限り、**Kafka** のクラスで **通常** の Broker を使用します。

4.6.1.4. 分離されたデータプレーンを使用する Apache Kafka の Knative ブローカーの作成

重要

分離されたデータプレーンを使用した Apache Kafka の Knative Broker 実装は、テクノロジープレビュー機能としてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

KafkaNamespaced ブローカーを作成するには、**eventing.knative.dev/broker.class** アノテーションを **KafkaNamespaced** に設定する必要があります。

前提条件

- OpenShift Serverless Operator、Knative Eventing、および **KnativeKafka** カスタムリソースが OpenShift Container Platform クラスターにインストールされている。
- Red Hat AMQ Streams などの Apache Kafka インスタンスにアクセスでき、Kafka トピックを作成している。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. YAML ファイルを使用して Apache Kafka ベースのブローカーを作成します。
   -

```
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) 分離されたデータプレーンで Apache Kafka ブローカーを使用するには、ブローカークラスの値は **KafkaNamespaced** である必要があります。

(2) (3) 参照される **ConfigMap** オブジェクトの **my-config** は、**Broker** オブジェクトと同じ namespace (この場合は **my-namespace**) に存在する必要があります。

2. Apache Kafka ベースのブローカー YAML ファイルを適用します。

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

重要

spec.config の **ConfigMap** オブジェクトは **Broker** オブジェクトと同じ namespace にある必要があります。

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

KafkaNamespaced クラスで最初の **Broker** オブジェクトを作成すると、**kafka-broker-receiver** および **kafka-broker-dispatcher** デプロイメントが namespace に作成されます。その後、同じ namespace 内で **KafkaNamespaced** クラスが含まれる全ブローカーにより、同じデータプレーンが使用されます。**KafkaNamespaced** クラスを持つブローカーが namespace に存在しない場合は、namespace のデータプレーンが削除されます。

4.6.2. Apache Kafka ブローカー設定

Config Map を作成し、Kafka **Broker** オブジェクトでこの ConfigMap を参照することで、レプリケーション係数、ブートストラップサーバー、および Kafka ブローカーのトピックパーティションの数を設定できます。

前提条件

- OpenShift Container Platform でクラスターまたは専用の管理者パーミッションを持っている。
- OpenShift Serverless Operator、Knative Eventing、および **KnativeKafka** カスタムリソース (CR) が OpenShift Container Platform クラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **kafka-broker-config** ConfigMap を変更するか、以下の設定が含まれる独自の ConfigMap を作成します。

```
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. ConfigMap 名。
2. ConfigMap が存在する namespace。
3. Kafka ブローカーのトピックパーティションの数。これは、イベントをブローカーに送信する速度を制御します。パーティションが多い場合には、コンピュートリソースが多く必要です。
4. トピックメッセージのレプリケーション係数。これにより、データ損失を防ぐことができます。レプリケーション係数を増やすには、より多くのコンピュートリソースとストレージが必要になります。
5. ブートストラップサーバーのコンマ区切りリスト。これは、OpenShift Container Platform クラスターの内部または外部にある可能性があり、ブローカーがイベントを受信してイベントを送信する Kafka クラスターのリストです。

重要

default.topic.replication.factor の値は、クラスター内の Kafka ブローカーインスタンスの数以下である必要があります。たとえば、Kafka ブローカーが 1 つしかない場合、**default.topic.replication.factor** の値は **"1"** より大きな値にすることはできません。

Kafka ブローカーの ConfigMap の例

```
apiVersion: v1
```

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

2. ConfigMap を適用します。

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

3. Kafka **Broker** オブジェクトの ConfigMap を指定します。

Broker オブジェクトの例

```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  name: <broker_name> 1
  namespace: <namespace> 2
  annotations:
    eventing.knative.dev/broker.class: Kafka 3
spec:
  config:
    apiVersion: v1
    kind: ConfigMap
    name: <config_map_name> 4
    namespace: <namespace> 5
...
```

1. ブローカー名。
2. ブローカーが存在する namespace。
3. ブローカークラスアノテーション。この例では、ブローカーはクラス値 Kafka を使用する **Kafka** ブローカーです。
4. ConfigMap 名。
5. ConfigMap が存在する namespace。

4. ブローカーを適用します。

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

4.6.3. Apache Kafka の Knative ブローカー実装のセキュリティー設定

Kafka クラスターは、通常、TLS または SASL 認証方法を使用して保護されます。TLS または SASL を使用して、保護された Red Hat AMQ Streams クラスターに対して動作するように Kafka ブローカーまたはチャネルを設定できます。

注記

Red Hat は、SASL と TLS の両方を一緒に有効にすることを推奨します。

4.6.3.1. Apache Kafka ブローカーの TLS 認証の設定

Transport Layer Security (TLS) は、Apache Kafka クライアントおよびサーバーによって、Knative と Kafka 間のトラフィックを暗号化するため、および認証のために使用されます。TLS は、Apache Kafka の Knative ブローカー実装でサポートされている唯一のトラフィック暗号化方式です。

前提条件

- OpenShift Container Platform でクラスターまたは専用の管理者パーミッションを持っている。
- OpenShift Serverless Operator、Knative Eventing、および **KnativeKafka** CR は、OpenShift Container Platform クラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- **.pem** ファイルとして Kafka クラスター CA 証明書が保存されている。
- Kafka クラスタークライアント証明書とキーが **.pem** ファイルとして保存されている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 証明書ファイルを **knative-eventing** namespace にシークレットファイルとして作成します。

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

重要

キー名に **ca.crt**、 **user.crt**、および **user.key** を使用します。これらの値は変更しないでください。

2. **KnativeKafka** CR を編集し、**broker** 仕様にシークレットへの参照を追加します。

```
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 ブローカーの SASL 認証の設定

Simple Authentication and Security Layer(SASL) は、Apache Kafka が認証に使用します。クラスターで SASL 認証を使用する場合、ユーザーは Kafka クラスターと通信するために Knative に認証情報を提供する必要があります。そうしないと、イベントを生成または消費できません。

前提条件

- OpenShift Container Platform でクラスターまたは専用の管理者パーミッションを持っている。
- OpenShift Serverless Operator、Knative Eventing、および **KnativeKafka** CR は、OpenShift Container Platform クラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- Kafka クラスターのユーザー名およびパスワードがある。
- 使用する SASL メカニズムを選択している (例: **PLAIN**、**SCRAM-SHA-256**、または **SCRAM-SHA-512**)。
- TLS が有効になっている場合は、Kafka クラスターの **ca.crt** 証明書ファイルがある。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 証明書ファイルを **knative-eventing** namespace にシークレットファイルとして作成します。

```
$ 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"
```

   - キー名に **ca.crt**、 **password**、および **sasl.mechanism** を使用します。これらの値は変更しないでください。
   - パブリック CA 証明書で SASL を使用する場合は、シークレットの作成時に **ca.crt** 引数ではなく **tls.enabled=true** フラグを使用する必要があります。以下に例を示します。

```
$ 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** CR を編集し、 **broker** 仕様にシークレットへの参照を追加します。

```
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. 関連情報

- Red Hat AMQ Streams のドキュメント
- Kafka での TLS および SASL

4.7. ブローカーの管理

ブローカーを作成した後、Knative (**kn**) CLI コマンドを使用するか、OpenShift Container Platform Web コンソールでブローカーを変更することで、ブローカーを管理できます。

4.7.1. CLI を使用したブローカーの管理

Knative (**kn**) CLI は、既存のブローカーを記述およびリストするために使用できるコマンドを提供します。

4.7.1.1. Knative CLI を使用した既存ブローカーの一覧表示

Knative (**kn**) CLI を使用してブローカーをリスト表示すると、合理的で直感的なユーザーインターフェイスが提供されます。**kn broker list** コマンドを使用し、Knative CLI を使用してクラスター内の既存ブローカーをリスト表示できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。

手順

- 既存ブローカーのリストを表示します。

```
$ kn broker list
```

出力例

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

4.7.1.2. Knative CLI を使用した既存ブローカーの記述

Knative (**kn**) CLI を使用してブローカーを記述すると、合理的で直感的なユーザーインターフェイスが提供されます。**kn broker describe** コマンドを使用し、Knative CLI を使用してクラスター内の既存ブローカーに関する情報を出力できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。

手順

- 既存ブローカーを記述します。

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

デフォルトブローカーを使用したコマンドの例

```
$ kn broker describe default
```

出力例

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

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

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

4.7.2. 開発者パースペクティブを使用してブローカーをシンクに接続する

トリガーを作成することで、OpenShift Container Platform **Developer** パースペクティブでブローカーをイベントシンクに接続できます。

前提条件

- OpenShift Serverless Operator、Knative Serving、および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Web コンソールにログインしており、**Developer** パースペクティブを使用している。

- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- Knative サービスやチャネルなどのシンクを作成しました。
- ブローカーを作成している。

手順

1. **Topology** ビューで、作成したブローカーをポイントします。矢印が表示されます。矢印をブローカーに接続するシンクにドラッグします。この操作により、**Add Trigger** ダイアログボックスが開きます。
2. **Add Trigger** ダイアログボックスで、トリガーの名前を入力し、**Add** をクリックします。

検証

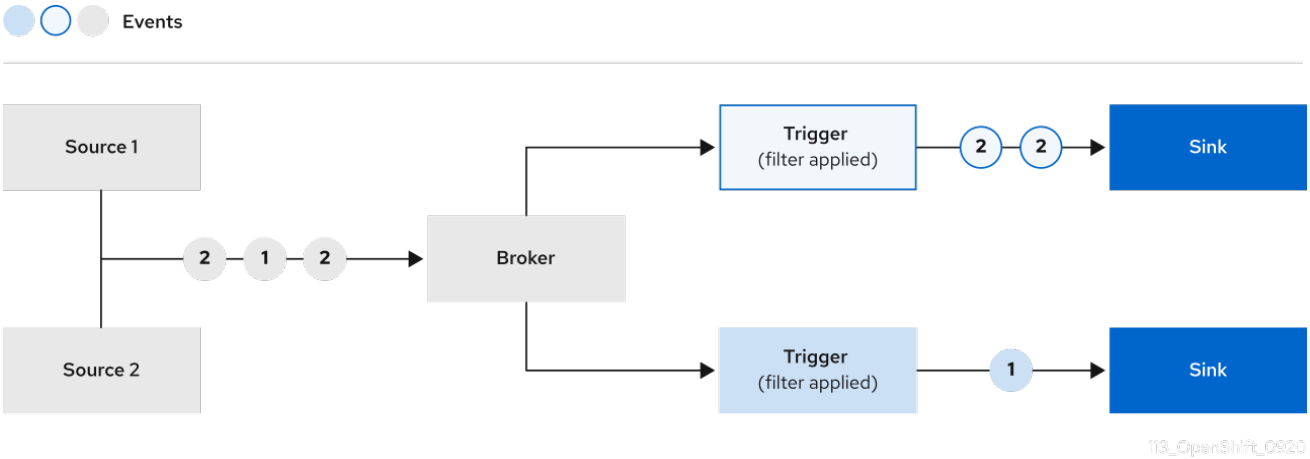
Topology ページを表示すると、ブローカーがシンクに接続されていることを確認できます。

1. **Developer** パースペクティブで、**Topology** に移動します。
2. ブローカーをシンクに接続する線をクリックすると、**Details** パネルでトリガーの詳細が表示されます。

第5章 トリガー

5.1. トリガーの概要

ブローカーはトリガーと組み合わせて、イベントをイベントソースからイベントシンクに配信できます。イベントは、HTTP **POST** リクエストとしてイベントソースからブローカーに送信されます。イベントがブローカーに送信された後に、それらはトリガーを使用して CloudEvent 属性 でフィルターされ、HTTP **POST** リクエストとしてイベントシンクに送信できます。



Apache Kafka の Knative ブローカーを使用している場合は、トリガーからイベントシンクへのイベントの配信順序を設定できます。トリガーのイベント配信順序の設定 を参照してください。

5.1.1. トリガーのイベント配信順序の設定

Kafka ブローカーを使用している場合は、トリガーからイベントシンクへのイベントの配信順序を設定できます。

前提条件

- OpenShift Serverless Operator、Knative Eventing、および Knative Kafka が OpenShift Container Platform クラスターにインストールされている。
- Kafka ブローカーがクラスターで使用可能であり、Kafka ブローカーが作成されている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- OpenShift (**oc**) CLI がインストールされている。

手順

1. **Trigger** オブジェクトを作成または変更し、**kafka.eventing.knative.dev/delivery.order** アノテーションを設定します。

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: <trigger_name>
```

```
  annotations:
    kafka.eventing.knative.dev/delivery.order: ordered
# ...
```

サポートされているコンシューマー配信保証は次のとおりです。

unordered

順序付けられていないコンシューマーは、適切なオフセット管理を維持しながら、メッセージを順序付けずに配信するノンブロッキングコンシューマーです。

ordered

順序付きコンシューマーは、CloudEvent サブスクライバーからの正常な応答を待ってから、パーティションの次のメッセージを配信する、パーティションごとのブロックコンシューマーです。
デフォルトの順序保証は **unordered** です。

2. **Trigger** オブジェクトを適用します。

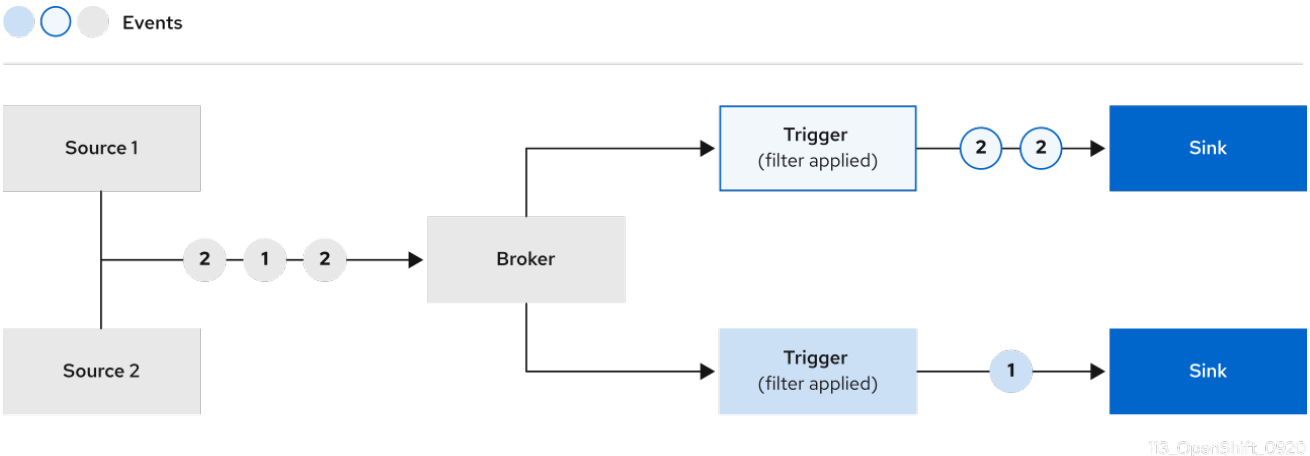
```
$ oc apply -f <filename>
```

5.1.2. 次のステップ

- イベントがイベントシンクに配信されなかった場合に適用される イベント配信パラメーター を設定します。

5.2. トリガーの作成

ブローカーはトリガーと組み合わせて、イベントをイベントソースからイベントシンクに配信できます。イベントは、HTTP **POST** リクエストとしてイベントソースからブローカーに送信されます。イベントがブローカーに送信された後に、それらはトリガーを使用して CloudEvent 属性 でフィルターされ、HTTP **POST** リクエストとしてイベントシンクに送信できます。



5.2.1. Administrator パースペクティブを使用したトリガーの作成

OpenShift Container Platform Web コンソールを使用すると、トリガーを作成するための合理的で直感的なユーザーインターフェイスが提供されます。Knative Eventing がクラスターにインストールされ、ブローカーが作成されると、Web コンソールを使用してトリガーを作成できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Web コンソールにログインしており、**Administrator** パースペクティブを使用している。
- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。
- Knative ブローカーを作成している。
- サブスクライバーとして使用する Knative サービスを作成している。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Serverless → Eventing** に移動します。
2. **Broker** タブで、トリガーを追加するブローカーの Options メニュー  を選択します。
3. リストで **Add Trigger** をクリックします。
4. **Add Trigger** のダイアログボックスで、Trigger の **Subscriber** を選択します。サブスクライバーは、ブローカーからイベントを受信する Knative サービスです。
5. **Add** をクリックします。

5.2.2. 開発者パースペクティブを使用したトリガーの作成

OpenShift Container Platform Web コンソールを使用すると、トリガーを作成するための合理的で直感的なユーザーインターフェイスが提供されます。Knative Eventing がクラスターにインストールされ、ブローカーが作成されると、Web コンソールを使用してトリガーを作成できます。

前提条件

- OpenShift Serverless Operator、Knative Serving、および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Web コンソールにログインしている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- トリガーに接続するために、ブローカーおよび Knative サービスまたは他のイベントシンクを作成している。

手順

1. **Developer** パースペクティブで、**Topology** ページに移動します。

2. トリガーを作成するブローカーにカーソルを合わせ、矢印をドラッグします。**Add Trigger** オプションが表示されます。
3. **Add Trigger** をクリックします。
4. **Subscriber** リストでシンクを選択します。
5. **Add** をクリックします。

検証

- サブスクリプションの作成後に、これを **Topology** ページで表示できます。ここでは、ブローカーをイベントシンクに接続する線として表されます。

トリガーの削除

1. **Developer** パースペクティブで、**Topology** ページに移動します。
2. 削除するトリガーをクリックします。
3. **Actions** コンテキストメニューで、**Delete Trigger** を選択します。

5.2.3. Knative CLI を使用したトリガーの作成

kn trigger create コマンドを使用して、トリガーを作成できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

- トリガーを作成します。

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

または、トリガーを作成し、ブローカー挿入を使用して **default** ブローカーを同時に作成できます。

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

デフォルトで、トリガーはブローカーに送信されたすべてのイベントを、そのブローカーにサブスクライブされるシンクに転送します。トリガーの **--filter** 属性を使用すると、ブローカーからイベントをフィルターできるため、サブスクライバーは定義された基準に基づくイベントのサブセットのみを受け取ることができます。

5.3. コマンドラインからのトリガーの一覧表示

Knative (**kn**) CLI を使用してトリガーをリスト表示すると、合理的で直感的なユーザーインターフェイスが提供されます。

5.3.1. Knative CLI の使用によるトリガーの一覧表示

kn trigger list コマンドを使用して、クラスター内の既存トリガーを一覧表示できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。

手順

1. 利用可能なトリガーのリストを出力します。

```
$ kn trigger list
```

出力例

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

2. オプション: JSON 形式でトリガーの一覧を出力します。

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

5.4. コマンドラインからのトリガーの説明

Knative (**kn**) CLI を使用してトリガーを記述すると、合理的で直感的なユーザーインターフェイスが提供されます。

5.4.1. Knative CLI を使用したトリガーの記述

kn trigger describe コマンドを使用し、Knative CLI を使用してクラスター内の既存トリガーに関する情報を出力できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。
- トリガーを作成している。

手順

- コマンドを入力します。

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

出力例

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

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

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

5.5. トリガーのシンクへの接続

トリガーをシンクに接続して、シンクへの送信前にブローカーからのイベントがフィルターされるようにします。トリガーに接続されているシンクは、**Trigger** オブジェクトのリソース仕様で **subscriber** として設定されます。

Apache Kafka シンクに接続された Trigger オブジェクトの例

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

1. シンクに接続されているトリガーの名前。
2. **KafkaSink** オブジェクトの名前。

5.6. コマンドラインからのトリガーのフィルタリング

Knative (**kn**) CLI を使用してイベントをフィルタリングすると、合理的で直感的なユーザーインターフェイスが提供されます。**kn trigger create** コマンドを適切なフラグとともに使用し、トリガーを使用してイベントをフィルタリングできます。

5.6.1. Knative CLI を使用したトリガーでのイベントのフィルター

以下のトリガーの例では、**type: dev.knative.samples.helloworld** 属性のイベントのみがイベントシンクに送付されます。

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

複数の属性を使用してイベントをフィルターすることもできます。以下の例は、type、source、および extension 属性を使用してイベントをフィルターする方法を示しています。

```
$ 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. コマンドラインからのトリガーの更新

Knative (**kn**) CLI を使用してトリガーを更新すると、合理的で直感的なユーザーインターフェイスが提供されます。

5.7.1. Knative CLI を使用したトリガーの更新

特定のフラグを指定して **kn trigger update** コマンドを使用して、トリガーの属性を更新できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

- トリガーを更新します。

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

  - トリガーを、受信イベントに一致するイベント属性をフィルターするように更新できます。たとえば、**type** 属性を使用します。

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

- トリガーからフィルター属性を削除できます。たとえば、キー **type** を使用してフィルター属性を削除できます。

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

- **--sink** パラメーターを使用して、トリガーのイベントシンクを変更できます。

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

5.8. コマンドラインからのトリガーの削除

Knative (**kn**) CLI を使用してトリガーを削除すると、合理的で直感的なユーザーインターフェイスが提供されます。

5.8.1. Knative CLI を使用したトリガーの削除

kn trigger delete コマンドを使用してトリガーを削除できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

- トリガーを削除します。

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

検証

1. 既存のトリガーをリスト表示します。

```
$ kn trigger list
```

2. トリガーが存在しないことを確認します。

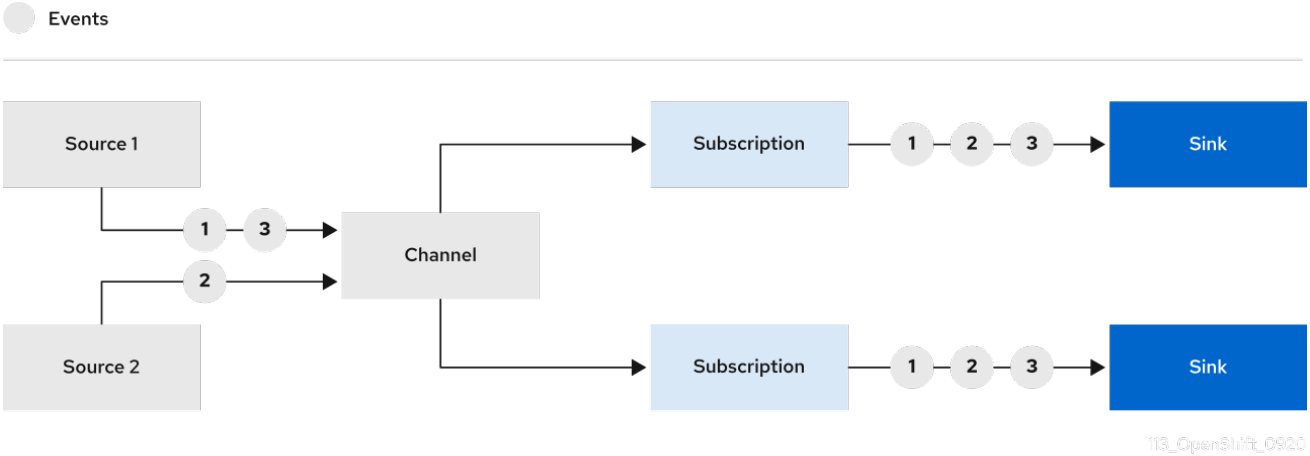
出力例

```
No triggers found.
```

第6章 チャネル

6.1. チャネルおよびサブスクリプション

チャネルは、単一のイベント転送および永続レイヤーを定義するカスタムリソースです。イベントがイベントソースまたは生成側からチャネルに送信された後に、これらのイベントはサブスクリプションを使用して複数の Knative サービスまたは他のシンクに送信できます。

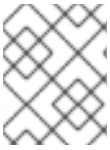


サポートされている **Channel** オブジェクトをインスタンス化することでチャネルを作成し、**Subscription** オブジェクトの **delivery** 仕様を変更して再配信の試行を設定できます。

Channel オブジェクトが作成されると、変更用の受付 Webhook はデフォルトのチャネル実装に基づいて **Channel** オブジェクトの **spec.channelTemplate** プロパティーのセットを追加します。たとえば、**InMemoryChannel** のデフォルト実装の場合、**Channel** オブジェクトは以下のようになります。

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

チャネルコントローラーは、その後に **spec.channelTemplate** 設定に基づいてサポートするチャネルインスタンスを作成します。



注記

spec.channelTemplate プロパティーは作成後に変更できません。それらは、ユーザーではなくデフォルトのチャネルメカニズムで設定されるためです。

このメカニズムが上記の例で使用される場合は、2 つのオブジェクト (汎用バッキングチャネルおよび **InMemoryChannel** チャネルなど) が作成されます。別のデフォルトチャネルの実装を使用している場合、**InMemoryChannel** は実装に固有のものに置き換えられます。たとえば、Apache Kafka の Knative ブローカーでは、**KafkaChannel** チャネルが作成されます。

バッキングチャネルは、サブスクリプションをユーザー作成のチャネルオブジェクトにコピーし、ユーザー作成チャネルオブジェクトのステータスを、バッキングチャネルのステータスを反映するように設定します。

6.1.1. チャネルの実装タイプ

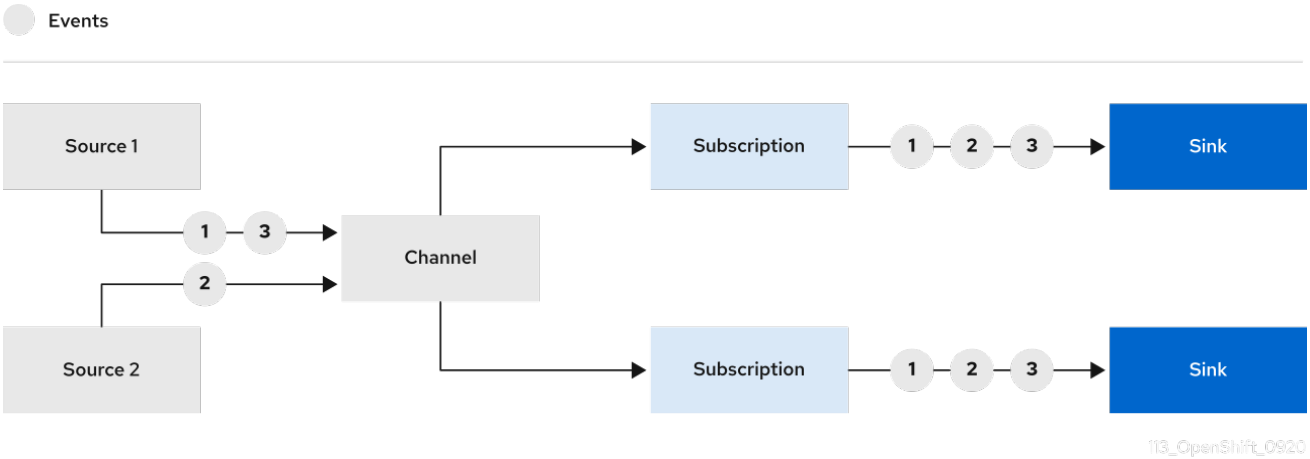
OpenShift Serverless は、**InMemoryChannel** および **KafkaChannel** チャネルの実装をサポートしています。**InMemoryChannel** チャネルは制限があるため、開発用途にのみ使用することを推奨します。実稼働環境では **KafkaChannel** チャネルを使用できます。

以下は、**InMemoryChannel** タイプのチャネルの制限です。

- イベントの永続性は利用できません。Pod がダウンすると、その Pod のイベントが失われます。
- **InMemoryChannel** チャネルはイベントの順序を実装しないため、チャネルで同時に受信される 2 つのイベントはいずれの順序でもサブスクライバーに配信できます。
- サブスクライバーがイベントを拒否する場合、再配信はデフォルトで試行されません。**Subscription** オブジェクトの **delivery** 仕様を変更することで、再配信の試行を設定できます。

6.2. チャネルの作成

チャネルは、単一のイベント転送および永続レイヤーを定義するカスタムリソースです。イベントがイベントソースまたは生成側からチャネルに送信された後に、これらのイベントはサブスクリプションを使用して複数の Knative サービスまたは他のシンクに送信できます。



サポートされている **Channel** オブジェクトをインスタンス化することでチャネルを作成し、**Subscription** オブジェクトの **delivery** 仕様を変更して再配信の試行を設定できます。

6.2.1. Administrator パースペクティブを使用したチャネルの作成

Knative Eventing がクラスターにインストールされると、Administrator パースペクティブを使用してチャネルを作成できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。

- Web コンソールにログインしており、**Administrator** パースペクティブを使用している。
- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Serverless → Eventing** に移動します。
2. **Create** リストで、**Channel** を選択します。**Channel** ページに移動します。
3. **タイプ** リストで、作成する **Channel** オブジェクトのタイプを選択します。

注記

現時点で、**InMemoryChannel** チャネルオブジェクトのみがデフォルトでサポートされます。Apache Kafka の Knative チャネルは、OpenShift Serverless に Apache Kafka の Knative ブローカー実装をインストールしている場合に使用できます。

4. **Create** をクリックします。

6.2.2. 開発者パースペクティブを使用したチャネルの作成

OpenShift Container Platform Web コンソールを使用すると、チャネルを作成するための合理的で直感的なユーザーインターフェイスが提供されます。Knative Eventing がクラスターにインストールされると、Web コンソールを使用してチャネルを作成できます。

前提条件

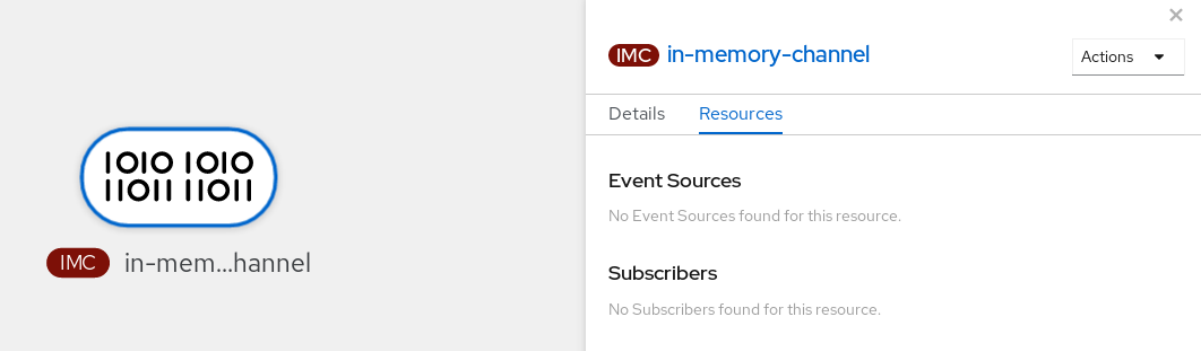
- OpenShift Container Platform Web コンソールにログインしている。
- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. **Developer** パースペクティブで、**+Add → Channel** に移動します。
2. **タイプ** リストで、作成する **Channel** オブジェクトのタイプを選択します。
3. **Create** をクリックします。

検証

- **Topology** ページに移動して、チャネルが存在することを確認します。



6.2.3. Knative CLI を使用したチャネルの作成

チャネルを作成するために Knative (**kn**) CLI を使用すると、YAML ファイルを直接修正するよりも合理的で直感的なユーザーインターフェイスが得られます。**kn channel create** コマンドを使用してチャネルを作成できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

- チャネルを作成します。

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

チャネルタイプはオプションですが、指定する場合は、**Group:Version:Kind** の形式で指定する必要があります。たとえば、**InMemoryChannel** オブジェクトを作成できます。

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

出力例

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

検証

- チャネルが存在することを確認するには、既存のチャネルをリスト表示し、出力を検査します。

```
$ kn channel list
```

出力例

```
kn channel list
```

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

チャネルの削除

- チャネルを削除します。

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

6.2.4. YAML を使用したデフォルト実装チャネルの作成

YAML ファイルを使用して Knative リソースを作成する場合は、宣言的 API を使用するため、再現性の高い方法でチャネルを宣言的に記述できます。YAML を使用して Serverless チャネルを作成するには、**Channel** オブジェクトを定義する YAML ファイルを作成し、**oc apply** コマンドを使用してそれを適用する必要があります。

前提条件

- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. **Channel** オブジェクトを YAML ファイルとして作成します。

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

2. YAML ファイルを適用します。

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

6.2.5. YAML を使用した Apache Kafka のチャネルの作成

YAML ファイルを使用して Knative リソースを作成する場合は、宣言的 API を使用するため、再現性の高い方法でチャネルを宣言的に記述できます。Kafka チャネルを作成することで、Kafka トピックに裏打ちされた Knative Eventing チャネルを作成できます。YAML を使用して Kafka チャネルを作成するには、**KafkaChannel** オブジェクトを定義する YAML ファイルを作成し、**oc apply** コマンドを使用してそれを適用する必要があります。

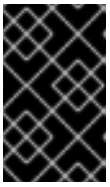
前提条件

- OpenShift Serverless Operator、Knative Eventing、および **KnativeKafka** カスタムリソースが OpenShift Container Platform クラスターにインストールされている。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. **KafkaChannel** オブジェクトを YAML ファイルとして作成します。

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



重要

OpenShift Serverless 上の **KafkaChannel** オブジェクトの API の **v1beta1** バージョンのみがサポートされます。非推奨となった **v1alpha1** バージョンの API は使用しないでください。

2. **KafkaChannel** YAML ファイルを適用します。

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

6.2.6. 次のステップ

- チャネルの作成後に、チャネルをシンクに 接続して、シンクがイベント を受信できるようにします。
- イベントがイベントシンクに配信されなかった場合に適用される イベント配信パラメーター を設定します。

6.3. チャネルのシンクへの接続

イベントソースまたはプロデューサーからチャネルに送信されたイベントは、**サブスクリプション** を使用して 1 つ以上のシンクに転送できます。サブスクリプションを作成するには、チャネルと、そのチャネルに送信されたイベントを消費するシンク (**subscriber** とも呼ばれる) を指定する **Subscription** オブジェクトを設定します。

6.3.1. 開発者パースペクティブを使用したサブスクリプションの作成

チャネルとイベントシンクを作成したら、サブスクリプションを作成してイベント配信を有効できます。OpenShift Container Platform Web コンソールを使用すると、サブスクリプションを作成するための合理的で直感的なユーザーインターフェイスが提供されます。

前提条件

- OpenShift Serverless Operator、Knative Serving、および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Web コンソールにログインしている。
- Knative サービスおよびチャネルなどのイベントシンクを作成している。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

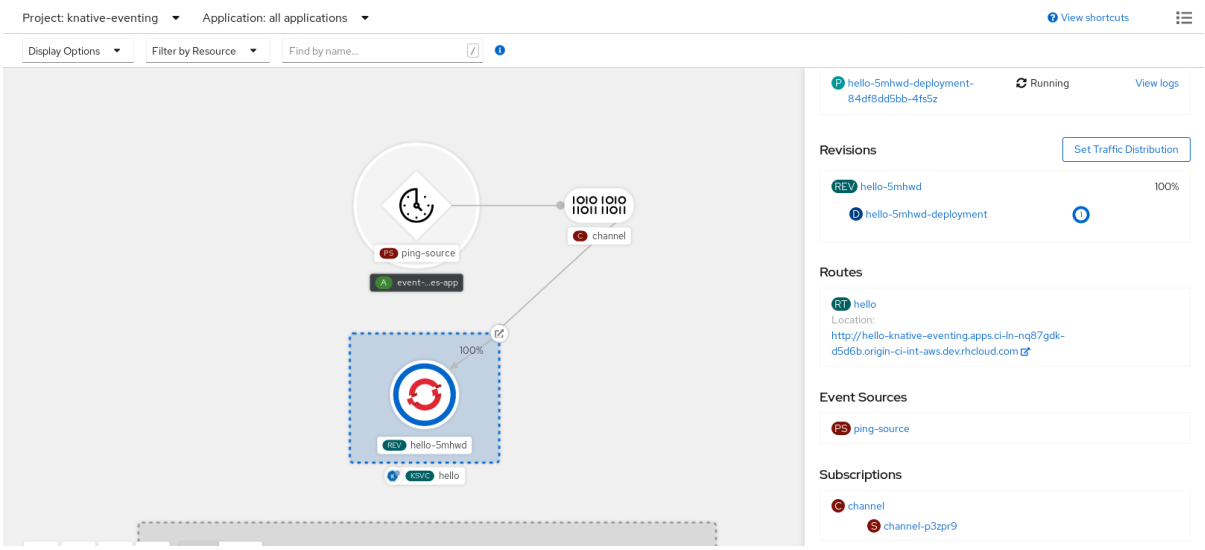
1. **Developer** パースペクティブで、**Topology** ページに移動します。
2. 以下の方法のいずれかを使用してサブスクリプションを作成します。
   a. サブスクリプションを作成するチャネルにカーソルを合わせ、矢印をドラッグします。**Add Subscription** オプションが表示されます。

      i. **Subscriber** リストでシンクを選択します。
      ii. **Add** をクリックします。

   b. このサービスが、チャネルと同じ namespace またはプロジェクトにある **Topology** ビューで利用可能な場合は、サブスクリプションを作成するチャネルをクリックし、矢印をサービスに直接ドラッグして、チャネルからそのサービスにサブスクリプションを即時に作成します。

検証

- サブスクリプションの作成後に、これを **Topology** ビューでチャネルをサービスに接続する行として表示できます。



6.3.2. YAML を使用したサブスクリプションの作成

チャネルとイベントシンクを作成したら、サブスクリプションを作成してイベント配信を有効できます。YAML ファイルを使用して Knative リソースを作成する場合は、宣言的 API を使用するため、再現性の高い方法でサブスクリプションを宣言的に記述できます。YAML を使用してサブスクリプションを作成するには、**Subscription** オブジェクトを定義する YAML ファイルを作成し、**oc apply** コマンドを使用してそれを適用する必要があります。

前提条件

- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

- **Subscription** オブジェクトを作成します。
  - YAML ファイルを作成し、以下のサンプルコードをこれにコピーします。

```
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. サブスクリプションの名前。
2. サブスクリプションが接続するチャネルの設定。
3. イベント配信の設定。これは、サブスクリプションに対してサブスクライバーに配信できないイベントに何が発生するかについて示します。これが設定されると、使用できないイベントが **deadLetterSink** に送信されます。イベントがドロップされると、イベントの再配信は試行されず、エラーのログがシステムに記録されます。**deadLetterSink** 値は Destination である必要があります。
4. サブスクライバーの設定。これは、イベントがチャネルから送信されるイベントシンクです。

- YAML ファイルを適用します。

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

6.3.3. Knative CLI を使用したサブスクリプションの作成

チャネルとイベントシンクを作成したら、サブスクリプションを作成してイベント配信を有効できます。サブスクリプションを作成するために Knative (**kn**) CLI を使用すると、YAML ファイルを直接修正するよりも合理的で直感的なユーザーインターフェイスが得られます。**kn subscription create** コマンドを適切なフラグとともに使用して、サブスクリプションを作成できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

- サブスクリプションを作成し、シンクをチャネルに接続します。

```
$ 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** は、処理する必要のあるクラウドイベントのソースを指定します。チャネル名を指定する必要があります。**Channel** カスタムリソースでサポートされるデフォルトの **InMemoryChannel** チャネルを使用しない場合は、チャネル名に指定されたチャネルタイプの **<group:version:kind>** の接頭辞を付ける必要があります。たとえば、これは Kafka 対応チャネルの **messaging.knative.dev:v1beta1:KafkaChannel** のようになります。

2. **--sink** は、イベントが配信されるターゲット宛先を指定します。デフォルトで、**<sink_name>** は、サブスクリプションと同じ namespace でこの名前の Knative サービスとして解釈されます。以下の接頭辞のいずれかを使用して、シンクのタイプを指定できます。

   ksvc
   : Knative サービス

   channel
   : 宛先として使用する必要のあるチャネル。ここで参照できるのは、デフォルトのチャネルタイプのみです。

   broker
   : Eventing ブローカー。

3. オプション: **--sink-dead-letter** は、イベントが配信に失敗する場合にイベントを送信するシンクを指定するために使用できるオプションのフラグです。詳細は、OpenShift Serverless の **Event 配信**についてのドキュメントを参照してください。

コマンドの例

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

出力例

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

検証

- サブスクリプションを使用してチャネルがイベントシンクまたは **サブスクライバー** に接続されていることを確認するには、既存のサブスクリプションをリスト表示し、出力を検査します。

```
$ kn subscription list
```

出力例

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

サブスクリプションの削除

- サブスクリプションを削除します。

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

6.3.4. Administrator パースペクティブを使用したサブスクリプションの作成

チャネルとイベントシンク (**subscriber** とも呼ばれます) を作成したら、サブスクリプションを作成してイベント配信を有効にできます。サブスクリプションは、イベントを配信するチャネルとサブスクライバーを指定する **Subscription** オブジェクトを設定することによって作成されます。障害の処理方法など、サブスクライバー固有のオプションを指定することもできます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Web コンソールにログインしており、**Administrator** パースペクティブを使用している。
- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。
- Knative チャネルを作成している。
- サブスクライバーとして使用する Knative サービスを作成している。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Serverless → Eventing** に移動します。
2. **Channel** タブで、サブスクリプションを追加するチャネルの Options メニュー を選択します。
3. リストで **Add Subscription** をクリックします。
4. **Add Subscription** のダイアログボックスで、サブスクリプションの **Subscriber** を選択します。サブスクライバーは、チャネルからイベントを受信する Knative サービスです。
5. **Add** をクリックします。

6.3.5. 次のステップ

- イベントがイベントシンクに配信されなかった場合に適用される イベント配信パラメーター を設定します。

6.4. デフォルトのチャネル実装

default-ch-webhook Config Map を使用して、Knative Eventing のデフォルトのチャネル実装を指定できます。クラスター全体または 1 つ以上の namespace に対して、デフォルトのチャネルの実装を指定できます。現在、**InMemoryChannel** および **KafkaChannel** チャネルタイプがサポートされています。

6.4.1. デフォルトチャネル実装の設定

前提条件

- OpenShift Container Platform に対する管理者権限を持っている。

- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。
- Apache Kafka の Knative チャネルをデフォルトのチャネル実装として使用する場合は、クラスターに **KnativeKafka** CR もインストールする必要があります。

手順

- **KnativeEventing** カスタムリソースを変更して、**default-ch-webhook** Config Map の設定の詳細を追加します。

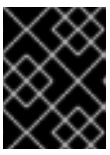
```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  config: (1)
    default-ch-webhook: (2)
      default-ch-config: |
        clusterDefault: (3)
          apiVersion: messaging.knative.dev/v1
          kind: InMemoryChannel
          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** で、変更した設定を追加する Config Map を指定できます。

(2) **default-ch-webhook** Config Map は、クラスターまたは 1 つ以上の namespace のデフォルトチャネルの実装を指定するために使用できます。

(3) クラスター全体のデフォルトのチャネルタイプの設定。この例では、クラスターのデフォルトのチャネル実装は **InMemoryChannel** です。

(4) namespace スコープのデフォルトのチャネルタイプの設定。この例では、**my-namespace** namespace のデフォルトのチャネル実装は **KafkaChannel** です。



重要

namespace 固有のデフォルトを設定すると、クラスター全体の設定が上書きされます。

6.5. チャネルのセキュリティー設定

6.5.1. Apache Kafka の Knative チャネルの TLS 認証設定

Transport Layer Security (TLS) は、Apache Kafka クライアントおよびサーバーによって、Knative と Kafka 間のトラフィックを暗号化するため、および認証のために使用されます。TLS は、Apache Kafka の Knative ブローカー実装でサポートされている唯一のトラフィック暗号化方式です。

前提条件

- OpenShift Container Platform でクラスターまたは専用の管理者パーミッションを持っている。
- OpenShift Serverless Operator、Knative Eventing、および **KnativeKafka** CR は、OpenShift Container Platform クラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- **.pem** ファイルとして Kafka クラスター CA 証明書が保存されている。
- Kafka クラスタークライアント証明書とキーが **.pem** ファイルとして保存されている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 選択された namespace にシークレットとして証明書ファイルを作成します。

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

重要

キー名に **ca.crt**、 **user.crt**、および **user.key** を使用します。これらの値は変更しないでください。

2. **KnativeKafka** カスタムリソースの編集を開始します。

```
$ oc edit knativekafka
```

3. シークレットおよびシークレットの namespace を参照します。

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

```
    enabled: true
  source:
    enabled: true
```

注記

ブートストラップサーバーで一致するポートを指定するようにしてください。

以下に例を示します。

```
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 Kafka の Knative チャネルの SASL 認証設定

Simple Authentication and Security Layer(SASL) は、Apache Kafka が認証に使用します。クラスターで SASL 認証を使用する場合、ユーザーは Kafka クラスターと通信するために Knative に認証情報を提供する必要があります。そうしないと、イベントを生成または消費できません。

前提条件

- OpenShift Container Platform でクラスターまたは専用の管理者パーミッションを持っている。
- OpenShift Serverless Operator、Knative Eventing、および **KnativeKafka** CR は、OpenShift Container Platform クラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- Kafka クラスターのユーザー名およびパスワードがある。
- 使用する SASL メカニズムを選択している (例: **PLAIN**、**SCRAM-SHA-256**、または **SCRAM-SHA-512**)。
- TLS が有効になっている場合は、Kafka クラスターの **ca.crt** 証明書ファイルがある。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 選択された namespace にシークレットとして証明書ファイルを作成します。

```
$ 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"
```

- キー名に **ca.crt**、 **password**、および **sasl.mechanism** を使用します。これらの値は変更しないでください。
- パブリック CA 証明書で SASL を使用する場合は、シークレットの作成時に **ca.crt** 引数ではなく **tls.enabled=true** フラグを使用する必要があります。以下に例を示します。

```
$ 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** カスタムリソースの編集を開始します。

```
$ oc edit knativekafka
```

3. シークレットおよびシークレットの namespace を参照します。

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

注記

ブートストラップサーバーで一致するポートを指定するようにしてください。

以下に例を示します。

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

```
    bootstrapServers: eventing-kafka-bootstrap.kafka.svc:9093
    enabled: true
  source:
    enabled: true
```

第7章 サブスクリプション

7.1. サブスクリプションの作成

チャネルとイベントシンクを作成したら、サブスクリプションを作成してイベント配信を有効できます。サブスクリプションは、イベントを配信するチャネルとシンク (**サブスクライバー**とも呼ばれます) を指定する **Subscription** オブジェクトを設定することによって作成されます。

7.1.1. Administrator パースペクティブを使用したサブスクリプションの作成

チャネルとイベントシンク (**subscriber** とも呼ばれます) を作成したら、サブスクリプションを作成してイベント配信を有効にできます。サブスクリプションは、イベントを配信するチャネルとサブスクライバーを指定する **Subscription** オブジェクトを設定することによって作成されます。障害の処理方法など、サブスクライバー固有のオプションを指定することもできます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Web コンソールにログインしており、**Administrator** パースペクティブを使用している。
- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。
- Knative チャネルを作成している。
- サブスクライバーとして使用する Knative サービスを作成している。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Serverless → Eventing** に移動します。
2. **Channel** タブで、サブスクリプションを追加するチャネルの Options メニュー を選択します。
3. リストで **Add Subscription** をクリックします。
4. **Add Subscription** のダイアログボックスで、サブスクリプションの **Subscriber** を選択します。サブスクライバーは、チャネルからイベントを受信する Knative サービスです。
5. **Add** をクリックします。

7.1.2. 開発者パースペクティブを使用したサブスクリプションの作成

チャネルとイベントシンクを作成したら、サブスクリプションを作成してイベント配信を有効できます。OpenShift Container Platform Web コンソールを使用すると、サブスクリプションを作成するための合理的で直感的なユーザーインターフェイスが提供されます。

前提条件

- OpenShift Serverless Operator、Knative Serving、および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Web コンソールにログインしている。
- Knative サービスおよびチャネルなどのイベントシンクを作成している。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

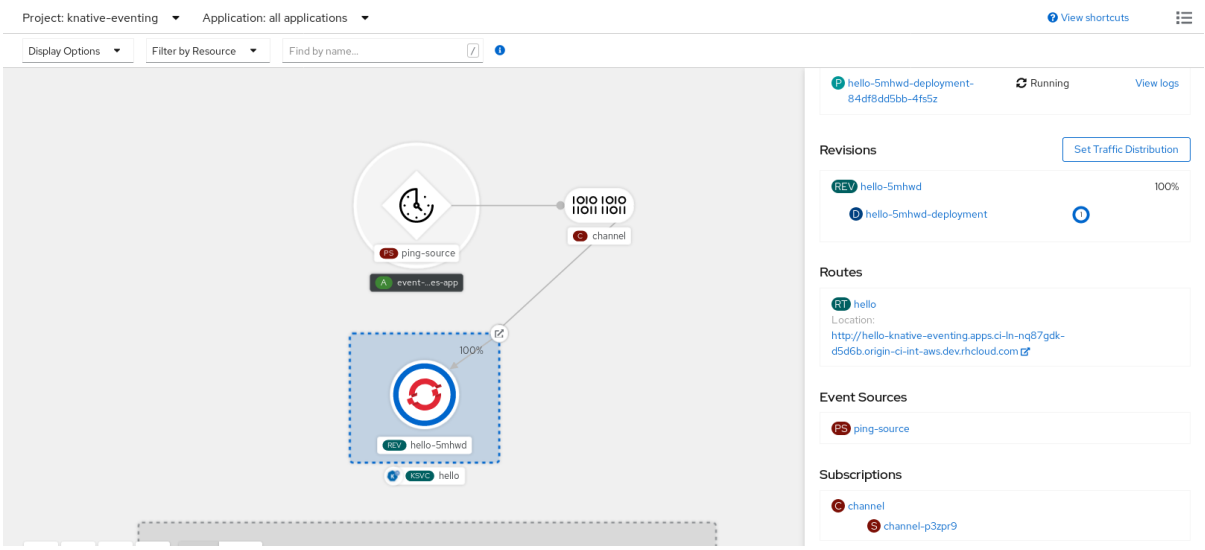
1. **Developer** パースペクティブで、**Topology** ページに移動します。
2. 以下の方法のいずれかを使用してサブスクリプションを作成します。
   a. サブスクリプションを作成するチャネルにカーソルを合わせ、矢印をドラッグします。**Add Subscription** オプションが表示されます。

      i. **Subscriber** リストでシンクを選択します。
      ii. **Add** をクリックします。

   b. このサービスが、チャネルと同じ namespace またはプロジェクトにある **Topology** ビューで利用可能な場合は、サブスクリプションを作成するチャネルをクリックし、矢印をサービスに直接ドラッグして、チャネルからそのサービスにサブスクリプションを即時に作成します。

検証

- サブスクリプションの作成後に、これを **Topology** ビューでチャネルをサービスに接続する行として表示できます。



7.1.3. YAML を使用したサブスクリプションの作成

チャネルとイベントシンクを作成したら、サブスクリプションを作成してイベント配信を有効できます。YAML ファイルを使用して Knative リソースを作成する場合は、宣言的 API を使用するため、再現性の高い方法でサブスクリプションを宣言的に記述できます。YAML を使用してサブスクリプションを作成するには、**Subscription** オブジェクトを定義する YAML ファイルを作成し、**oc apply** コマンドを使用してそれを適用する必要があります。

前提条件

- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

- **Subscription** オブジェクトを作成します。
  - YAML ファイルを作成し、以下のサンプルコードをこれにコピーします。

```
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. サブスクリプションの名前。
2. サブスクリプションが接続するチャネルの設定。
3. イベント配信の設定。これは、サブスクリプションに対してサブスクライバーに配信できないイベントに何が発生するかについて示します。これが設定されると、使用できないイベントが **deadLetterSink** に送信されます。イベントがドロップされると、イベントの再配信は試行されず、エラーのログがシステムに記録されます。**deadLetterSink** 値は Destination である必要があります。
4. サブスクライバーの設定。これは、イベントがチャネルから送信されるイベントシンクです。

- YAML ファイルを適用します。

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

7.1.4. Knative CLI を使用したサブスクリプションの作成

チャネルとイベントシンクを作成したら、サブスクリプションを作成してイベント配信を有効できます。サブスクリプションを作成するために Knative (**kn**) CLI を使用すると、YAML ファイルを直接修正するよりも合理的で直感的なユーザーインターフェイスが得られます。**kn subscription create** コマンドを適切なフラグとともに使用して、サブスクリプションを作成できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

- サブスクリプションを作成し、シンクをチャネルに接続します。

```
$ 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** は、処理する必要のあるクラウドイベントのソースを指定します。チャネル名を指定する必要があります。**Channel** カスタムリソースでサポートされるデフォルトの **InMemoryChannel** チャネルを使用しない場合は、チャネル名に指定されたチャネルタイプの **<group:version:kind>** の接頭辞を付ける必要があります。たとえば、これは Kafka 対応チャネルの **messaging.knative.dev:v1beta1:KafkaChannel** のようになります。

2. **--sink** は、イベントが配信されるターゲット宛先を指定します。デフォルトで、**<sink_name>** は、サブスクリプションと同じ namespace でこの名前の Knative サービスとして解釈されます。以下の接頭辞のいずれかを使用して、シンクのタイプを指定できます。

   ksvc
   : Knative サービス

   channel
   : 宛先として使用する必要のあるチャネル。ここで参照できるのは、デフォルトのチャネルタイプのみです。

   broker
   : Eventing ブローカー。

3. オプション: **--sink-dead-letter** は、イベントが配信に失敗する場合にイベントを送信するシンクを指定するために使用できるオプションのフラグです。詳細は、OpenShift Serverless の **Event 配信**についてのドキュメントを参照してください。

   コマンドの例

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

   出力例

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

検証

- サブスクリプションを使用してチャネルがイベントシンクまたは **サブスクライバー** に接続されていることを確認するには、既存のサブスクリプションをリスト表示し、出力を検査します。

  ```
  $ kn subscription list
  ```

  出力例

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

サブスクリプションの削除

- サブスクリプションを削除します。

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

7.1.5. 次のステップ

- イベントがイベントシンクに配信されなかった場合に適用される イベント配信パラメーター を設定します。

7.2. サブスクリプションの管理

7.2.1. Knative CLI を使用したサブスクリプションの記述

kn subscription describe コマンドを使用し、Knative (**kn**) CLI を使用して、端末のサブスクリプションに関する情報を出力できます。サブスクリプションを記述するために Knative CLI を使用すると、YAML ファイルを直接表示するよりも合理的で直感的なユーザーインターフェイスが得られます。

前提条件

- Knative (**kn**) CLI がインストールされている。
- クラスターにサブスクリプションを作成している。

手順

- サブスクリプションを記述します。

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

出力例

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

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

7.2.2. Knative CLI を使用したサブスクリプションの一覧表示

kn subscription list コマンドを使用し、Knative (**kn**) CLI を使用してクラスター内の既存サブスクリプションをリスト表示できます。Knative CLI を使用してサブスクリプションをリスト表示すると、合理的で直感的なユーザーインターフェイスが提供されます。

前提条件

- Knative (**kn**) CLI がインストールされている。

手順

- クラスターのサブスクリプションをリスト表示します。

```
$ kn subscription list
```

出力例

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

7.2.3. Knative CLI を使用したサブスクリプションの更新

kn subscription update コマンドや適切なフラグを使用し、Knative (**kn**) CLI を使用してサブスクリプションを端末から更新できます。サブスクリプションを更新するために Knative CLI を使用すると、YAML ファイルを直接更新するよりも合理的で直感的なユーザーインターフェイスが得られます。

前提条件

- Knative (**kn**) CLI がインストールされている。
- サブスクリプションを作成している。

手順

- サブスクリプションを更新します。

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

1 **--sink** は、イベントが配信される、更新されたターゲット宛先を指定します。以下の接頭辞のいずれかを使用して、シンクのタイプを指定できます。

ksvc
: Knative サービス

channel
: 宛先として使用する必要のあるチャネル。ここで参照できるのは、デフォルトのチャネルタイプのみです。

broker
: Eventing ブローカー。

2 オプション: **--sink-dead-letter** は、イベントが配信に失敗する場合にイベントを送信するシンクを指定するために使用できるオプションのフラグです。詳細は、OpenShift Serverless の **Event 配信**についてのドキュメントを参照してください。

コマンドの例

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

第8章 イベント配信

イベントがイベントシンクに配信されなかった場合に適用されるイベント配信パラメーターを設定できます。さまざまなチャネルとブローカーのタイプには、イベント配信のために従う独自の動作パターンがあります。

デッドレターシンクを含むイベント配信パラメーターを設定すると、イベントシンクへの配信に失敗したすべてのイベントが再試行されるようになります。それ以外の場合は、未配信のイベントが破棄される。

重要

イベントが、Apache Kafka のチャネルまたはブローカーレシーバーに正常に配信される場合、受信側は **202** ステータスコードで応答します。つまり、このイベントは Kafka トピック内に安全に保存され、失われることはありません。受信側がその他のステータスコードを返す場合は、イベントは安全に保存されず、ユーザーがこの問題を解決するために手順を実行する必要があります。

8.1. 設定可能なイベント配信パラメーター

以下のパラメーターはイベント配信用に設定できます。

dead letter sink

deadLetterSink 配信パラメーターを設定して、イベントが配信に失敗した場合にこれを指定されたイベントシンクに保存することができます。デッドレターシンクに格納されていない未配信のイベントは破棄されます。デッドレターシンクは、Knative サービス、Kubernetes サービス、または URI など、Knative Eventing シンクコントラクトに準拠する任意のアドレス指定可能なオブジェクトです。

retries

retry 配信パラメーターを整数値で設定することで、イベントが dead letter sink に送信される前に配信を再試行する必要のある最小回数を設定できます。

back off delay

backoffDelay 配信パラメーターを設定し、失敗後にイベント配信が再試行される前の遅延の時間を指定できます。**backoffDelay** パラメーターの期間は ISO 8601 形式を使用して指定されます。たとえば、**PT1S** は 1 秒の遅延を指定します。

back off policy

backoffPolicy 配信パラメーターは再試行バックオフポリシーを指定するために使用できます。ポリシーは **linear** または **exponential** のいずれかとして指定できます。**linear** バックオフポリシーを使用する場合、バックオフ遅延は **backoffDelay * <numberOfRetries>** に等しくなります。**exponential** バックオフポリシーを使用する場合、バックオフ遅延は **backoffDelay*2^<numberOfRetries>** と等しくなります。

8.2. イベント配信パラメーターの設定例

Broker、**Trigger**、**Channel**、および **Subscription** オブジェクトのイベント配信パラメーターを設定できます。ブローカーまたはチャネルのイベント配信パラメーターを設定すると、これらのパラメーターは、それらのオブジェクト用に作成されたトリガーまたはサブスクリプションに伝播されます。トリガーまたはサブスクリプションのイベント配信パラメーターを設定して、ブローカーまたはチャネルの設定をオーバーライドすることもできます。

Broker オブジェクトの例

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

Trigger オブジェクトの例

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

Channel オブジェクトの例

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

Subscription オブジェクトの例

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

8.3. トリガーのイベント配信順序の設定

Kafka ブローカーを使用している場合は、トリガーからイベントシンクへのイベントの配信順序を設定できます。

前提条件

- OpenShift Serverless Operator、Knative Eventing、および Knative Kafka が OpenShift Container Platform クラスターにインストールされている。
- Kafka ブローカーがクラスターで使用可能であり、Kafka ブローカーが作成されている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- OpenShift (**oc**) CLI がインストールされている。

手順

1. **Trigger** オブジェクトを作成または変更し、**kafka.eventing.knative.dev/delivery.order** アノテーションを設定します。

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

サポートされているコンシューマー配信保証は次のとおりです。

unordered

順序付けられていないコンシューマーは、適切なオフセット管理を維持しながら、メッセージを順序付けずに配信するノンブロッキングコンシューマーです。

ordered

順序付きコンシューマーは、CloudEvent サブスクライバーからの正常な応答を待ってから、パーティションの次のメッセージを配信する、パーティションごとのブロックコンシューマーです。
デフォルトの順序保証は **unordered** です。

2. **Trigger** オブジェクトを適用します。

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

第9章 イベント検出

9.1. イベントソースおよびイベントソースタイプの一覧表示

OpenShift Container Platform クラスターに存在する、または使用可能なすべてのイベントソースやイベントソースタイプのリストを表示できます。OpenShift Container Platform Web コンソールの Knative (**kn**) CLI または **Developer** パースペクティブを使用し、利用可能なイベントソースまたはイベントソースタイプを一覧表示できます。

9.2. コマンドラインからのイベントソースタイプの一覧表示

Knative (**kn**) CLI を使用すると、クラスターで使用可能なイベントソースタイプを表示するための合理的で直感的なユーザーインターフェイスが提供されます。

9.2.1. Knative CLI の使用による利用可能なイベントソースタイプの一覧表示

kn source list-types CLI コマンドを使用して、クラスターで作成して使用できるイベントソースタイプをリスト表示できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。

手順

1. ターミナルに利用可能なイベントソースタイプをリスト表示します。

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

出力例

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

2. オプション: OpenShift Container Platform では、利用可能なイベントソースタイプを YAML 形式でリストすることもできます。

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

9.3. 開発者パースペクティブからのイベントソースタイプの一覧表示

クラスターで使用可能なすべてのイベントソースタイプを一覧表示できます。OpenShift Container Platform Web コンソールを使用すると、使用可能なイベントソースタイプを表示するための合理的で直感的なユーザーインターフェイスが提供されます。

9.3.1. 開発者パースペクティブ内での利用可能なイベントソースタイプの表示

前提条件

- OpenShift Container Platform Web コンソールにログインしている。
- OpenShift Serverless Operator および Knative Eventing が OpenShift Container Platform クラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

1. **Developer** パースペクティブにアクセスします。
2. **+Add** をクリックします。
3. **Event source** をクリックします。
4. 利用可能なイベントソースタイプを表示します。

9.4. コマンドラインからのイベントソースの一覧表示

Knative (**kn**) CLI を使用すると、クラスターの既存イベントソースを表示するための合理的で直感的なユーザーインターフェイスが提供されます。

9.4.1. Knative CLI の使用による利用可能なイベントリソースの一覧表示

kn source list コマンドを使用して、既存のイベントソースを一覧表示できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。
- Knative (**kn**) CLI がインストールされている。

手順

1. ターミナルにある既存のイベントソースをリスト表示します。

```
$ kn source list
```

出力例

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

2. オプションで、**--type** フラグを使用して、特定タイプのイベントソースのみを一覧表示できます。

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

コマンドの例

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

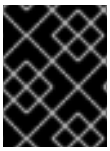
出力例

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

第10章 イベント設定のチューニング

10.1. KNATIVE EVENTING システムのデプロイメント設定のオーバーライド

KnativeEventing カスタムリソース (CR) の **ワークロード** 仕様を変更することで、特定のデプロイメントのデフォルト設定をオーバーライドできます。現在、デフォルトの構成設定のオーバーライドは、**eventing-controller**、**eventing-webhook**、および **imc-controller** フィールド、およびプローブの **readiness** フィールドと **liveness** フィールドでサポートされています。



重要

replicas の仕様は、Horizontal Pod Autoscaler (HPA) を使用するデプロイのレプリカの数をオーバーライドできず、**eventing-webhook** デプロイでは機能しません。

注記

デフォルトでデプロイメントに定義されているプローブのみをオーバーライドできます。

Knative Serving デプロイメントはすべて、以下の例外を除き、デフォルトで readiness および liveness プローブを定義します。

- **net-kourier-controller** および **3scale-kourier-gateway** は readiness プローブのみを定義します。
- **net-istio-controller** および **net-istio-webhook** はプローブを定義しません。

10.1.1. デプロイメント設定のオーバーライド

現在、デフォルトの構成設定のオーバーライドは、**eventing-controller**、**eventing-webhook**、および **imc-controller** フィールド、およびプローブの **readiness** フィールドと **liveness** フィールドでサポートされています。

重要

replicas の仕様は、Horizontal Pod Autoscaler (HPA) を使用するデプロイのレプリカの数をオーバーライドできず、**eventing-webhook** デプロイでは機能しません。

次の例では、**KnativeEventing** CR が **eventing-controller** デプロイメントをオーバーライドして、次のようにします。

- **readiness** プローブのタイムアウト **eventing-controller** は 10 秒に設定されています。
- デプロイメントには、CPU およびメモリーのリソース制限が指定されています。
- デプロイメントには 3 つのレプリカがあります。
- **example-label:labellabel** が追加されました。
- **example-annotation: annotation** が追加されます。

- **nodeSelector** フィールドは、**disktype: hdd** ラベルを持つノードを選択するように設定されます。

KnativeEventing CR の例

```
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 **readiness** および **liveness** プローブオーバーライドを使用して、プローブハンドラーに関連するフィールド (**exec**、**grpc**、**httpGet**、および **tcpSocket**) を除き、Kubernetes API で指定されているデプロイメントのコンテナー内のプローブのすべてのフィールドをオーバーライドできます。

注記

KnativeEventing CR ラベルおよびアノテーション設定は、デプロイメント自体と結果として生成される Pod の両方のデプロイメントのラベルおよびアノテーションを上書きします。

10.1.2. コンシューマーグループ ID とトピック名の変更

トリガー、ブローカー、チャネルで使用されるコンシューマーグループ ID とトピック名を生成するためのテンプレートを変更できます。

前提条件

- OpenShift Container Platform でクラスターまたは専用の管理者パーミッションを持っている。
- OpenShift Serverless Operator、Knative Eventing、および **KnativeKafka** カスタムリソース (CR) が OpenShift Container Platform クラスターにインストールされている。

- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. トリガー、ブローカー、チャネルで使用されるコンシューマーグループ ID とトピック名を生成するためのテンプレートを変更するには、**KnativeKafka** リソースを変更します。

```
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. トリガーで使用されるコンシューマーグループ ID を生成するためのテンプレート。有効な Go **text/template** 値を使用します。デフォルトは **{% raw %}"knative-trigger-{{ .Namespace }}-{{ .Name }}"{% endraw %}** です。
2. ブローカーが使用する Kafka トピック名を生成するためのテンプレート。有効な Go **text/template** 値を使用します。デフォルトは **{% raw %}"knative-broker-{{ .Namespace }}-{{ .Name }}"{% endraw %}** です。
3. チャネルで使用される Kafka トピック名を生成するためのテンプレート。有効な Go **text/template** 値を使用します。デフォルトは **{% raw %}"messaging-kafka.{{ .Namespace }}.{{ .Name }}"{% endraw %}** です。

テンプレート設定の例

```
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 ファイルを適用します。

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

関連情報

- Kubernetes API ドキュメントのプローブ設定セクション

10.2. 高可用性

高可用性 (HA) は Kubernetes API の標準的な機能で、中断が生じる場合に API が稼働を継続するのに役立ちます。HA デプロイメントでは、アクティブなコントローラーがクラッシュまたは削除されると、別のコントローラーをすぐに使用できます。このコントローラーは、現在使用できないコントローラーによって処理されていた API の処理を引き継ぎます。

OpenShift Serverless の HA は、リーダーの選択によって利用できます。これは、Knative Serving または Eventing コントロールプレーンのインストール後にデフォルトで有効になります。リーダー選択の HA パターンを使用する場合は、必要時に備えてコントローラーのインスタンスがスケジュールされ、クラスター内で実行されます。このコントローラーインスタンスは、リーダー選出ロックと呼ばれる共有リソースを使用するために競合します。リーダー選択ロックのリソースにアクセスできるコントローラーのインスタンスはリーダーと呼ばれます。

OpenShift Serverless の HA は、リーダーの選択によって利用できます。これは、Knative Serving または Eventing コントロールプレーンのインストール後にデフォルトで有効になります。リーダー選択の HA パターンを使用する場合は、必要時に備えてコントローラーのインスタンスがスケジュールされ、クラスター内で実行されます。このコントローラーインスタンスは、リーダー選出ロックと呼ばれる共有リソースを使用するために競合します。リーダー選択ロックのリソースにアクセスできるコントローラーのインスタンスはリーダーと呼ばれます。

10.2.1. Knative Eventing の高可用性レプリカの設定

Knative Eventing の **eventing-controller**、**eventing-webhook**、**imc-controller**、**imc-dispatcher**、**mt-broker-controller** コンポーネントは、デフォルトでそれぞれ 2 つのレプリカを持つように設定されており、高可用性 (HA) を利用することができます。**KnativeServing** カスタムリソース (CR) の **spec.high-availability.replicas** 値を変更して、これらのコンポーネントのレプリカ数を変更できます。

注記

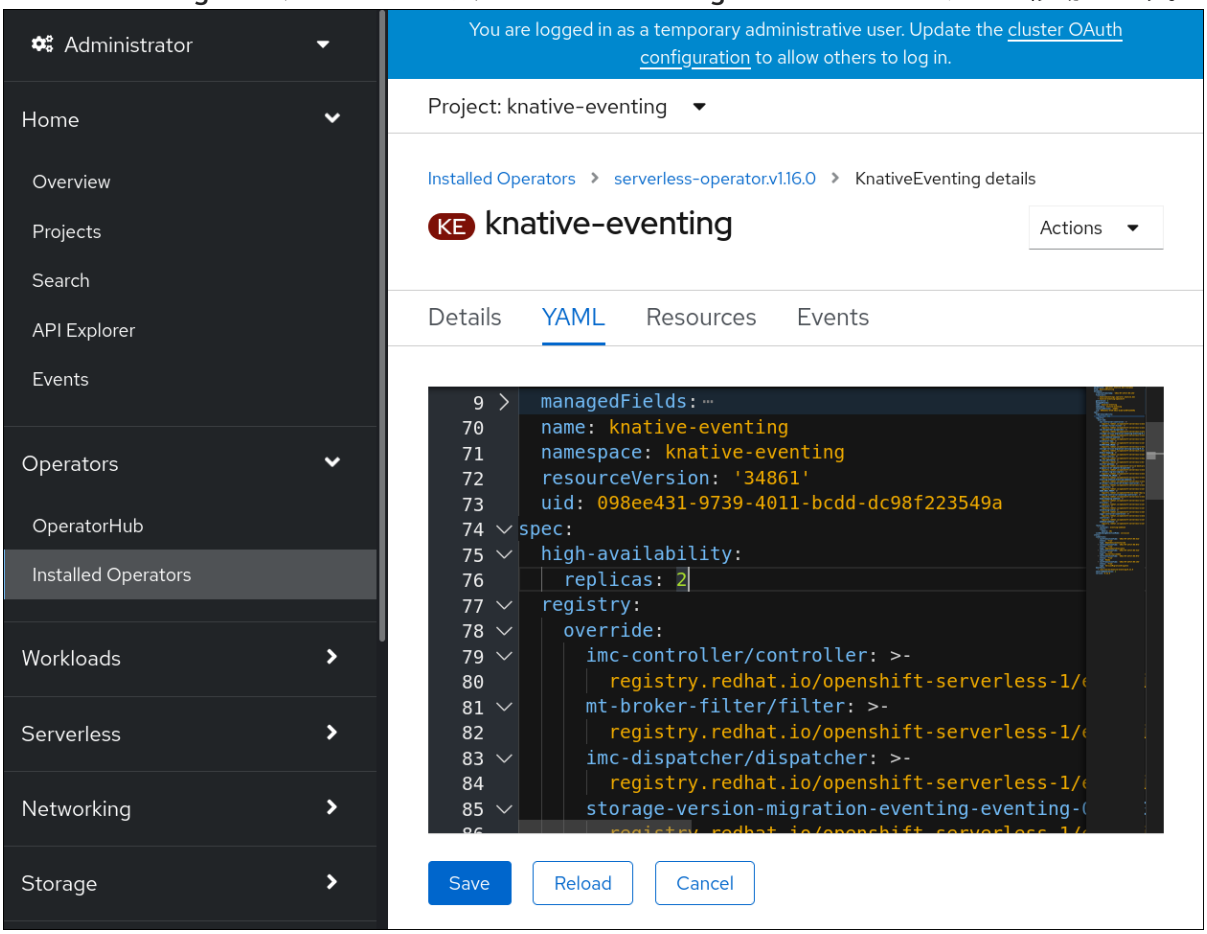
Knative Eventing の場合、HA では **mt-broker-filter** および **mt-broker-ingress** デプロイメントはスケーリングされません。複数のデプロイメントが必要な場合は、これらのコンポーネントを手動でスケーリングします。

前提条件

- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。
- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**OperatorHub → Installed Operators** に移動します。
2. **knative-eventing** namespace を選択します。
3. OpenShift Serverless Operator の **Provided API** 一覧で **Knative Eventing** をクリックし、**Knative Eventing** タブに移動します。
4. **knative-serving** をクリックしてから、**knative-eventing** ページの **YAML** タブに移動します。



5. **KnativeEvening** CR のレプリカ数を変更します。

サンプル YAML

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

10.2.2. Apache Kafka の Knative ブローカー実装の高可用性レプリカの設定

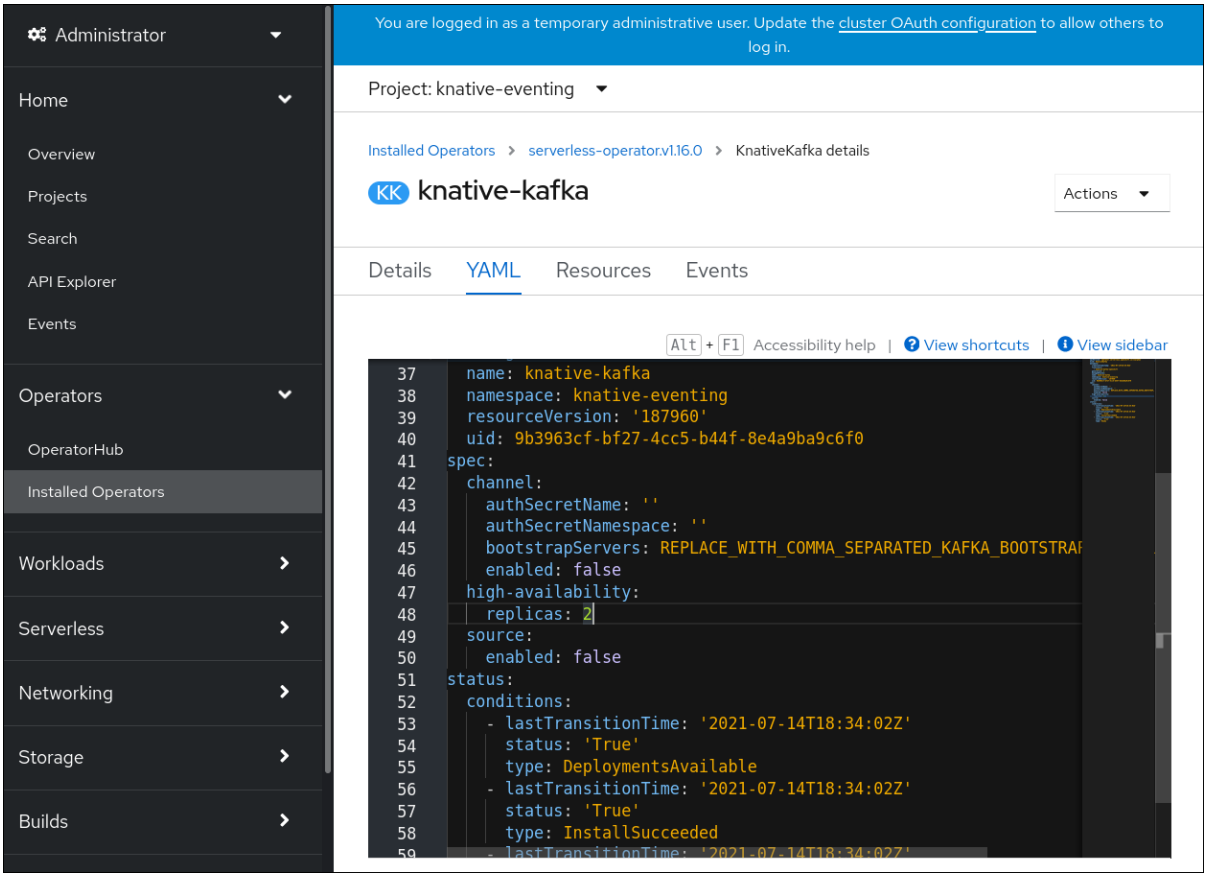
高可用性 (HA) は、Apache Kafka コンポーネント **kafka-controller** および **kafka-webhook-eventing** の Knative ブローカー実装にはデフォルトで提供されており、これらはデフォルトでそれぞれ 2 つのレプリカを持つように設定されています。**KnativeKafka** カスタムリソース (CR) の **spec.high-availability.replicas** 値を変更して、これらのコンポーネントのレプリカ数を変更できます。

前提条件

- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。
- OpenShift Serverless Operator と Apache Kafka 用の Knative ブローカーがクラスターにインストールされている。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**OperatorHub → Installed Operators** に移動します。
2. **knative-eventing** namespace を選択します。
3. OpenShift Serverless Operator の **Provided APIs** の一覧で **Knative Kafka** をクリックし、**Knative Kafka** タブに移動します。
4. **knative-kafka** をクリックしてから、**knative-kafka** ページの **YAML** タブに移動します。



5. **KnativeKafka** CR のレプリカ数を変更します。

サンプル YAML

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  name: knative-kafka
  namespace: knative-eventing
```

```
spec:
  high-availability:
    replicas: 3
```

第11章 イベント用の KUBE-RBAC-PROXY の設定

kube-rbac-proxy コンポーネントは、Knative Eventing の内部認証および認可機能を提供します。

11.1. イベント用の KUBE-RBAC-PROXY リソースの設定

OpenShift Serverless Operator CR を使用して、**kube-rbac-proxy** コンテナーのリソース割り当てをグローバルにオーバーライドできます。

注記

特定のデプロイメントのリソース割り当てをオーバーライドすることもできます。

次の設定では、Knative Eventing **kube-rbac-proxy** の最小および最大の CPU およびメモリー割り当てを設定します。

KnativeEventing CR の例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  config:
    deployment:
      "kube-rbac-proxy-cpu-request": "10m" (1)
      "kube-rbac-proxy-memory-request": "20Mi" (2)
      "kube-rbac-proxy-cpu-limit": "100m" (3)
      "kube-rbac-proxy-memory-limit": "100Mi" (4)
```

1. 最小 CPU 割り当てを設定します。
2. 最小 RAM 割り当てを設定します。
3. 最大 CPU 割り当てを設定します。
4. 最大 RAM 割り当てを設定します。

第12章 SERVICE MESH での CONTAINERSOURCE の使用

Service Mesh でコンテナーソースを使用できます。

12.1. SERVICE MESH を使用した CONTAINERSOURCE の設定

この手順では、Service Mesh を使用してコンテナーソースを設定する方法を説明します。

前提条件

- Service Mesh と Serverless の統合が設定されている。

手順

1. **ServiceMeshMemberRoll** のメンバーである namespace に **Service** を作成します。

   events-display-service.yaml 設定ファイルの例

```
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** のメンバーである namespace。

   2 このアノテーションは、Service Mesh サイドカーを Knative サービス Pod に挿入します。

2. **Service** リソースを適用します。

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

3. **ServiceMeshMemberRoll** のメンバーである namespace に **ContainerSource** を 作成し、**event-display** に設定されたシンクを作成します。

   test-heartbeats-containersource.yaml 設定ファイルの例

```
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** の一部である namespace。

2 これらのアノテーションにより、Service Mesh と **ContainerSource** オブジェクトの統合が可能になります。

4. **ContainerSource** リソースを適用します。

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

5. オプション: メッセージダンパー機能のログを調べて、イベントが Knative イベントシンクに送信されたことを確認します。

コマンドの例

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

出力例

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

```
Data,
  {
    "id": 1,
    "label": ""
  }
```

第13章 SERVICE MESH でのシンクバインディングの使用

Service Mesh でシンクバインディングを使用できます。

13.1. SERVICE MESH を使用したシンクバインディングの設定

この手順では、Service Mesh を使用してシンクバインディングを設定する方法を説明します。

前提条件

- Service Mesh と Serverless の統合が設定されている。

手順

1. **ServiceMeshMemberRoll** のメンバーである namespace に **Service** オブジェクトを作成します。

   events-display-service.yaml 設定ファイルの例

```
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** のメンバーである namespace。

   2 このアノテーションは、Service Mesh サイドカーを Knative サービス Pod に挿入します。

2. **Service** オブジェクトを適用します。

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

3. **SinkBinding** オブジェクトを作成します。

   heartbeat-sinkbinding.yaml 設定ファイルの例

```
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** の一部である namespace。
2. ラベル **app: heartbeat-cron** を持つ任意のジョブをイベントシンクにバインドします。

4. **SinkBinding** オブジェクトを適用します。

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

5. **CronJob** オブジェクトを作成します。

heartbeat-cronjob.yaml 設定ファイルの例

```
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** の一部である namespace。
2. Service Mesh サイドカーを **CronJob** Pod に挿入します。

6. **CronJob** オブジェクトを適用します。

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

7. オプション: メッセージダンパー機能のログを調べて、イベントが Knative イベントシンクに送信されたことを確認します。

コマンドの例

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

出力例

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