OpenShift Container Platform 4.11

서버리스

OpenShift 서버리스 설치, 사용법, 릴리스 정보

Last Updated: 2024-01-05

OpenShift Container Platform 4.11 서버리스

OpenShift 서버리스 설치, 사용법, 릴리스 정보

법적 공지

초록

이 문서에서는 OpenShift Container Platform에서 OpenShift 서버리스를 사용하는 방법에 대한 정보를 제공합니다.

차례

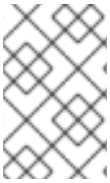
1장. 릴리스 노트

참고

OpenShift Serverless 라이프 사이클 및 지원되는 플랫폼에 대한 자세한 내용은 플랫폼 라이프 사이클 정책을 참조하십시오.

릴리스 노트에는 새 기능 및 더 이상 사용되지 않는 기능, 변경 사항 중단 및 알려진 문제에 대한 정보가 포함되어 있습니다. 다음 릴리스 노트는 OpenShift Container Platform의 최신 OpenShift Serverless 릴리스에 적용됩니다.

OpenShift Serverless 기능에 대한 개요는 OpenShift Serverless 정보를 참조하십시오.



참고

OpenShift Serverless는 오픈 소스 Knative 프로젝트를 기반으로 합니다.

최신 Knative 구성 요소 릴리스에 대한 자세한 내용은 Knative 블로그를 참조하십시오.

1.1. API 버전 정보

API 버전은 OpenShift Serverless의 특정 기능 및 사용자 정의 리소스의 개발 상태에 대한 중요한 척도입니다. 올바른 API 버전을 사용하지 않는 클러스터에서 리소스를 생성하면 배포에 문제가 발생할 수 있습니다.

OpenShift Serverless Operator는 최신 버전을 사용하기 위해 더 이상 사용되지 않는 API를 사용하는 이전 리소스를 자동으로 업그레이드합니다. 예를 들어 **v1beta1**과 같은 이전 **ApiServerSource** API 버전을 사용하는 리소스를 클러스터에 생성한 경우 OpenShift Serverless Operator는 사용 가능하고 **v1beta1** 버전이 더 이상 사용되지 않을 때 API의 **v1** 버전을 사용하도록 이러한 리소스를 자동으로 업데이트합니다.

더 이상 사용되지 않는 API의 이전 버전은 향후 릴리스에서 제거될 수 있습니다. 더 이상 사용되지 않는 API 버전을 사용해도 리소스가 실패하지 않습니다. 그러나 제거된 API 버전을 사용하려고 하면 리소스가 실패합니다. 문제를 방지하기 위해 최신 버전을 사용하도록 매니페스트가 업데이트되었는지 확인합니다.

1.2. 일반적으로 사용 가능한 기술 프리뷰 기능

일반적으로 사용할 수 있는 기능(GA)은 완전히 지원되며 프로덕션용으로 적합합니다. 기술 프리뷰(TP) 기능은 실험적인 기능이며 프로덕션 용도로는 사용할 수 없습니다. TP 기능에 대한 자세한 내용은 Red Hat 고객 포털에서 기술 프리뷰 지원 범위를 참조하십시오.

다음 표에서는 GA 및 TP인 OpenShift Serverless 기능에 대한 정보를 제공합니다.

표 1.1. 일반적으로 사용 가능한 기술 프리뷰 기능 추적

기능	1.26	1.27	1.28
kn func	GA	GA	GA
Quarkus 함수	GA	GA	GA
Node.js 함수	TP	TP	GA

기능	1.26	1.27	1.28
TypeScript 함수	TP	TP	GA
Python 함수	-	-	TP
서비스 메시 mTLS	GA	GA	GA
emptyDir 볼륨	GA	GA	GA
HTTPS 리디렉션	GA	GA	GA
Kafka 브로커	GA	GA	GA
Kafka 싱크	GA	GA	GA
Knative 서비스에 대한 init 컨테이너 지원	GA	GA	GA
Knative 서비스에 대한 PVC 지원	GA	GA	GA
내부 트래픽의 경우 TLS	TP	TP	TP
네임스페이스 범위 브로커	-	TP	TP
멀티컨테이너 지원	-	-	TP

1.3. 사용되지 않거나 삭제된 기능

GA(일반적으로 사용 가능) 또는 이전 릴리스에서 기술 프리뷰(TP)인 일부 기능은 더 이상 사용되지 않거나 제거되었습니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Serverless에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Serverless에서 더 이상 사용되지 않고 삭제된 주요 기능의 최신 목록은 다음 표를 참조하십시오.

표 1.2. 사용되지 않거나 삭제된 기능 추적

기능	1.20	1.21	1.22 ~ 1.26	1.27	1.28
KafkaBinding API	더 이상 사용되지 않음	더 이상 사용되지 않음	제거됨	제거됨	제거됨
kn func emit (C**kn func invoke** 1.21+)	더 이상 사용되지 않음	제거됨	제거됨	제거됨	제거됨

기능	1.20	1.21	1.22 ~ 1.26	1.27	1.28
Eventing **v1alpha1** API 제공 및	-	-	-	더 이상 사용되지 않음	더 이상 사용되지 않음
enable-secret-informer-filtering 주석	-	-	-	-	더 이상 사용되지 않음

1.4. RED HAT OPENSHIFT SERVERLESS 1.28 릴리스 노트

OpenShift Serverless 1.28이 출시되었습니다. 다음은 OpenShift Container Platform의 OpenShift Serverless와 관련된 새로운 기능, 변경 사항 및 알려진 문제에 대해 설명합니다.

1.4.1. 새로운 기능

- OpenShift Serverless에서 Knative Serving 1.7을 사용합니다.
- OpenShift Serverless에서 Knative Eventing 1.7을 사용합니다.
- OpenShift Serverless에서 Kourier 1.7을 사용합니다.
- OpenShift Serverless에서 Knative(**kn**) CLI 1.7을 사용합니다.
- OpenShift Serverless에서 Knative Kafka 1.7을 사용합니다.
- **kn func** CLI 플러그인은 이제 **func** 1.9.1 버전을 사용합니다.
- 이제 OpenShift Serverless Functions의 node.js 및 TypeScript 런타임을 GA(일반 사용 가능)로 사용할 수 있습니다.
- OpenShift Serverless Functions의 Python 런타임은 이제 기술 프리뷰로 사용할 수 있습니다.
- Knative Serving에 대한 멀티컨테이너 지원을 기술 프리뷰로 사용할 수 있습니다. 이 기능을 사용하면 단일 Knative 서비스를 사용하여 멀티컨테이너 Pod를 배포할 수 있습니다.
- OpenShift Serverless 1.29 이상에서는 Knative Eventing의 다음 구성 요소가 두 Pod에서 하나로 축소됩니다.
  - **imc-controller**
  - **imc-dispatcher**
  - **mt-broker-controller**
  - **mt-broker-filter**
  - **mt-broker-ingress**
- Serving CR에 대한 **서버리스.openshift.io/enable-secret-informer-filtering** 주석이 더 이상 사용되지 않습니다. 주석은 Istio에만 유효하며 Kourier에는 유효하지 않습니다.

OpenShift Serverless 1.28을 사용하면 OpenShift Serverless Operator에서 **net-istio** 및 **net-kourier** 모두에 환경 변수 **ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID** 를 삽입할 수 있습니다.

OpenShift Serverless 1.28에서 일부 향후 버전으로 업그레이드하는 경우 문제를 방지하려면 사용자가 **networking.internal.knative.dev/certificate-uid:some_cuid** 로 시크릿에 주석을 달아야 합니다.

1.4.2. 확인된 문제

- 현재 IBM Power, IBM zSystems 및 IBM® LinuxONE의 OpenShift Serverless Functions에서는 Python에 대한 런타임이 지원되지 않습니다.
  이러한 아키텍처에서 node.js, TypeScript 및 Quarkus 함수가 지원됩니다.

- Windows 플랫폼에서는 **app.sh** 파일 권한으로 Source-to-Image 빌더를 사용하여 Python 함수를 로컬로 빌드, 실행 또는 배포할 수 없습니다.
  이 문제를 해결하려면 Linux용 Windows를 사용하십시오.

1.5. RED HAT OPENSHIFT SERVERLESS 1.27 릴리스 노트

OpenShift Serverless 1.27을 사용할 수 있습니다. 다음은 OpenShift Container Platform의 OpenShift Serverless와 관련된 새로운 기능, 변경 사항 및 알려진 문제에 대해 설명합니다.

중요

OpenShift Serverless 1.26은 OpenShift Container Platform 4.12에서 완전히 지원되는 가장 빠른 릴리스입니다. OpenShift Serverless 1.25 이상은 OpenShift Container Platform 4.12에 배포되지 않습니다.

따라서 OpenShift Container Platform을 버전 4.12로 업그레이드하기 전에 먼저 OpenShift Serverless를 버전 1.26 또는 1.27로 업그레이드합니다.

1.5.1. 새로운 기능

- OpenShift Serverless에서 Knative Serving 1.6을 사용합니다.
- OpenShift Serverless에서 Knative Eventing 1.6을 사용합니다.
- OpenShift Serverless에서 Kourier 1.6을 사용합니다.
- OpenShift Serverless에서 Knative(**kn**) CLI 1.6을 사용합니다.
- OpenShift Serverless에서 Knative Kafka 1.6을 사용합니다.
- **kn func** CLI 플러그인은 이제 **func** 1.8.1을 사용합니다.
- 네임스페이스 범위 브로커가 이제 기술 프리뷰로 제공됩니다. 예를 들어 이러한 브로커를 사용하여 역할 기반 액세스 제어(RBAC) 정책을 구현할 수 있습니다.
- 이제 **KafkaSink** 는 기본적으로 **CloudEvent** 바이너리 콘텐츠 모드를 사용합니다. 바이너리 콘텐츠 모드는 **CloudEvent** 대신 본문의 헤더를 사용하므로 구조화된 모드보다 효율적입니다. 예를 들어 HTTP 프로토콜의 경우 HTTP 헤더를 사용합니다.

- OpenShift Container Platform 4.10 이상에서 OpenShift 경로를 사용하여 외부 트래픽에 HTTP/2 프로토콜을 통해 gRPC 프레임워크를 사용할 수 있습니다. 이를 통해 클라이언트와 서버 간 통신의 효율성 및 속도가 향상됩니다.

- Knative Operator Serving 및 Eventings CRD의 API 버전 **v1alpha1** 은 1.27에서 더 이상 사용되지 않습니다. 이는 향후 버전에서 제거됩니다. 대신 **v1beta1** 버전을 사용하는 것이 좋습니다. 이는 Serverless Operator를 업그레이드할 때 CRD가 자동으로 업데이트되므로 기존 설치에는 영향을 미치지 않습니다.

- 전달 제한 시간 기능은 기본적으로 활성화되어 있습니다. 이를 통해 전송된 각 HTTP 요청에 대한 타임아웃을 지정할 수 있습니다. 이 기능은 기술 프리뷰로 남아 있습니다.

1.5.2. 해결된 문제

- 이전 버전에서는 Knative 서비스가 **Ready** 상태가 되지 않아 로드 밸런서가 준비될 때까지 대기 시간을 보고하는 경우가 있었습니다. 이 문제가 해결되었습니다.

1.5.3. 확인된 문제

- OpenShift Serverless를 Red Hat OpenShift Service Mesh와 통합하면 클러스터에 너무 많은 보안이 있을 때 **net-kourier** Pod가 시작 시 메모리가 부족합니다.

- 네임스페이스 범위 브로커는 네임스페이스 범위의 브로커를 삭제한 후에도 사용자 네임스페이스에 **ClusterRoleBindings** 를 남겨 둘 수 있습니다.
  이 경우 사용자 네임스페이스에서 **rbac-proxy-reviews-prom-rb-knative-kafka-broker-data-plane-{{.Namespace}}** 라는 **ClusterRoleBinding** 을 삭제합니다.

- Ingress에 **net-istio** 를 사용하고 **security.dataPlane.mtls: true** 를 사용하여 SMCP를 통해 mTLS를 활성화하는 경우 Service Mesh는 OpenShift Serverless에 대한 **DomainMapping** 을 허용하지 않는 ***.local** 호스트에 대한 **DestinationRule** 을 배포합니다.
  이 문제를 해결하려면 **security.dataPlane.mtls: true** 를 사용하는 대신 **PeerAuthentication** 을 배포하여 mTLS를 활성화합니다.

1.6. RED HAT OPENSHIFT SERVERLESS 1.26 릴리스 노트

OpenShift Serverless 1.26이 공개되었습니다. 다음은 OpenShift Container Platform의 OpenShift Serverless와 관련된 새로운 기능, 변경 사항 및 알려진 문제에 대해 설명합니다.

1.6.1. 새로운 기능

- Quarkus를 사용하는 OpenShift Serverless Functions가 이제 GA입니다.

- OpenShift Serverless에서 Knative Serving 1.5를 사용합니다.

- OpenShift Serverless에서 Knative Eventing 1.5를 사용합니다.

- OpenShift Serverless에서 Kourier 1.5를 사용합니다.

- OpenShift Serverless에서 Knative(**kn**) CLI 1.5를 사용합니다.

- OpenShift Serverless에서 Knative Kafka 1.5를 사용합니다.

- OpenShift Serverless에서 Knative Operator 1.3을 사용합니다.

- **kn func** CLI 플러그인에서 **func** 1.8.1을 사용합니다.

- PVC(영구 볼륨 클레임)는 이제 GA입니다. PVC는 Knative 서비스에 영구 데이터 스토리지를 제공합니다.

- 새로운 트리거 필터 기능을 이제 개발자 프리뷰로 사용할 수 있습니다. 이를 통해 사용자는 필터 표현식 세트를 지정할 수 있으며 각 표현식은 각 이벤트에 대해 true 또는 false로 평가됩니다.
  새 트리거 필터를 활성화하려면 Operator 구성 맵의 **KnativeEventing** 유형의 섹션에 **new-trigger-filters: enabled** 항목을 추가합니다.

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
...
...
spec:
  config:
    features:
      new-trigger-filters: enabled
...
```

- Knative Operator 1.3은 **operator.knative.dev** 용으로 API의 업데이트된 **v1beta1** 버전을 추가합니다.
  KnativeServing 및 **KnativeEventing** 사용자 정의 리소스 구성 맵의 **v1alpha1** 에서 **v1beta1** 로 업데이트하려면 **apiVersion** 키를 편집합니다.

  KnativeServing 사용자 정의 리소스 구성 맵의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
...
```

  KnativeEventing 사용자 정의 리소스 구성 맵의 예

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

1.6.2. 해결된 문제

- 이전에는 Kafka 브로커, Kafka 소스 및 Kafka 싱크에서 FIPS(Federal Information Processing Standards) 모드가 비활성화되었습니다. 이 문제가 해결되었으며 이제 FIPS 모드를 사용할 수 있습니다.

1.6.3. 확인된 문제

- Ingress에 **net-istio** 를 사용하고 **security.dataPlane.mtls: true** 를 사용하여 SMCP를 통해 mTLS를 활성화하는 경우 Service Mesh는 OpenShift Serverless에 대한 **DomainMapping** 을 허용하지 않는 ***.local** 호스트에 대한 **DestinationRule** 을 배포합니다.
  이 문제를 해결하려면 **security.dataPlane.mtls: true** 를 사용하는 대신 **PeerAuthentication** 을 배포하여 mTLS를 활성화합니다.

추가 리소스

- 새 트리거 필터에 대한 Knative 문서

1.7. RED HAT OPENSHIFT SERVERLESS 1.25.0 릴리스 노트

OpenShift Serverless 1.25.0이 공개되었습니다. 다음은 OpenShift Container Platform의 OpenShift Serverless와 관련된 새로운 기능, 변경 사항 및 알려진 문제에 대해 설명합니다.

1.7.1. 새로운 기능

- OpenShift Serverless에서 Knative Serving 1.4를 사용합니다.
- OpenShift Serverless에서 Knative Eventing 1.4를 사용합니다.
- OpenShift Serverless에서 Kourier 1.4를 사용합니다.
- OpenShift Serverless에서 Knative(**kn**) CLI 1.4를 사용합니다.
- OpenShift Serverless에서 Knative Kafka 1.4를 사용합니다.
- **kn func** CLI 플러그인에서 **func** 1.7.0을 사용합니다.
- 함수 생성 및 배포를 위한 IDE(통합 개발 환경) 플러그인을 이제 Visual Studio Code 및 IntelliJ 에서 사용할 수 있습니다.
- Knative Kafka 브로커는 이제 GA입니다. Knative Kafka 브로커는 Apache Kafka를 직접 대상으로 하는 Knative 브로커 API의 고성능 구현입니다.
  MT-Channel-Broker를 사용하지 않는 것이 좋지만 Knative Kafka 브로커를 대신 사용하지 않는 것이 좋습니다.
- Knative Kafka 싱크는 이제 GA입니다. **KafkaSink** 는 **CloudEvent** 를 사용하여 Apache Kafka 주제로 보냅니다. 이벤트는 구조화된 또는 바이너리 콘텐츠 모드로 지정할 수 있습니다.
- 내부 트래픽에 대한 TLS 활성화가 기술 프리뷰로 제공됩니다.

1.7.2. 해결된 문제

- 이전 버전에서는 활성 상태 프로브가 실패한 후 컨테이너를 재시작한 경우 Knative Serving에 준비 상태 프로브가 실패한 문제가 있었습니다. 이 문제가 해결되었습니다.

1.7.3. 확인된 문제

- Kafka 브로커, Kafka 소스 및 Kafka 싱크에 대해 연방 정보 처리 표준(FIPS) 모드가 비활성화됩니다.
- **SinkBinding** 오브젝트는 서비스에 대한 사용자 정의 리버전 이름을 지원하지 않습니다.
- Knative Serving 컨트롤러 Pod는 클러스터의 시크릿을 감시하기 위해 새로운 정보원을 추가합니다. informer에는 캐시에 시크릿이 포함되어 있으므로 컨트롤러 Pod의 메모리 사용량이 증가합니다.
  Pod가 메모리가 부족하면 배포의 메모리 제한을 늘려 문제를 해결할 수 있습니다.
- Ingress에 **net-istio** 를 사용하고 **security.dataPlane.mtls: true** 를 사용하여 SMCP를 통해 mTLS를 활성화하는 경우 Service Mesh는 OpenShift Serverless에 대한 **DomainMapping** 을 허용하지 않는 ***.local** 호스트에 대한 **DestinationRule** 을 배포합니다.
  이 문제를 해결하려면 **security.dataPlane.mtls: true** 를 사용하는 대신 **PeerAuthentication** 을 배포하여 mTLS를 활성화합니다.

추가 리소스

- TLS 인증 구성

1.8. RED HAT OPENSHIFT SERVERLESS 1.24.0 릴리스 정보

OpenShift Serverless 1.24.0을 사용할 수 있습니다. 다음은 OpenShift Container Platform의 OpenShift Serverless와 관련된 새로운 기능, 변경 사항 및 알려진 문제에 대해 설명합니다.

1.8.1. 새로운 기능

- OpenShift Serverless에서 Knative Serving 1.3을 사용합니다.
- OpenShift Serverless에서 Knative Eventing 1.3을 사용합니다.
- OpenShift Serverless에서 Kourier 1.3을 사용합니다.
- OpenShift Serverless에서 Knative **kn** CLI 1.3을 사용합니다.
- OpenShift Serverless에서 Knative Kafka 1.3을 사용합니다.
- **kn func** CLI 플러그인에서 **func** 0.24를 사용합니다.
- 이제 Knative 서비스에 대한 init 컨테이너 지원을 일반적으로 사용할 수 있습니다(GA).
- OpenShift Serverless 논리를 개발자 프리뷰로 사용할 수 있습니다. 서버리스 애플리케이션을 관리하기 위한 선언적 워크플로 모델을 정의할 수 있습니다.
- 이제 OpenShift Serverless에서 비용 관리 서비스를 사용할 수 있습니다.

1.8.2. 해결된 문제

- OpenShift Serverless를 Red Hat OpenShift Service Mesh와 통합하면 클러스터에 너무 많은 시크릿이 있을 때 **net-istio-controller** pod가 시작 시 메모리 부족으로 실행됩니다.
  이제 **net-istio-controller** 가 **networking.internal.knative.dev/certificate-uid** 레이블이 있는 보안만 고려하도록 하여 필요한 메모리 양을 줄일 수 있습니다.
- OpenShift Serverless Functions Technology Preview에서는 이제 기본적으로 Cloud Native Buildpacks 를 사용하여 컨테이너 이미지를 빌드합니다.

1.8.3. 확인된 문제

- Kafka 브로커, Kafka 소스 및 Kafka 싱크에 대해 연방 정보 처리 표준(FIPS) 모드가 비활성화됩니다.
- OpenShift Serverless 1.23에서 KafkaBinding을 지원하고 **kafka-binding** Webhook가 제거되었습니다. 그러나 기존 **kafkabindings.webhook.kafka.sources.knative.dev MutatingWebhookConfiguration** 은 더 이상 존재하지 않는 **kafka-source-webhook** 서비스를 가리키는 남아 있을 수 있습니다.
  클러스터의 KafkaBindings의 특정 사양에 대해 **kafkabindings.webhook.kafka.sources.knative.dev MutatingWebhookConfiguration** 은 생성 및 업데이트 이벤트를(예: Deployments, Knative 서비스 또는 Jobs)에 전달하도록 구성할 수 있습니다. 그러면 Webhook를 통해 오류가 발생합니다.

이 문제를 해결하려면 OpenShift Serverless 1.23으로 업그레이드한 후 클러스터에서 **kafkabindings.webhook.kafka.sources.knative.dev MutatingWebhookConfiguration** 을 수동으로 삭제합니다.

```
$ oc delete mutatingwebhookconfiguration kafkabindings.webhook.kafka.sources.knative.dev
```

- Ingress에 **net-istio** 를 사용하고 **security.dataPlane.mtls: true** 를 사용하여 SMCP를 통해 mTLS를 활성화하는 경우 Service Mesh는 OpenShift Serverless에 대한 **DomainMapping** 을 허용하지 않는 ***.local** 호스트에 대한 **DestinationRule** 을 배포합니다.
  이 문제를 해결하려면 **security.dataPlane.mtls: true** 를 사용하는 대신 **PeerAuthentication** 을 배포하여 mTLS를 활성화합니다.

1.9. RED HAT OPENSHIFT SERVERLESS 1.23.0 릴리스 정보

OpenShift Serverless 1.23.0을 사용할 수 있습니다. 다음은 OpenShift Container Platform의 OpenShift Serverless와 관련된 새로운 기능, 변경 사항 및 알려진 문제에 대해 설명합니다.

1.9.1. 새로운 기능

- OpenShift Serverless에서 Knative Serving 1.2를 사용합니다.
- OpenShift Serverless에서 Knative Eventing 1.2를 사용합니다.
- OpenShift Serverless에서 Kourier 1.2를 사용합니다.
- OpenShift Serverless에서 Knative(**kn**) CLI 1.2를 사용합니다.
- OpenShift Serverless에서 Knative Kafka 1.2를 사용합니다.
- **kn func** CLI 플러그인에서 **func** 0.24를 사용합니다.
- Kafka 브로커와 함께 **kafka.eventing.knative.dev/external.topic** 주석을 사용할 수 있습니다. 이 주석을 사용하면 브로커가 자체 내부 주제를 생성하는 대신 외부로 관리되는 기존 주제를 사용할 수 있습니다.
- **kafka-ch-controller** 및 **kafka-webhook** Kafka 구성 요소가 더 이상 존재하지 않습니다. 이러한 구성 요소는 **kafka-webhook-eventing** 구성 요소로 교체되었습니다.
- OpenShift Serverless Functions Technology Preview에서는 기본적으로 S2I(Source-to-Image)를 사용하여 컨테이너 이미지를 빌드합니다.

1.9.2. 확인된 문제

- Kafka 브로커, Kafka 소스 및 Kafka 싱크에 대해 연방 정보 처리 표준(FIPS) 모드가 비활성화됩니다.
- Kafka 브로커를 포함하는 네임스페이스를 삭제하는 경우 브로커의 **auth.secret.ref.name** 보안이 브로커 전에 삭제되면 네임스페이스 종료자가 제거되지 않을 수 있습니다.
- 많은 Knative 서비스로 OpenShift Serverless를 실행하면 Knative 활성화기 Pod가 600MB의 기본 메모리 제한에 근접하게 실행될 수 있습니다. 메모리 사용량이 이 제한에 도달하면 이러한 Pod를 재시작할 수 있습니다. activator 배포에 대한 요청 및 제한은 **KnativeServing** 사용자 정의 리소스를 수정하여 구성할 수 있습니다.

```
apiVersion: operator.knative.dev/v1beta1
```

```
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  deployments:
  - name: activator
    resources:
    - container: activator
      requests:
        cpu: 300m
        memory: 60Mi
      limits:
        cpu: 1000m
        memory: 1000Mi
```

- 함수의 로컬 빌드 전략으로 Cloud Native Buildpacks 를 사용하는 경우 **kn func** 는 podman을 자동으로 시작하거나 원격 데몬에 SSH 터널을 사용할 수 없습니다. 이러한 문제의 해결 방법은 함수를 배포하기 전에 로컬 개발 컴퓨터에서 Docker 또는 podman 데몬을 이미 실행하는 것입니다.
- 현재 Quarkus 및 Golang 런타임의 경우 온프레미스 함수 빌드가 실패합니다. Node, Typescript, Python, Springboot 런타임에서 올바르게 작동합니다.
- Ingress에 **net-istio** 를 사용하고 **security.dataPlane.mtls: true** 를 사용하여 SMCP를 통해 mTLS를 활성화하는 경우 Service Mesh는 OpenShift Serverless에 대한 **DomainMapping** 을 허용하지 않는 ***.local** 호스트에 대한 **DestinationRule** 을 배포합니다.
  이 문제를 해결하려면 **security.dataPlane.mtls: true** 를 사용하는 대신 **PeerAuthentication** 을 배포하여 mTLS를 활성화합니다.

추가 리소스

- S2I(Source-to-Image)

1.10. RED HAT OPENSHIFT SERVERLESS 1.22.0 릴리스 정보

OpenShift Serverless 1.22.0이 공개되었습니다. 다음은 OpenShift Container Platform의 OpenShift Serverless와 관련된 새로운 기능, 변경 사항 및 알려진 문제에 대해 설명합니다.

1.10.1. 새로운 기능

- OpenShift Serverless에서 Knative Serving 1.1을 사용합니다.
- OpenShift Serverless에서 Knative Eventing 1.1을 사용합니다.
- OpenShift Serverless에서 Kourier 1.1을 사용합니다.
- OpenShift Serverless에서 Knative(**kn**) CLI 1.1을 사용합니다.
- OpenShift Serverless에서 Knative Kafka 1.1을 사용합니다.
- **kn func** CLI 플러그인에서 **func** 0.23를 사용합니다.
- Knative 서비스에 대한 init 컨테이너 지원은 이제 기술 프리뷰로 제공됩니다.
- Knative 서비스에 대한 PVC(영구 볼륨 클레임) 지원이 기술 프리뷰로 제공됩니다.

- **knative-serving**,**knative-serving-ingress**,**knative-eventing** 및 **knative-kafka** 시스템 네임스페이스에 기본적으로 **knative.openshift.io/part-of: "openshift-serverless"** 라벨이 지정됩니다.
- **Knative Eventing - Kafka Broker/Trigger** 대시보드가 추가되어 웹 콘솔에서 Kafka 브로커를 시각화하고 메트릭을 트리거할 수 있습니다.
- 웹 콘솔에서 KafkaSink 메트릭을 시각화할 수 있는 **Knative Eventing - KafkaSink** 대시보드가 추가되었습니다.
- **Knative Eventing - Broker/Trigger** 대시보드를 **Knative Eventing - 채널 기반 브로커/Trigger** 대시보드라고 합니다.
- **knative.openshift.io/part-of: "openshift-serverless"** 라벨은 **knative.openshift.io/system-namespace** 레이블을 대체했습니다.
- Knative Serving YAML 구성 파일의 이름 지정 스타일은 camel 케이스 (**ExampleName**)에서 하이픈 스타일 (**example-name**)으로 변경되었습니다. 이 릴리스부터 Knative Serving YAML 구성 파일을 생성하거나 편집할 때 하이픈 스타일 표기법을 사용합니다.

1.10.2. 확인된 문제

- Kafka 브로커, Kafka 소스 및 Kafka 싱크에 대해 연방 정보 처리 표준(FIPS) 모드가 비활성화됩니다.

1.11. RED HAT OPENSHIFT SERVERLESS 1.21.0 릴리스 정보

OpenShift Serverless 1.21.0이 공개되었습니다. 다음은 OpenShift Container Platform의 OpenShift Serverless와 관련된 새로운 기능, 변경 사항 및 알려진 문제에 대해 설명합니다.

1.11.1. 새로운 기능

- OpenShift Serverless에서 Knative Serving 1.0 사용
- OpenShift Serverless에서 Knative Eventing 1.0을 사용합니다.
- OpenShift Serverless에서 Kourier 1.0을 사용합니다.
- OpenShift Serverless에서 Knative(**kn**) CLI 1.0을 사용합니다.
- OpenShift Serverless에서 Knative Kafka 1.0을 사용합니다.
- **kn func** CLI 플러그인에서 **func** 0.21을 사용합니다.
- Kafka 싱크를 기술 프리뷰로 사용할 수 있습니다.
- Knative 오픈 소스 프로젝트는 kebab-cased 키를 일관되게 사용하기 위해 사용 중단 연고 구성 키를 사용하기 시작했습니다. 결과적으로 OpenShift Serverless 1.18.0 릴리스 노트에 앞서 언급한 **defaultExternalScheme** 키가 더 이상 사용되지 않으며 **default-external-scheme** 키로 교체되었습니다. 키에 대한 사용 지침은 동일하게 유지됩니다.

1.11.2. 해결된 문제

- OpenShift Serverless 1.20.0에서는 **kn event send** 를 사용하여 서비스에 이벤트를 보내는 데 영향을 미치는 이벤트 전달 문제가 발생했습니다. 이제 이 문제가 해결되었습니다.

- OpenShift Serverless 1.20.0(**func** 0.20)에서 **http** 템플릿으로 생성된 TypeScript 함수가 클러스터에 배포되지 않았습니다. 이제 이 문제가 해결되었습니다.
- OpenShift Serverless 1.20.0(**func** 0.20)에서 **gcr.io** 레지스트리를 사용하여 함수를 배포하면 오류와 함께 실패했습니다. 이제 이 문제가 해결되었습니다.
- OpenShift Serverless 1.20.0(**func** 0.20)에서 **kn func create** 명령을 사용하여 Springboot 함수 프로젝트 디렉터리를 생성한 다음 **kn func build** 명령을 실행하면 오류 메시지가 표시됩니다. 이제 이 문제가 해결되었습니다.
- OpenShift Serverless 1.19.0(**func** 0.19)에서는 podman을 사용하여 일부 런타임에서 함수를 빌드할 수 없었습니다. 이제 이 문제가 해결되었습니다.

1.11.3. 확인된 문제

- 현재 도메인 매핑 컨트롤러는 현재 지원되지 않는 경로가 포함된 브로커의 URI를 처리할 수 없습니다.
  즉, **DomainMapping** CR(사용자 정의 리소스)을 브로커에 매핑하려면 브로커의 Ingress 서비스로 **DomainMapping** CR을 구성하고 브로커의 정확한 경로를 사용자 정의 도메인에 추가해야 합니다.

  DomainMapping CR의 예

  ```
  apiVersion: serving.knative.dev/v1alpha1
  kind: DomainMapping
  metadata:
    name: <domain-name>
    namespace: knative-eventing
  spec:
    ref:
      name: broker-ingress
      kind: Service
      apiVersion: v1
  ```

  브로커의 URI는 < **domain-name>/<broker-namespace>/<broker-name> 입니다**.

1.12. RED HAT OPENSHIFT SERVERLESS 1.20.0 릴리스 정보

OpenShift Serverless 1.20.0이 공개되었습니다. 다음은 OpenShift Container Platform의 OpenShift Serverless와 관련된 새로운 기능, 변경 사항 및 알려진 문제에 대해 설명합니다.

1.12.1. 새로운 기능

- OpenShift Serverless에서 Knative Serving 0.26을 사용합니다.
- OpenShift Serverless에서 Knative Eventing 0.26을 사용합니다.
- OpenShift Serverless에서 Kourier 0.26을 사용합니다.
- OpenShift Serverless에서 Knative(**kn**) CLI 0.26을 사용합니다.
- OpenShift Serverless에서 Knative Kafka 0.26을 사용합니다.
- **kn func** CLI 플러그인에서 **func** 0.20을 사용합니다.

- Kafka 브로커는 이제 기술 프리뷰로 사용할 수 있습니다.

중요

현재 기술 프리뷰에 있는 Kafka 브로커는 FIPS에서 지원되지 않습니다.

- **kn 이벤트** 플러그인을 기술 프리뷰로 사용할 수 있습니다.
- **kn service create** 명령의 **--min-scale** 및 **--max-scale** 플래그가 더 이상 사용되지 않습니다. 대신 **--scale-min** 및 **--scale-max** 플래그를 사용합니다.

1.12.2. 확인된 문제

- OpenShift Serverless는 HTTPS를 사용하는 기본 주소로 Knative 서비스를 배포합니다. 클러스터 내부의 리소스로 이벤트를 보낼 때 보낸 사람에 클러스터 인증 기관(CA)이 구성되어 있지 않습니다. 이로 인해 클러스터가 전역적으로 허용되는 인증서를 사용하지 않는 한 이벤트 전달이 실패합니다.
  예를 들어 공개적으로 액세스 가능한 주소로 이벤트 전달이 작동합니다.

```
$ kn event send --to-url https://ce-api.foo.example.com/
```

  반면 서비스에서 사용자 정의 CA에서 발행한 HTTPS 인증서와 함께 공용 주소를 사용하는 경우 이 전송이 실패합니다.

```
$ kn event send --to Service:serving.knative.dev/v1:event-display
```

  브로커 또는 채널과 같은 다른 주소 지정 가능 개체로 이벤트를 전송하는 것은 이 문제의 영향을 받지 않으며 예상대로 작동합니다.

- Kafka 브로커는 현재 연방 정보 처리 표준(FIPS) 모드가 활성화된 클러스터에서 작동하지 않습니다.
- **kn func create** 명령을 사용하여 Springboot 함수 프로젝트 디렉터리를 생성하면 **kn func build** 명령의 후속 실행이 실패하고 다음 오류 메시지가 표시됩니다.

```
[analyzer] no stack metadata found at path ''
[analyzer] ERROR: failed to : set API for buildpack 'paketo-buildpacks/ca-certificates@3.0.2':
buildpack API version '0.7' is incompatible with the lifecycle
```

  이 문제를 해결하려면 함수 구성 파일 **func.yaml** 에서 **builder** 속성을 **gcr.io/paketo-buildpacks/builder:base** 로 변경할 수 있습니다.

- **gcr.io** 레지스트리를 사용하여 함수를 배포하면 이 오류 메시지와 함께 실패합니다.

```
Error: failed to get credentials: failed to verify credentials: status code: 404
```

  이 문제를 해결하려면 **quay.io** 또는 **docker.io** 와 같은 **gcr.io** 와 다른 레지스트리를 사용하십시오.

- **http** 템플릿으로 생성된 TypeScript 기능은 클러스터에 배포되지 않습니다.
  이 문제를 해결하려면 **func.yaml** 파일에서 다음 섹션을 교체합니다.

```
buildEnvs: []
```

이 경우 다음을 수행합니다.

```
buildEnvs:
- name: BP_NODE_RUN_SCRIPTS
  value: build
```

- **func** 버전 0.20에서 일부 런타임은 podman을 사용하여 함수를 빌드하지 못할 수 있습니다. 다음과 유사한 오류 메시지가 표시될 수 있습니다.

```
ERROR: failed to image: error during connect: Get
"http://%2Fvar%2Frun%2Fdocker.sock/v1.40/info": EOF
```

  - 이 문제에 대한 다음 해결 방법이 있습니다.

    a. service **ExecStart** 정의에 **--time=0** 을 추가하여 podman 서비스를 업데이트합니다.

    서비스 구성 예

    ```
    ExecStart=/usr/bin/podman $LOGGING system service --time=0
    ```

    b. 다음 명령을 실행하여 podman 서비스를 다시 시작합니다.

    ```
    $ systemctl --user daemon-reload
    ```

    ```
    $ systemctl restart --user podman.socket
    ```

  - 또는 TCP를 사용하여 podman API를 노출할 수 있습니다.

    ```
    $ podman system service --time=0 tcp:127.0.0.1:5534 &
    export DOCKER_HOST=tcp://127.0.0.1:5534
    ```

1.13. RED HAT OPENSHIFT SERVERLESS 1.19.0 릴리스 노트

OpenShift Serverless 1.19.0이 공개되었습니다. 다음은 OpenShift Container Platform의 OpenShift Serverless와 관련된 새로운 기능, 변경 사항 및 알려진 문제에 대해 설명합니다.

1.13.1. 새로운 기능

- OpenShift Serverless에서 Knative Serving 0.25를 사용합니다.
- OpenShift Serverless에서 Knative Eventing 0.25를 사용합니다.
- OpenShift Serverless에서 Kourier 0.25를 사용합니다.
- OpenShift Serverless에서 Knative(**kn**) CLI 0.25를 사용합니다.
- OpenShift Serverless에서 Knative Kafka 0.25를 사용합니다.
- **kn func** CLI 플러그인에서 **func** 0.19를 사용합니다.
- **KafkaBinding** API는 OpenShift Serverless 1.19.0에서 더 이상 사용되지 않으며 향후 릴리스에서 제거됩니다.

- 이제 HTTPS 리디렉션이 지원되며 클러스터 또는 각 Knative 서비스에 대해 전역적으로 구성할 수 있습니다.

1.13.2. 해결된 문제

- 이전 릴리스에서는 Kafka 채널 디스패처가 응답하기 전에 로컬 커밋이 성공할 때만 대기했습니다. 이로 인해 Apache Kafka 노드 실패 시 이벤트가 손실되었을 수 있었습니다. 이제 Kafka 채널 디스패처가 응답하기 전에 모든 동기화 복제본이 커밋될 때까지 기다립니다.

1.13.3. 확인된 문제

- **func** 버전 0.19에서는 일부 런타임에서 podman을 사용하여 함수를 빌드할 수 없을 수 있습니다. 다음과 유사한 오류 메시지가 표시될 수 있습니다.

```
ERROR: failed to image: error during connect: Get
"http://%2Fvar%2Frun%2Fdocker.sock/v1.40/info": EOF
```

  - 이 문제에 대한 다음 해결 방법이 있습니다.

    a. service **ExecStart** 정의에 **--time=0** 을 추가하여 podman 서비스를 업데이트합니다.

    서비스 구성 예

    ```
    ExecStart=/usr/bin/podman $LOGGING system service --time=0
    ```

    b. 다음 명령을 실행하여 podman 서비스를 다시 시작합니다.

    ```
    $ systemctl --user daemon-reload
    ```

    ```
    $ systemctl restart --user podman.socket
    ```

  - 또는 TCP를 사용하여 podman API를 노출할 수 있습니다.

```
$ podman system service --time=0 tcp:127.0.0.1:5534 &
export DOCKER_HOST=tcp://127.0.0.1:5534
```

1.14. RED HAT OPENSHIFT SERVERLESS 1.18.0 릴리스 정보

OpenShift Serverless 1.18.0을 사용할 수 있습니다. 다음은 OpenShift Container Platform의 OpenShift Serverless와 관련된 새로운 기능, 변경 사항 및 알려진 문제에 대해 설명합니다.

1.14.1. 새로운 기능

- OpenShift Serverless에서 Knative Serving 0.24.0을 사용합니다.
- OpenShift Serverless에서 Knative Eventing 0.24.0을 사용합니다.
- OpenShift Serverless에서 Kourier 0.24.0을 사용합니다.
- OpenShift Serverless에서 Knative(**kn**) CLI 0.24.0을 사용합니다.
- OpenShift Serverless에서 Knative Kafka 0.24.7을 사용합니다.

- **kn func** CLI 플러그인에서 **func** 0.18.0을 사용합니다.

- 향후 OpenShift Serverless 1.19.0 릴리스에서는 보안을 강화하기 위해 외부 경로의 URL 체계가 기본적으로 HTTPS로 설정됩니다.
  이러한 변경 사항을 워크로드에 적용하지 않으려면 다음 YAML을 **KnativeServing** CR(사용자 정의 리소스)에 추가하여 1.19.0으로 업그레이드하기 전에 기본 설정을 덮어쓸 수 있습니다.

```
...
spec:
  config:
    network:
      defaultExternalScheme: "http"
...
```

  변경 사항이 1.18.0에 이미 적용되려면 다음 YAML을 추가합니다.

```
...
spec:
  config:
    network:
      defaultExternalScheme: "https"
...
```

- 향후 OpenShift Serverless 1.19.0 릴리스에서는 Kourier 게이트웨이가 노출되는 기본 서비스 유형은 **LoadBalancer** 가 아닌 **ClusterIP** 가 됩니다.
  이러한 변경 사항을 워크로드에 적용하지 않으려면 다음 YAML을 **KnativeServing** CR(사용자 정의 리소스)에 추가하여 1.19.0으로 업그레이드하기 전에 기본 설정을 덮어쓸 수 있습니다.

```
...
spec:
  ingress:
    kourier:
      service-type: LoadBalancer
...
```

- 이제 OpenShift Serverless에서 **emptyDir** 볼륨을 사용할 수 있습니다. 자세한 내용은 Knative Serving에 대한 OpenShift Serverless 설명서를 참조하십시오.

- 이제 **kn func** 을 사용하여 함수를 생성할 때 rust 템플릿을 사용할 수 있습니다.

1.14.2. 해결된 문제

- Camel-K 이전 1.4 버전은 OpenShift Serverless 1.17.0과 호환되지 않았습니다. Camel-K의 문제가 해결되었으며 Camel-K 버전 1.4.1은 OpenShift Serverless 1.17.0과 함께 사용할 수 있습니다.

- 이전 버전에서는 Kafka 채널 또는 새 Kafka 소스에 대한 새 서브스크립션을 생성한 경우 Kafka 데이터 플레인에서 새로 생성된 서브스크립션 또는 싱크에서 준비 상태를 보고한 후 메시지를 디스패치할 준비가 되었습니다.
  결과적으로 데이터 플레인에서 준비 상태를 보고하지 않은 시간 동안 전송된 메시지가 구독자 또는 싱크로 전달되지 않았을 수 있습니다.

  OpenShift Serverless 1.18.0에서 문제가 해결되어 초기 메시지가 더 이상 손실되지 않습니다. 문제에 대한 자세한 내용은 기술 자료 문서 #6343981 을 참조하십시오.

1.14.3. 확인된 문제

- 이전 버전의 Knative **kn** CLI에서는 이전 버전의 Knative Serving 및 Knative Eventing API를 사용할 수 있습니다. 예를 들어 **kn** CLI 버전 0.23.2는 **v1alpha1** API 버전을 사용합니다. 반면 OpenShift Serverless의 최신 릴리스는 더 이상 이전 API 버전을 지원하지 않을 수 있습니다. 예를 들어 OpenShift Serverless 1.18.0에서는 **kafkasources.sources.knative.dev** API의 버전 **v1alpha1** 을 더 이상 지원하지 않습니다.

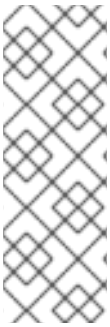
  결과적으로 최신 OpenShift Serverless와 함께 이전 버전의 Knative **kn** CLI를 사용하면 **kn** 에서 오래된 API를 찾을 수 없기 때문에 오류가 발생할 수 있습니다. 예를 들어 **kn** CLI의 버전 0.23.2는 OpenShift Serverless 1.18.0에서 작동하지 않습니다.

  문제를 방지하려면 OpenShift Serverless 릴리스에 사용 가능한 최신 **kn** CLI 버전을 사용하십시오. OpenShift Serverless 1.18.0의 경우 Knative **kn** CLI 0.24.0을 사용합니다.

2장. 서버리스 정보

2.1. OPENSHIFT SERVERLESS 개요

OpenShift Serverless에서는 개발자가 OpenShift Container Platform에서 서버리스 이벤트 중심 애플리케이션을 생성하고 배포할 수 있는 Kubernetes 기본 구성 블록을 제공합니다. OpenShift Serverless는 엔터프라이즈급 서버리스 플랫폼을 활성화하여 하이브리드 및 멀티 클라우드 환경에 대한 이식성과 일관성을 제공하는 오픈 소스 Knative 프로젝트를 기반으로 합니다.



참고

OpenShift Serverless는 OpenShift Container Platform과 다른 간격으로 릴리스되기 때문에 OpenShift Serverless 문서는 마이너 버전의 제품 버전에 대한 별도의 문서 세트를 유지 관리하지 않습니다. 현재 문서 세트는 특정 주제 또는 특정 기능에 대해 버전별 제한이 호출되지 않는 한 현재 지원되는 모든 OpenShift Serverless 버전에 적용됩니다.

OpenShift Serverless 라이프 사이클 및 지원되는 플랫폼에 대한 자세한 내용은 플랫폼 라이프 사이클 정책을 참조하십시오.

2.1.1. 추가 리소스

- 사용자 정의 리소스 정의를 사용하여 Kubernetes API 확장
- 사용자 정의 리소스 정의에서 리소스 관리
- 서버리스란 무엇인가?

2.2. KNATIVE SERVING

Knative Serving은 클라우드 네이티브 애플리케이션 을 생성, 배포 및 관리하려는 개발자를 지원합니다. OpenShift Container Platform 클러스터에서 서버리스 워크로드의 동작을 정의하고 제어하는 Kubernetes CRD(사용자 정의 리소스 정의)로 오브젝트 세트를 제공합니다.

개발자는 이러한 CRD를 사용하여 복잡한 사용 사례를 다룰 때 구성 요소로 사용할 수 있는 CR(사용자 정의 리소스) 인스턴스를 생성합니다. 예를 들면 다음과 같습니다.

- 신속한 서버리스 컨테이너 배포.
- Pod 자동 스케일링.

2.2.1. Knative Serving 리소스

서비스

service.serving.knative.dev CRD를 사용하면 워크로드의 라이프사이클을 자동으로 관리하여 네트워크를 통해 애플리케이션을 배포하고 연결할 수 있습니다. 이 CRD는 사용자 생성 서비스 또는 사용자 정의 리소스가 변경될 때마다 경로, 구성, 새 버전을 생성합니다. Knative의 개발자 상호 작용은 대부분 서비스 수정을 통해 수행됩니다.

버전

revision.serving.knative.dev CRD는 워크로드에 수행된 각 수정의 코드 및 구성에 대한 시점 스냅샷입니다. 버전은 변경할 수 없는 오브젝트이며 필요한 기간에 유지할 수 있습니다.

Route

route.serving.knative.dev CRD는 네트워크 끝점을 하나 이상의 버전에 매핑합니다. 부분 트래픽 및 이름이 지정된 경로를 포함하여 다양한 방법으로 트래픽을 관리할 수 있습니다.

설정

configuration.serving.knative.dev CRD는 배포에 필요한 상태를 유지 관리합니다. 코드와 구성을 명확하게 분리합니다. 구성을 수정하면 새 버전이 생성됩니다.

2.3. KNATIVE EVENTING

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

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

Knative Eventing에서는 다음 유스 케이스를 지원합니다.

소비자를 생성하지 않고 이벤트 게시

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

게시자를 생성하지 않고 이벤트 사용

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

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

주소 지정 가능 리소스

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

호출 가능한 리소스

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

2.3.1. Knative Kafka 사용

Knative Kafka는 OpenShift Serverless에서 지원되는 Apache Kafka 메시지 스트리밍 플랫폼을 사용할 수 있는 통합 옵션을 제공합니다. Kafka는 이벤트 소스, 채널, 브로커 및 이벤트 싱크 기능에 대한 옵션을 제공합니다.

참고

Knative Kafka는 현재 IBM Z 및 IBM Power에서 지원되지 않습니다.

Knative Kafka는 다음과 같은 추가 옵션을 제공합니다.

- Kafka 소스
- Kafka 채널

- Kafka 브로커
- Kafka 싱크

2.3.2. 추가 리소스

- **KnativeKafka** 사용자 정의 리소스 설치.
- Red Hat AMQ Streams 설명서
- Kafka 문서의 Red Hat AMQ Streams TLS 및 SASL
- 이벤트 전달

2.4. OPENSHIFT SERVERLESS FUNCTIONS 정보

OpenShift Serverless Functions를 사용하면 개발자가 상태 비저장 이벤트 중심 함수를 OpenShift Container Platform에서 Knative 서비스로 생성하고 배포할 수 있습니다. **kn func** CLI는 Knative **kn** CLI의 플러그인으로 제공됩니다. **kn func** CLI를 사용하여 컨테이너 이미지를 클러스터에서 Knative 서비스로 생성, 빌드, 배포할 수 있습니다.

2.4.1. 포함된 런타임

OpenShift Serverless Functions는 다음 런타임에 대한 기본 기능을 생성하는 데 사용할 수 있는 템플릿을 제공합니다.

- Quarkus
- Node.js
- TypeScript

2.4.2. 다음 단계

- 함수 시작하기.

3장. 서버리스 설치

3.1. OPENSHIFT SERVERLESS 설치 준비

OpenShift Serverless를 설치하기 전에 지원되는 구성 및 사전 요구 사항에 대한 다음 정보를 확인하십시오.

- OpenShift Serverless는 제한된 네트워크 환경에서 설치할 수 있습니다.
- OpenShift Serverless는 현재 단일 클러스터의 멀티 테넌트 구성에서 사용할 수 없습니다.

3.1.1. 지원되는 구성

OpenShift Serverless에 지원되는 일련의 기능, 구성, 통합과 현재 버전 및 이전 버전은 지원되는 구성 페이지에서 확인할 수 있습니다.

3.1.2. 확장 및 성능

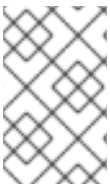
OpenShift Serverless는 3개의 주요 노드 및 3개의 작업자 노드 구성, 각각 64개의 CPU, 457GB 메모리, 각각 394GB의 스토리지를 사용하여 테스트되었습니다.

이 구성을 사용하여 생성할 수 있는 최대 Knative 서비스 수는 3000개입니다. 1개의 Knative 서비스에서 3개의 Kubernetes 서비스를 생성하기 때문에 OpenShift Container Platform Kubernetes 서비스 제한에 10,000 개에 해당합니다.

제로 응답 시간에서 평균 규모는 약 3.4초이며, 간단한 Quarkus 애플리케이션의 경우 최대 응답 시간인 8초, 4.5초의 99.9번째 백분위수입니다. 이러한 시간은 애플리케이션 및 애플리케이션 런타임에 따라 다를 수 있습니다.

3.1.3. 클러스터 크기 요구 사항 정의

OpenShift Serverless를 설치하고 사용하려면 OpenShift Container Platform 클러스터의 크기를 올바르게 조정해야 합니다.



참고

다음 요구 사항은 OpenShift Container Platform 클러스터의 작업자 머신 풀에만 관련이 있습니다. 컨트롤 플레인 노드는 일반 스케줄링에 사용되지 않으며 요구 사항에서 생략됩니다.

OpenShift Serverless를 사용하기 위한 최소 요구 사항은 CPU 10개와 40GB 메모리가 있는 클러스터입니다. 기본적으로 각 Pod는 ~400m의 CPU를 요청하므로 최소 요구 사항은 이 값을 기반으로 합니다.

OpenShift Serverless를 실행하는 총 크기 요구 사항은 설치된 구성 요소와 배포된 애플리케이션에 따라 다르며 배포에 따라 다를 수 있습니다.

3.1.4. 컴퓨팅 머신 세트를 사용하여 클러스터 스케일링

OpenShift Container Platform **MachineSet** API를 사용하여 클러스터를 원하는 크기로 수동으로 확장할 수 있습니다. 최소 요구 사항은 일반적으로 기본 컴퓨팅 머신 세트 중 하나를 두 개의 추가 머신으로 확장해야 함을 의미합니다. 컴퓨팅 머신 세트 수동 스케일링을 참조하십시오.

3.1.4.1. 고급 사용 사례에 대한 추가 요구 사항

OpenShift Container Platform에서 로깅 또는 미터링과 같은 고급 사용 사례의 경우 더 많은 리소스를 배포해야 합니다. 이러한 사용 사례에 권장되는 요구 사항은 CPU 24개 및 메모리 96GB입니다.

클러스터에 HA(고가용성)를 활성화하면 Knative Serving 컨트롤 플레인을 복제할 때마다 0.5 ~ 1.5코어 및 200MB ~ 2GB의 메모리가 필요합니다. HA는 기본적으로 일부 Knative Serving 구성 요소에 사용됩니다. "고가용성 복제본 구성"에 대한 문서에 따라 HA를 비활성화할 수 있습니다.

3.1.5. 추가 리소스

- 제한된 네트워크에서 Operator Lifecycle Manager 사용
- OperatorHub 이해

3.2. OPENSHIFT SERVERLESS OPERATOR 설치

OpenShift Serverless Operator를 설치하면 OpenShift Container Platform 클러스터에 Knative Serving, Knative Eventing, Knative Kafka를 설치하고 사용할 수 있습니다. OpenShift Serverless Operator는 클러스터의 Knative CRD(사용자 정의 리소스 정의)를 관리하며 각 구성 요소의 개별 구성 맵을 직접 수정하지 않고도 구성할 수 있습니다.

3.2.1. 웹 콘솔에서 OpenShift Serverless Operator 설치

OpenShift Container Platform 웹 콘솔을 사용하여 OperatorHub에서 OpenShift Serverless Operator를 설치할 수 있습니다. 이 Operator를 설치하면 Knative 구성 요소를 설치하고 사용할 수 있습니다.

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- 클러스터에 Marketplace 기능이 활성화되었거나 Red Hat Operator 카탈로그 소스가 수동으로 구성되어 있습니다.
- OpenShift Container Platform 웹 콘솔에 로그인했습니다.

절차

1. OpenShift Container Platform 웹 콘솔에서 **Operator → OperatorHub** 페이지로 이동합니다.
2. 스크롤하거나 **키워드로 필터링 상자**에 키워드 **서버리스**를 입력하여 OpenShift Serverless Operator를 찾습니다.
3. Operator에 대한 정보를 검토하고 **설치**를 클릭합니다.
4. **Operator 설치** 페이지에서 다음을 수행합니다.
   a. **설치 모드**는 **클러스터의 모든 네임스페이스(기본값)**입니다. 이 모드에서는 클러스터의 모든 네임스페이스를 감시하고 사용할 수 있도록 기본 **openshift-serverless** 네임스페이스에 Operator를 설치합니다.
   b. **설치된 네임스페이스**는 **openshift-serverless** 입니다.
   c. **stable** 채널을 **업데이트 채널**로 선택합니다. **stable** 채널을 사용하면 안정적인 최신 릴리스의 OpenShift Pipelines Operator를 설치할 수 있습니다.

d. **자동** 또는 **수동** 승인 전략을 선택합니다.

5. 이 OpenShift Container Platform 클러스터에서 선택한 네임스페이스에서 Operator를 사용할 수 있도록 하려면 **설치**를 클릭합니다.

6. **카탈로그 → Operator 관리** 페이지에서 OpenShift Serverless Operator 서브스크립션 설치 및 업그레이드 진행 상황을 모니터링할 수 있습니다.

   a. **수동** 승인 전략을 선택한 경우 설치 계획을 검토하고 승인할 때까지 서브스크립션의 업그레이드 상태가 **업그레이드 중**으로 유지됩니다. **설치 계획** 페이지에서 승인한 후 subscription 업그레이드 상태가 **최신**으로 이동합니다.

   b. **자동** 승인 전략을 선택한 경우 업그레이드 상태가 개입 없이 **최신** 상태로 확인되어야 합니다.

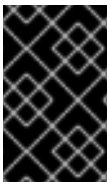
검증

서브스크립션의 업그레이드 상태가 **최신**인 경우 **카탈로그 → 설치된 Operator**를 선택하여 OpenShift Serverless Operator가 표시되고 관련 네임스페이스에서 **상태**가 최종적으로 **InstallSucceeded**로 표시되는지 확인합니다.

그러지 않은 경우 다음을 수행합니다.

1. **카탈로그 → Operator 관리** 페이지로 전환한 후 **Operator 서브스크립션** 및 **설치 계획** 탭의 **상태**에 장애나 오류가 있는지 검사합니다.

2. **워크로드 → Pod** 페이지의 **openshift-serverless** 프로젝트에서 문제를 보고하는 Pod의 로그를 확인하여 추가로 문제를 해결합니다.



중요

OpenShift Serverless에서 Red Hat OpenShift distributed tracing을 사용하려면 Knative Serving 또는 Knative Eventing을 설치하기 전에 Red Hat OpenShift distributed tracing을 설치하고 구성해야 합니다.

3.2.2. CLI에서 OpenShift Serverless Operator 설치

CLI를 사용하여 OperatorHub에서 OpenShift Serverless Operator를 설치할 수 있습니다. 이 Operator를 설치하면 Knative 구성 요소를 설치하고 사용할 수 있습니다.

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- 클러스터에 Marketplace 기능이 활성화되었거나 Red Hat Operator 카탈로그 소스가 수동으로 구성되어 있습니다.
- OpenShift Container Platform 클러스터에 로그인했습니다.

절차

1. **Namespace**,**OperatorGroup**, **Subscription** 오브젝트가 포함된 YAML 파일을 생성하여 OpenShift Serverless Operator에 네임스페이스를 서브스크립션합니다. 예를 들어 다음 콘텐츠를 사용하여 **서버리스-subscription.yaml** 파일을 생성합니다.

   서브스크립션 예

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-serverless
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: serverless-operators
  namespace: openshift-serverless
spec: {}
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: serverless-operator
  namespace: openshift-serverless
spec:
  channel: stable (1)
  name: serverless-operator (2)
  source: redhat-operators (3)
  sourceNamespace: openshift-marketplace (4)
```

1. Operator의 채널 이름입니다. **stable** 채널을 사용하면 OpenShift Serverless Operator의 최신 안정적인 버전을 설치할 수 있습니다.
2. 등록할 Operator의 이름입니다. OpenShift Serverless Operator의 경우 항상 **서버리스 운영자** 입니다.
3. Operator를 제공하는 CatalogSource의 이름입니다. 기본 OperatorHub 카탈로그 소스에 **redhat-operators** 를 사용합니다.
4. CatalogSource의 네임스페이스입니다. 기본 OperatorHub 카탈로그 소스에는 **openshift-marketplace**를 사용합니다.

2. **Subscription** 오브젝트를 생성합니다.

```
$ oc apply -f serverless-subscription.yaml
```

검증

CSV(클러스터 서비스 버전)가 **Succeeded** 단계에 도달했는지 확인합니다.

명령 예

```
$ oc get csv
```

출력 예

```
NAME                  DISPLAY                       VERSION   REPLACES              PHASE
serverless-operator.v1.25.0   Red Hat OpenShift Serverless   1.25.0    serverless-operator.v1.24.0
Succeeded
```

중요

OpenShift Serverless에서 Red Hat OpenShift distributed tracing을 사용하려면 Knative Serving 또는 Knative Eventing을 설치하기 전에 Red Hat OpenShift distributed tracing을 설치하고 구성해야 합니다.

3.2.3. 글로벌 구성

OpenShift Serverless Operator는 **KnativeServing** 및 **KnativeEventing** 사용자 정의 리소스의 값을 시스템 구성 맵에 표시하는 등 Knative 설치의 글로벌 구성을 관리합니다. 수동으로 적용되는 구성 맵에 대한 모든 업데이트는 Operator에서 덮어씁니다. 그러나 Knative 사용자 정의 리소스를 수정하면 이러한 구성 맵에 대한 값을 설정할 수 있습니다.

Knative에는 접두사 **config-** 를 사용하여 이름이 지정된 여러 구성 맵이 있습니다. 모든 Knative 구성 맵은 적용되는 사용자 정의 리소스와 동일한 네임스페이스에 생성됩니다. 예를 들어 **knative-serving** 네임스페이스에 **KnativeServing** 사용자 정의 리소스가 생성되는 경우 이 네임스페이스에 모든 Knative Serving 구성 맵도 생성됩니다.

Knative 사용자 정의 리소스의 **spec.config** 에는 **config-< name >이라는 각 구성 맵에 대해 하나의 <name >** 항목이 있으며 구성 맵 **데이터에** 사용되는 값이 있습니다.

3.2.4. 추가 리소스

- 사용자 정의 리소스 정의에서 리소스 관리
- 영구저장장치 이해
- 사용자 정의 PKI 구성

3.2.5. 다음 단계

- OpenShift Serverless Operator가 설치되면 Knative Serving을 설치하거나 Knative Eventing을 설치할 수 있습니다.

3.3. KNATIVE CLI 설치

Knative(**kn**) CLI에는 자체 로그인 메커니즘이 없습니다. 클러스터에 로그인하려면 OpenShift CLI(**oc**)를 설치하고 **oc login** 명령을 사용해야 합니다. CLI의 설치 옵션은 운영 체제에 따라 다를 수 있습니다.

운영 체제에 대한 OpenShift CLI(**oc**)를 설치하고 **oc** 로 로그인하는 방법에 대한 자세한 내용은 OpenShift CLI 시작하기 설명서를 참조하십시오.

OpenShift Serverless는 Knative(**kn**) CLI를 사용하여 설치할 수 없습니다. 클러스터 관리자는 OpenShift Serverless Operator 설치 설명서에 설명된 대로 OpenShift Serverless Operator 를 설치하고 Knative 구성 요소를 설정해야 합니다.

중요

최신 OpenShift Serverless 릴리스에서 이전 버전의 Knative(**kn**) CLI를 사용하려는 경우 API를 찾을 수 없으며 오류가 발생합니다.

예를 들어 Knative Serving 및 Knative Eventing API의 1.3 버전을 사용하는 1.24.0 OpenShift Serverless 릴리스에서 버전 1.2를 사용하는 Knative(**kn**) CLI의 1.23.0 릴리스를 사용하는 경우 이전 1.2 API 버전을 계속 찾기 때문에 CLI가 작동하지 않습니다.

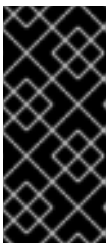
문제를 방지하려면 OpenShift Serverless 릴리스에 최신 Knative(**kn**) CLI 버전을 사용하고 있는지 확인합니다.

3.3.1. OpenShift Container Platform 웹 콘솔을 통한 Knative CLI 설치

OpenShift Container Platform 웹 콘솔을 사용하면 Knative(**kn**) CLI를 설치하기 위해 간소화되고 직관적인 사용자 인터페이스가 제공됩니다. OpenShift Serverless Operator가 설치되면 OpenShift Container Platform 웹 콘솔의 **명령줄 툴** 페이지에서 Linux(amd64, s390x, ppc64le), macOS 또는 Windows용 Knative(**kn**) CLI를 다운로드할 수 있는 링크가 표시됩니다.

사전 요구 사항

- OpenShift Container Platform 웹 콘솔에 로그인했습니다.
- OpenShift Serverless Operator 및 Knative Serving이 OpenShift Container Platform 클러스터에 설치되어 있습니다.



중요

libc를 사용할 수 없는 경우 CLI 명령을 실행할 때 다음과 같은 오류가 표시될 수 있습니다.

```
$ kn: No such file or directory
```

- 이 절차에 대한 확인 단계를 사용하려면 OpenShift (**oc**) CLI를 설치해야 합니다.

절차

1. **명령줄 툴** 페이지에서 Knative(**kn**) CLI를 다운로드합니다. 웹 콘솔의 오른쪽 상단에 있는  아이콘을 클릭하고 목록에서 **명령줄 툴을 선택하여 명령줄** 툴에 액세스할 수 있습니다.
2. 아카이브의 압축을 풉니다.

```
$ tar -xf <file>
```

3. **kn** 바이너리를 **PATH**의 디렉터리로 이동합니다.
4. **PATH**를 확인하려면 다음을 실행합니다.

```
$ echo $PATH
```

검증

- 다음 명령을 실행하여 올바른 Knative CLI 리소스 및 경로가 생성되었는지 확인합니다.

```
$ oc get ConsoleCLIDownload
```

출력 예

```
NAME                 DISPLAY NAME                                     AGE
kn                   kn - OpenShift Serverless Command Line Interface (CLI)   2022-09-
20T08:41:18Z
oc-cli-downloads     oc - OpenShift Command Line Interface (CLI)              2022-09-
20T08:00:20Z
```

```
$ oc get route -n openshift-serverless
```

출력 예

```
NAME   HOST/PORT                                PATH   SERVICES                  PORT
TERMINATION     WILDCARD
kn     kn-openshift-serverless.apps.example.com          knative-openshift-metrics-3   http-cli
edge/Redirect   None
```

3.3.2. RPM 패키지 관리자를 사용하여 Linux용 Knative CLI 설치

RHEL(Red Hat Enterprise Linux)의 경우 **yum** 또는 **dnf** 와 같은 패키지 관리자를 사용하여 Knative(**kn**) CLI를 RPM으로 설치할 수 있습니다. 이를 통해 시스템에서 Knative CLI 버전을 자동으로 관리할 수 있습니다. 예를 들어, **dnf 업그레이드** 와 같은 명령을 사용하면 새 버전이 사용 가능한 경우 **kn** 을 포함한 모든 패키지를 업그레이드합니다.

사전 요구 사항

- Red Hat 계정에 활성 OpenShift Container Platform 서브스크립션이 있어야 합니다.

절차

1. Red Hat Subscription Manager에 등록합니다.

```
# subscription-manager register
```

2. 최신 서브스크립션 데이터를 가져옵니다.

```
# subscription-manager refresh
```

3. 서브스크립션을 등록된 시스템에 연결합니다.

```
# subscription-manager attach --pool=<pool_id> 1
```

1 활성 OpenShift Container Platform 서브스크립션의 풀 ID

4. Knative(**kn**) CLI에 필요한 리포지토리를 활성화합니다.
   - Linux (x86_64, amd64)

```
# subscription-manager repos --enable="openshift-serverless-1-for-rhel-8-x86_64-rpms"
```

-

- Linux on IBM Z 및 LinuxONE(s390x)

```
# subscription-manager repos --enable="openshift-serverless-1-for-rhel-8-s390x-rpms"
```

- Linux on IBM Power(ppc64le)

```
# subscription-manager repos --enable="openshift-serverless-1-for-rhel-8-ppc64le-rpms"
```

5. 패키지 관리자를 사용하여 Knative(**kn**) CLI를 RPM으로 설치합니다.

yum 명령 예

```
# yum install openshift-serverless-clients
```

3.3.3. Linux의 Knative CLI 설치

RPM 또는 다른 패키지 관리자가 설치되지 않은 Linux 배포를 사용하는 경우 Knative(**kn**) CLI를 바이너리 파일로 설치할 수 있습니다. 이렇게 하려면 **tar.gz** 아카이브를 다운로드하여 압축을 풀고 **PATH** 의 디렉터리에 바이너리를 추가해야 합니다.

사전 요구 사항

- RHEL 또는 Fedora를 사용하지 않는 경우 **libc** 가 라이브러리 경로의 디렉터리에 설치되어 있는지 확인합니다.

중요

libc를 사용할 수 없는 경우 CLI 명령을 실행할 때 다음과 같은 오류가 표시될 수 있습니다.

```
$ kn: No such file or directory
```

절차

1. 관련 Knative (**kn**) CLI **tar.gz** 아카이브를 다운로드합니다.
   - Linux (x86_64, amd64)
   - Linux on IBM Z 및 LinuxONE(s390x)
   - Linux on IBM Power(ppc64le)

   Serverless 클라이언트 다운로드 미러에서 해당 버전의 해당 디렉터리로 이동하여 **kn** 버전을 다운로드할 수도 있습니다.

2. 아카이브의 압축을 풉니다.

```
$ tar -xf <filename>
```

3. **kn** 바이너리를 **PATH**의 디렉터리로 이동합니다.
4. **PATH**를 확인하려면 다음을 실행합니다.

```
$ echo $PATH
```

3.3.4. macOS용 Knative CLI 설치

macOS를 사용하는 경우 Knative(**kn**) CLI를 바이너리 파일로 설치할 수 있습니다. 이렇게 하려면 **tar.gz** 아카이브를 다운로드하여 압축을 풀고 **PATH** 의 디렉터리에 바이너리를 추가해야 합니다.

절차

1. Knative(**kn**) CLI **tar.gz** 아카이브를 다운로드합니다.
   Serverless 클라이언트 다운로드 미러에서 해당 버전의 해당 디렉터리로 이동하여 **kn** 버전을 다운로드할 수도 있습니다.
2. 아카이브의 압축을 해제하고 추출합니다.
3. **kn** 바이너리를 **PATH**의 디렉터리로 이동합니다.
4. **PATH**를 확인하려면 터미널 창을 열고 다음을 실행합니다.

```
$ echo $PATH
```

3.3.5. Windows용 Knative CLI 설치

Windows를 사용하는 경우 Knative(**kn**) CLI를 바이너리 파일로 설치할 수 있습니다. 이렇게 하려면 ZIP 아카이브를 다운로드하여 압축 해제하고 **PATH** 의 디렉터리에 바이너리를 추가해야 합니다.

절차

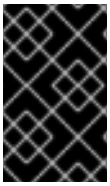
1. Knative(**kn**) CLI ZIP 아카이브를 다운로드합니다.
   Serverless 클라이언트 다운로드 미러에서 해당 버전의 해당 디렉터리로 이동하여 **kn** 버전을 다운로드할 수도 있습니다.
2. ZIP 프로그램으로 아카이브의 압축을 풉니다.
3. **kn** 바이너리를 **PATH**의 디렉터리로 이동합니다.
4. **PATH**를 확인하려면 명령 프롬프트를 열고 다음 명령을 실행합니다.

```
C:\> path
```

3.4. KNATIVE SERVING 설치

Knative Serving을 설치하면 클러스터에서 Knative 서비스 및 기능을 생성할 수 있습니다. 또한 애플리케이션에 대한 자동 스케일링 및 네트워킹 옵션과 같은 추가 기능을 사용할 수 있습니다.

OpenShift Serverless Operator를 설치한 후 기본 설정을 사용하여 **Knative Serving** 을 설치하거나 KnativeServing CR(사용자 정의 리소스)에서 고급 설정을 구성할 수 있습니다. **KnativeServing** CR의 구성 옵션에 대한 자세한 내용은 글로벌 구성을 참조하십시오.



중요

OpenShift Serverless에서 Red Hat OpenShift distributed tracing을 사용하려면 Knative Serving을 설치하기 전에 Red Hat OpenShift distributed tracing을 설치하고 구성해야 합니다.

3.4.1. 웹 콘솔을 사용하여 Knative Serving 설치

OpenShift Serverless Operator를 설치한 후 OpenShift Container Platform 웹 콘솔을 사용하여 Knative Serving을 설치합니다. 기본 설정을 사용하여 **Knative Serving** 을 설치하거나 KnativeServing CR(사용자 정의 리소스)에서 고급 설정을 구성할 수 있습니다.

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- OpenShift Container Platform 웹 콘솔에 로그인했습니다.
- OpenShift Serverless Operator를 설치했습니다.

절차

1. OpenShift Container Platform 웹 콘솔의 **관리자** 화면에서 **Operator → 설치된 Operator**로 이동합니다.
2. 페이지 상단에 있는 **프로젝트** 드롭다운이 **Project: knative-serving**으로 설정되어 있는지 확인합니다.
3. OpenShift Serverless Operator의 **제공되는 API** 목록에서 **Knative Serving**을 클릭하여 **Knative Serving** 탭으로 이동합니다.
4. **Knative Serving 생성을** 클릭합니다.
5. **Knative Serving 생성** 페이지에서 **생성**을 클릭하면 기본 설정을 사용하여 Knative Serving을 설치할 수 있습니다.
   제공된 양식을 사용하여 **KnativeServing** 오브젝트를 편집하거나 YAML을 편집하여 Knative Serving 설치에 대한 설정을 수정할 수도 있습니다.
   - 해당 양식은 **KnativeServing** 오브젝트 생성을 완전히 제어할 필요가 없는 단순한 구성에 사용하는 것이 좋습니다.
   - **KnativeServing** 오브젝트 생성을 완전히 제어해야 하는 복잡한 구성의 경우 YAML을 편집하는 것이 좋습니다. **Knative Serving 생성** 페이지의 오른쪽 상단에 있는 **YAML 편집** 링크를 클릭하여 YAML에 액세스할 수 있습니다.
   양식을 작성하거나 YAML을 수정한 후 **생성**을 클릭합니다.

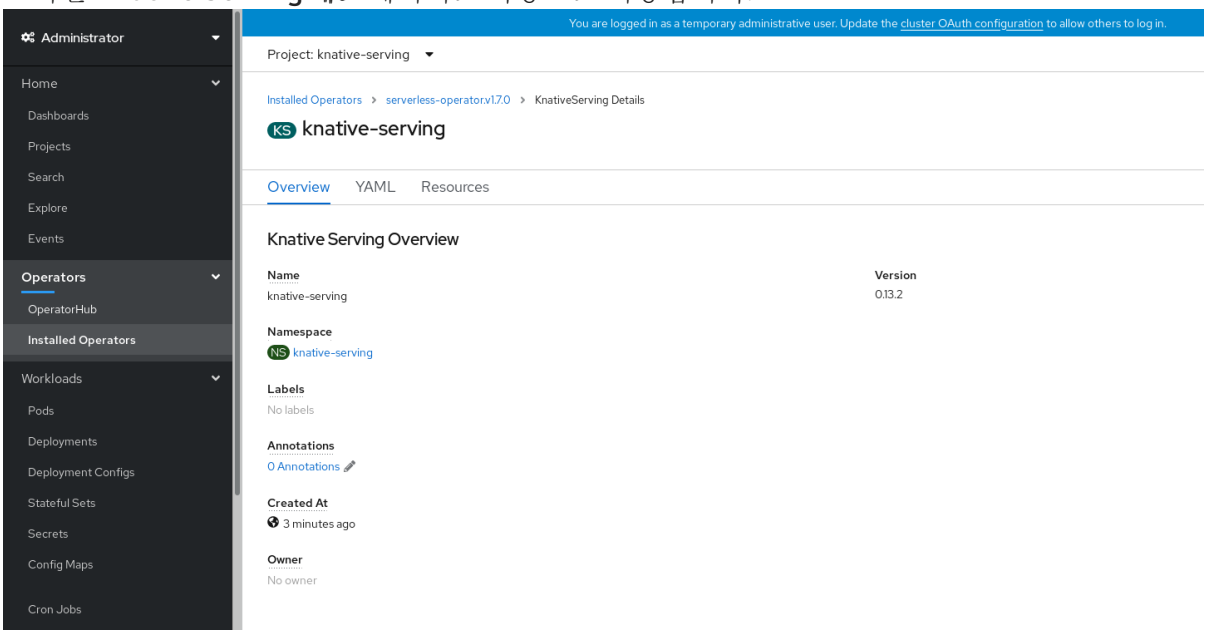
   

   참고

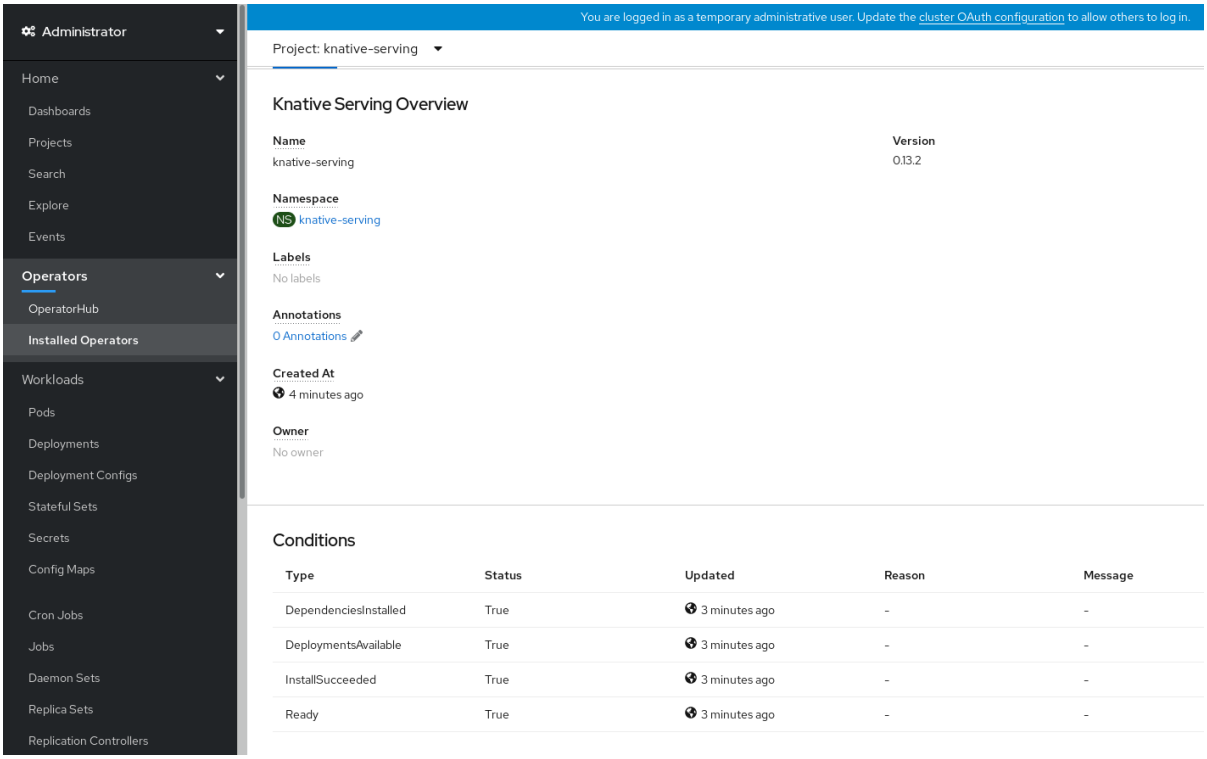
   KnativeServing 사용자 정의 리소스 정의의 구성 옵션에 대한 자세한 내용은 *고급 설치 구성 옵션* 설명서를 참조하십시오.
6. Knative Serving을 설치한 후에는 **KnativeServing** 오브젝트가 생성되고 **Knative Serving** 탭으로 자동으로 이동합니다. 리소스 목록에 **knative-serving** 사용자 정의 리소스가 표시됩니다.

검증

1. **Knative Serving** 탭에서 **knative-serving** 사용자 정의 리소스를 클릭합니다.
2. 그러면 **Knative Serving 개요** 페이지로 자동으로 이동합니다.



3. **조건** 목록을 보려면 아래로 스크롤합니다.
4. 예제 이미지에 표시된 대로 상태가 **True**인 조건 목록이 표시되어야 합니다.



참고

Knative Serving 리소스를 생성하는 데 몇 초가 걸릴 수 있습니다. **리소스** 탭에서 해당 상태를 확인할 수 있습니다.

5. 조건이 **알 수 없음** 또는 **False** 상태인 경우 몇 분 정도 기다렸다가 리소스가 생성된 것을 확인한 후 다시 확인하십시오.

3.4.2. YAML을 사용하여 Knative Serving 설치

OpenShift Serverless Operator를 설치한 후 기본 설정을 사용하여 **Knative Serving** 을 설치하거나 KnativeServing CR(사용자 정의 리소스)에서 고급 설정을 구성할 수 있습니다. 다음 절차에 따라 YAML 파일 및 **oc** CLI를 사용하여 Knative Serving을 설치할 수 있습니다.

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- OpenShift Serverless Operator를 설치했습니다.
- OpenShift CLI(**oc**)를 설치합니다.

절차

1. **serving.yaml**이라는 파일을 생성하고 다음 예제 YAML을 이 파일에 복사합니다.

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
```

2. **serving.yaml** 파일을 적용합니다.

```
$ oc apply -f serving.yaml
```

검증

1. 설치가 완료되었는지 확인하려면 다음 명령을 입력합니다.

```
$ oc get knativeserving.operator.knative.dev/knative-serving -n knative-serving --
template='{{range .status.conditions}}{{printf "%s=%s\n" .type .status}}{{end}}'
```

출력 예

```
DependenciesInstalled=True
DeploymentsAvailable=True
InstallSucceeded=True
Ready=True
```

참고

Knative Serving 리소스를 생성하는 데 몇 초가 걸릴 수 있습니다.

조건이 **알 수 없음** 또는 **False** 상태인 경우 몇 분 정도 기다렸다가 리소스가 생성된 것을 확인한 후 다시 확인하십시오.

2. Knative Serving 리소스가 생성되었는지 확인합니다.

```
$ oc get pods -n knative-serving
```

-

출력 예

```
NAME                                                READY   STATUS      RESTARTS   AGE
activator-67ddf8c9d7-p7rm5                          2/2     Running     0          4m
activator-67ddf8c9d7-q84fz                          2/2     Running     0          4m
autoscaler-5d87bc6dbf-6nqc6                         2/2     Running     0          3m59s
autoscaler-5d87bc6dbf-h64rl                         2/2     Running     0          3m59s
autoscaler-hpa-77f85f5cc4-lrts7                     2/2     Running     0          3m57s
autoscaler-hpa-77f85f5cc4-zx7hl                     2/2     Running     0          3m56s
controller-5cfc7cb8db-nlccl                         2/2     Running     0          3m50s
controller-5cfc7cb8db-rmv7r                         2/2     Running     0          3m18s
domain-mapping-86d84bb6b4-r746m                     2/2     Running     0          3m58s
domain-mapping-86d84bb6b4-v7nh8                     2/2     Running     0          3m58s
domainmapping-webhook-769d679d45-bkcnj              2/2     Running     0          3m58s
domainmapping-webhook-769d679d45-fff68              2/2     Running     0          3m58s
storage-version-migration-serving-serving-0.26.0--1-6qlkb   0/1     Completed   0          3m56s
webhook-5fb774f8d8-6bqrt                            2/2     Running     0          3m57s
webhook-5fb774f8d8-b8lt5                            2/2     Running     0          3m57s
```

3. 필요한 네트워킹 구성 요소가 자동으로 생성된 **knative-serving-ingress** 네임스페이스에 설치되었는지 확인합니다.

```
$ oc get pods -n knative-serving-ingress
```

출력 예

```
NAME                                    READY   STATUS    RESTARTS   AGE
net-kourier-controller-7d4b6c5d95-62mkf   1/1     Running   0          76s
net-kourier-controller-7d4b6c5d95-qmgm2   1/1     Running   0          76s
3scale-kourier-gateway-6688b49568-987qz   1/1     Running   0          75s
3scale-kourier-gateway-6688b49568-b5tnp   1/1     Running   0          75s
```

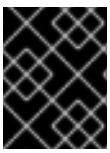
3.4.3. 다음 단계

- Knative 이벤트 중심 아키텍처를 사용하려면 Knative Eventing을 설치할 수 있습니다.

3.5. KNATIVE EVENTING 설치

클러스터에서 이벤트 중심 아키텍처를 사용하려면 Knative Eventing을 설치합니다. 이벤트 소스, 브로커 및 채널과 같은 Knative 구성 요소를 생성한 다음 이를 사용하여 이벤트 또는 외부 시스템으로 이벤트를 전송할 수 있습니다.

OpenShift Serverless Operator를 설치한 후 기본 설정을 사용하여 Knative Eventing을 설치하거나 **KnativeEventing** CR(사용자 정의 리소스)에서 고급 설정을 구성할 수 있습니다. **KnativeEventing** CR의 구성 옵션에 대한 자세한 내용은 글로벌 구성을 참조하십시오.



중요

OpenShift Serverless에서 Red Hat OpenShift 분산 추적을 사용하려면 Knative Eventing을 설치하기 전에 Red Hat OpenShift distributed tracing을 설치하고 구성해야 합니다.

3.5.1. 웹 콘솔을 사용하여 Knative Eventing 설치

OpenShift Serverless Operator를 설치한 후 OpenShift Container Platform 웹 콘솔을 사용하여 Knative Eventing을 설치합니다. 기본 설정을 사용하여 Knative Eventing을 설치하거나 **KnativeEventing** CR(사용자 정의 리소스)에서 고급 설정을 구성할 수 있습니다.

사전 요구 사항

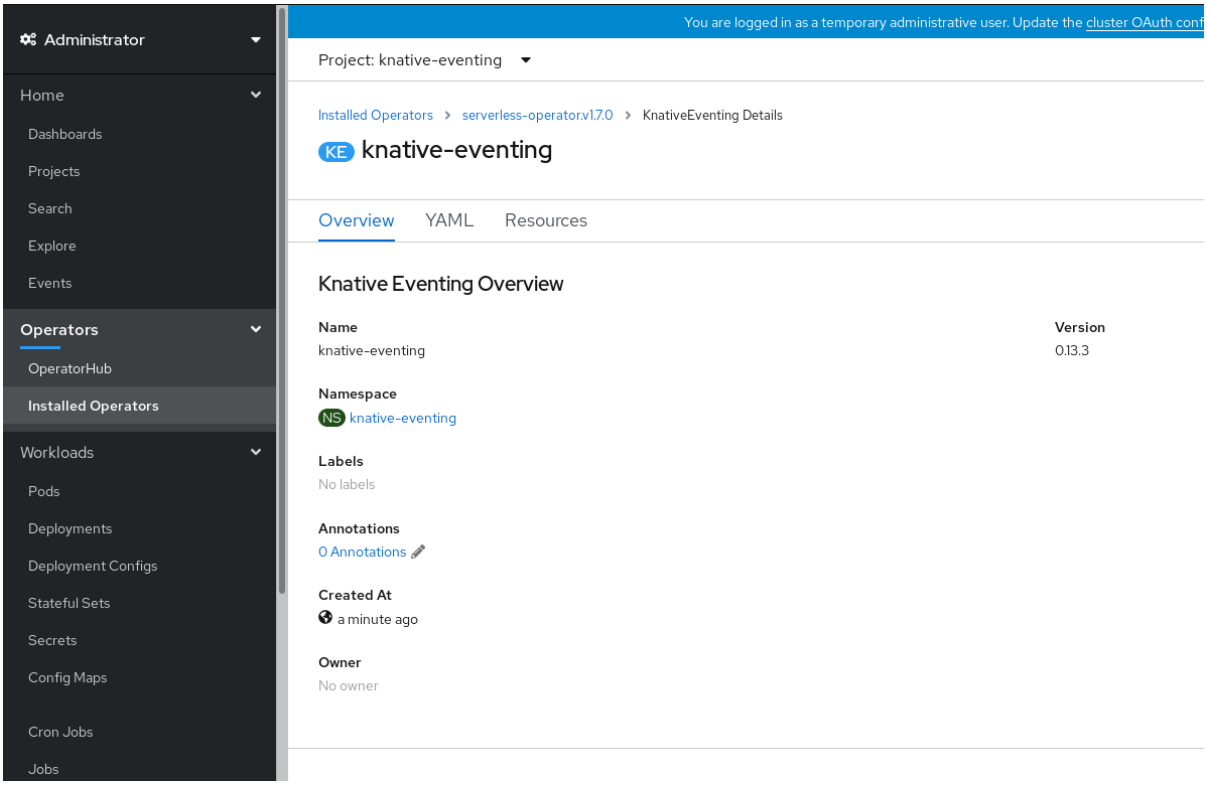
- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- OpenShift Container Platform 웹 콘솔에 로그인했습니다.
- OpenShift Serverless Operator를 설치했습니다.

절차

1. OpenShift Container Platform 웹 콘솔의 **관리자** 화면에서 **Operator → 설치된 Operator**로 이동합니다.
2. 페이지 상단에 있는 **프로젝트** 드롭다운이 **Project: knative-eventing**으로 설정되어 있는지 확인합니다.
3. OpenShift Serverless Operator의 **제공되는 API** 목록에서 **Knative Eventing**을 클릭하여 **Knative Eventing** 탭으로 이동합니다.
4. **Knative Eventing 생성을** 클릭합니다.
5. **Knative Eventing 생성** 페이지에서 제공된 기본 양식을 사용하거나 YAML을 편집하여 **KnativeEventing** 오브젝트를 구성하도록 선택할 수 있습니다.
   - 해당 양식은 **KnativeEventing** 오브젝트 생성을 완전히 제어할 필요가 없는 단순한 구성에 사용하는 것이 좋습니다.
     선택 사항입니다. 양식을 사용하여 **KnativeEventing** 오브젝트를 구성하는 경우 Knative Eventing 배포에 구현하려는 변경을 수행합니다.
6. **생성**을 클릭합니다.
   - **KnativeEventing** 오브젝트 생성을 완전히 제어해야 하는 복잡한 구성의 경우 YAML을 편집하는 것이 좋습니다. **Knative Eventing 생성** 페이지의 오른쪽 상단에 있는 **YAML 편집** 링크를 클릭하여 YAML에 액세스할 수 있습니다.
     선택 사항입니다. YAML을 편집하여 **KnativeEventing** 오브젝트를 구성하는 경우 YAML에 Knative Eventing 배포에 구현하려는 변경을 수행합니다.
7. **생성**을 클릭합니다.
8. Knative Eventing을 설치한 후에는 **KnativeEventing** 오브젝트가 생성되고 **Knative Eventing** 탭으로 자동으로 이동합니다. 리소스 목록에 **knative-eventing** 사용자 정의 리소스가 표시됩니다.

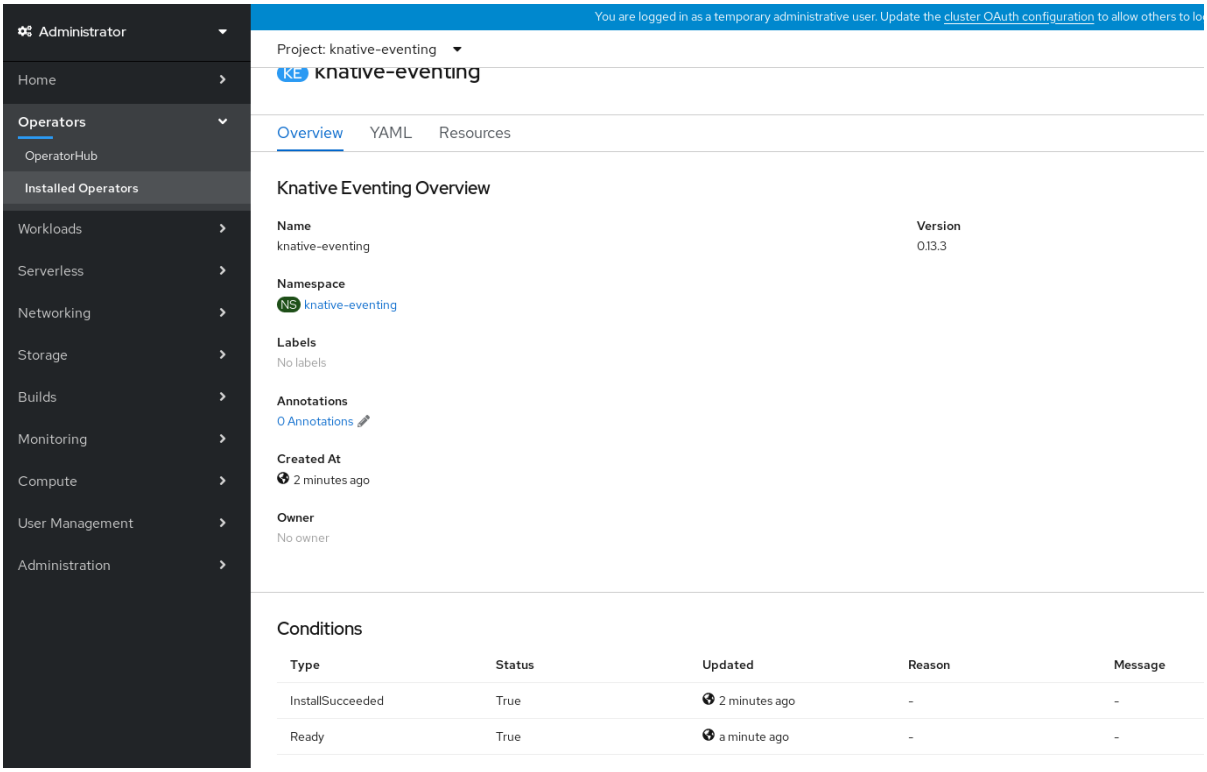
검증

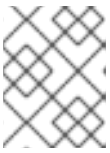
1. **Knative Eventing** 탭에서 **knative-eventing** 사용자 정의 리소스를 클릭합니다.
2. 그러면 자동으로 **Knative Eventing 개요** 페이지로 이동합니다.



3. **조건** 목록을 보려면 아래로 스크롤합니다.

4. 예제 이미지에 표시된 대로 상태가 **True**인 조건 목록이 표시되어야 합니다.





참고

Knative Eventing 리소스를 생성하는 데 몇 초가 걸릴 수 있습니다. **리소스** 탭에서 해당 상태를 확인할 수 있습니다.

5. 조건이 **알 수 없음** 또는 **False** 상태인 경우 몇 분 정도 기다렸다가 리소스가 생성된 것을 확인한 후 다시 확인하십시오.

3.5.2. YAML을 사용하여 Knative Eventing 설치

OpenShift Serverless Operator를 설치한 후 기본 설정을 사용하여 Knative Eventing을 설치하거나 **KnativeEventing** CR(사용자 정의 리소스)에서 고급 설정을 구성할 수 있습니다. 다음 절차에 따라 YAML 파일 및 **oc** CLI를 사용하여 Knative Eventing을 설치할 수 있습니다.

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- OpenShift Serverless Operator를 설치했습니다.
- OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. **eventing.yaml**이라는 파일을 생성합니다.
2. 다음 샘플 YAML을 **eventing.yaml**에 복사합니다.

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

3. 선택 사항입니다. Knative Eventing 배포를 위해 구현하려는 YAML을 변경합니다.
4. 다음을 입력하여 **eventing.yaml** 파일을 적용합니다.

```
$ oc apply -f eventing.yaml
```

검증

1. 다음 명령을 입력하고 출력을 관찰하여 설치가 완료되었는지 확인합니다.

```
$ oc get knativeeventing.operator.knative.dev/knative-eventing \
  -n knative-eventing \
  --template='{{range .status.conditions}}{{printf "%s=%s\n" .type .status}}{{end}}'
```

출력 예

```
InstallSucceeded=True
Ready=True
```

참고

Knative Eventing 리소스를 생성하는 데 몇 초가 걸릴 수 있습니다.

2. 조건이 **알 수 없음** 또는 **False** 상태인 경우 몇 분 정도 기다렸다가 리소스가 생성된 것을 확인한 후 다시 확인하십시오.

3. 다음을 입력하여 Knative Eventing 리소스가 생성되었는지 확인합니다.

```
$ oc get pods -n knative-eventing
```

출력 예

```
NAME                                  READY   STATUS    RESTARTS   AGE
broker-controller-58765d9d49-g9zp6    1/1     Running   0          7m21s
eventing-controller-65fdd66b54-jw7bh  1/1     Running   0          7m31s
eventing-webhook-57fd74b5bd-kvhlz     1/1     Running   0          7m31s
imc-controller-5b75d458fc-ptvm2       1/1     Running   0          7m19s
imc-dispatcher-64f6d5fccb-kkc4c       1/1     Running   0          7m18s
```

3.5.3. Knative Kafka 설치

Knative Kafka는 OpenShift Serverless에서 지원되는 Apache Kafka 메시지 스트리밍 플랫폼을 사용할 수 있는 통합 옵션을 제공합니다. Knative Kafka 기능은 **KnativeKafka** 사용자 정의 리소스를 설치한 경우 OpenShift Serverless 설치에서 사용할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing을 클러스터에 설치했습니다.
- Red Hat AMQ Streams 클러스터에 액세스할 수 있습니다.
- 확인 단계를 사용하려면 OpenShift CLI(**oc**)를 설치합니다.
- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있습니다.
- OpenShift Container Platform 웹 콘솔에 로그인되어 있습니다.

절차

1. **관리자** 화면에서 **Operator → 설치된 Operator**로 이동합니다.
2. 페이지 상단에 있는 **프로젝트** 드롭다운이 **Project: knative-eventing**으로 설정되어 있는지 확인합니다.
3. OpenShift Serverless Operator의 **Provided APIs** 목록에서 **Knative Kafka**를 검색하여 **Create Instance**를 클릭합니다.
4. **Knative Kafka 생성** 페이지에서 **KnativeKafka** 오브젝트를 구성합니다.

중요

클러스터에서 Kafka 채널, 소스, 브로커 또는 싱크를 사용하려면 사용할 옵션에 대해 **활성화된** 스위치를 **true** 로 전환해야 합니다. 이러한 스위치는 기본적으로 **false**로 설정됩니다. 또한 Kafka 채널, 브로커 또는 싱크를 사용하려면 부트스트랩 서버를 지정해야 합니다.

KnativeKafka 사용자 정의 리소스의 예

```
apiVersion: operator.serverless.openshift.io/v1alpha1
```

```
kind: KnativeKafka
metadata:
  name: knative-kafka
  namespace: knative-eventing
spec:
  channel:
    enabled: true (1)
    bootstrapServers: <bootstrap_servers> (2)
  source:
    enabled: true (3)
  broker:
    enabled: true (4)
    defaultConfig:
      bootstrapServers: <bootstrap_servers> (5)
      numPartitions: <num_partitions> (6)
      replicationFactor: <replication_factor> (7)
  sink:
    enabled: true (8)
```

1. 개발자가 클러스터에서 **KafkaChannel** 채널 유형을 사용할 수 있습니다.
2. AMQ Streams 클러스터에 있는 쉼표로 구분된 부트스트랩 서버 목록입니다.
3. 개발자가 클러스터에서 **KafkaSource** 이벤트 소스 유형을 사용할 수 있습니다.
4. 개발자가 클러스터에서 Knative Kafka 브로커 구현을 사용할 수 있습니다.
5. Red Hat AMQ Streams 클러스터의 쉼표로 구분된 부트스트랩 서버 목록입니다.
6. **Broker** 오브젝트에서 지원하는 Kafka 주제의 파티션 수를 정의합니다. 기본값은 **10**입니다.
7. **Broker** 오브젝트에서 지원하는 Kafka 항목의 복제 요소를 정의합니다. 기본값은 **3** 입니다.
8. 개발자가 클러스터에서 Kafka 싱크를 사용할 수 있습니다.

참고

replicationfactor 값은 Red Hat AMQ Streams 클러스터의 노드 수보다 작거나 같아야 합니다.

a. 해당 양식은 **KnativeKafka** 오브젝트 생성을 완전히 제어할 필요가 없는 단순한 구성에 사용하는 것이 좋습니다.

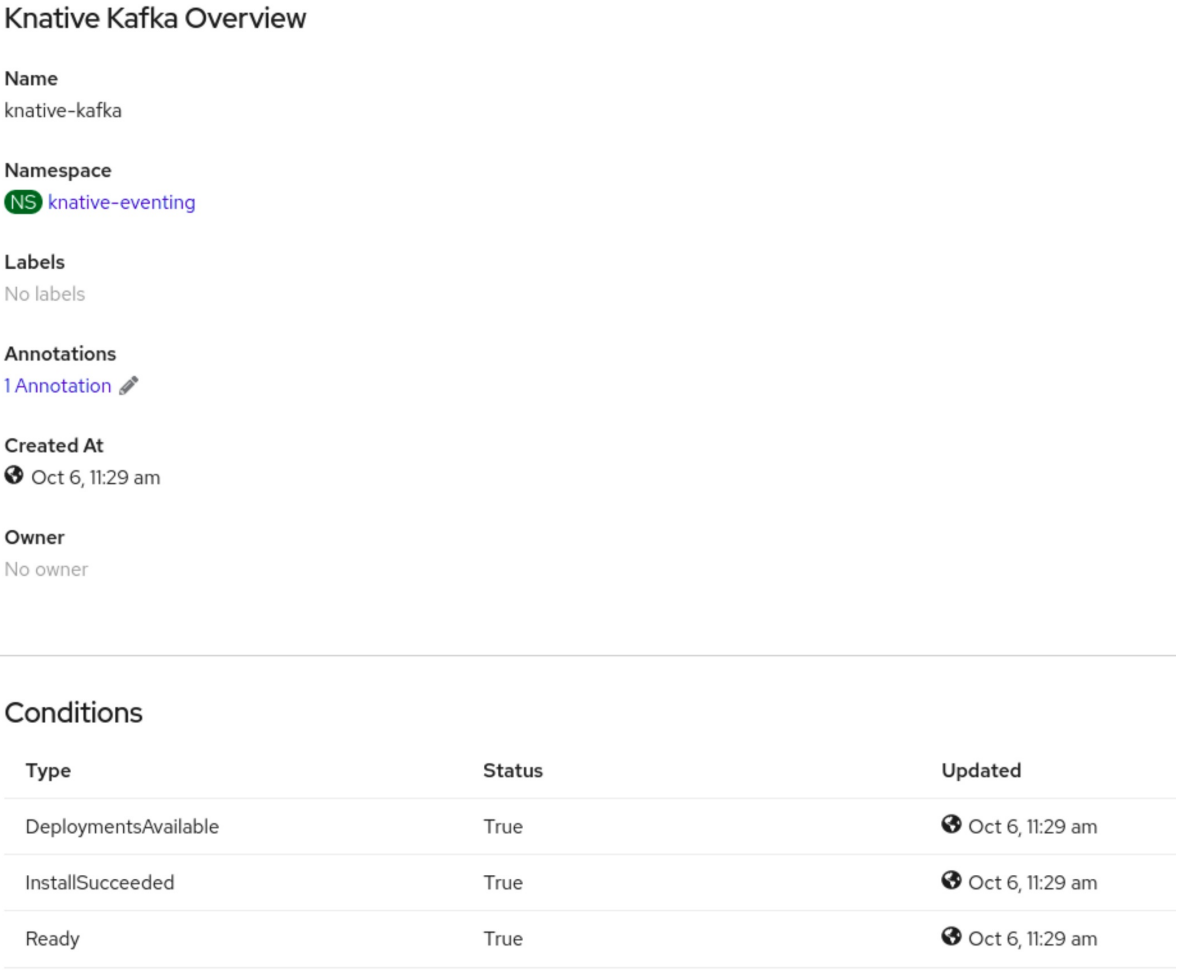
b. **KnativeKafka** 오브젝트 생성을 완전히 제어해야 하는 복잡한 구성의 경우 YAML을 편집하는 것이 좋습니다. **Knative Kafka 생성** 페이지의 오른쪽 상단에 있는 **YAML 편집** 링크를 클릭하여 YAML에 액세스할 수 있습니다.

5. Kafka에 대한 선택적 구성을 완료한 후 **생성**을 클릭합니다. 그러면 리소스 목록에 **knative-kafka**가 있는 **Knative Kafka** 탭으로 자동으로 이동합니다.

검증

1. **Knative Kafka** 탭에서 **knative-kafka** 리소스를 클릭합니다. 그러면 자동으로 **Knative Kafka 개요** 페이지로 이동합니다.

2. 리소스에 대한 **조건** 목록을 확인하고 상태가 **True**인지 확인합니다.



조건 상태가 **알 수 없음** 또는 **False**인 경우 몇 분 정도 기다린 후 페이지를 새로 고칩니다.

3. Knative Kafka 리소스가 생성되었는지 확인합니다.

```
$ oc get pods -n knative-eventing
```

출력 예

```
NAME                                   READY   STATUS    RESTARTS   AGE
kafka-broker-dispatcher-7769fbbcbb-xgffn    2/2     Running   0          44s
kafka-broker-receiver-5fb56f7656-fhq8d      2/2     Running   0          44s
kafka-channel-dispatcher-84fd6cb7f9-k2tjv   2/2     Running   0          44s
kafka-channel-receiver-9b7f795d5-c76xr      2/2     Running   0          44s
kafka-controller-6f95659bf6-trd6r           2/2     Running   0          44s
kafka-source-dispatcher-6bf98bdfff-8bcsn    2/2     Running   0          44s
kafka-webhook-eventing-68dc95d54b-825xs     2/2     Running   0          44s
```

3.5.4. 다음 단계

- Knative 서비스를 사용하려면 Knative Serving을 설치할 수 있습니다.

3.6. KNATIVE KAFKA 구성

Knative Kafka는 OpenShift Serverless에서 지원되는 Apache Kafka 메시지 스트리밍 플랫폼을 사용할 수 있는 통합 옵션을 제공합니다. Kafka는 이벤트 소스, 채널, 브로커 및 이벤트 싱크 기능에 대한 옵션을 제공합니다.

클러스터 관리자는 핵심 OpenShift Serverless 설치의 일부로 제공되는 Knative Eventing 구성 요소 외에도 **KnativeKafka** 사용자 정의 리소스 (CR)를 설치할 수 있습니다.

참고

Knative Kafka는 현재 IBM Z 및 IBM Power에서 지원되지 않습니다.

KnativeKafka CR은 다음과 같은 추가 옵션을 제공합니다.

- Kafka 소스
- Kafka 채널
- Kafka 브로커
- Kafka 싱크

3.7. OPENSHIFT SERVERLESS FUNCTIONS 구성

애플리케이션 코드 배포 프로세스를 개선하기 위해 OpenShift Serverless를 사용하여 이벤트 중심 기능을 OpenShift Container Platform에서 Knative 서비스로 배포할 수 있습니다. 함수를 개발하려면 설정 단계를 완료해야 합니다.

3.7.1. 사전 요구 사항

클러스터에서 OpenShift Serverless Functions을 사용하려면 다음 단계를 완료해야 합니다.

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.

참고

함수는 Knative 서비스로 배포됩니다. 함수에 이벤트 중심 아키텍처를 사용하려면 Knative Eventing도 설치해야 합니다.

- **oc** CLI 가 설치되어 있어야 합니다.
- Knative(**kn**) CLI 가 설치되어 있습니다. Knative CLI를 설치하면 함수를 생성하고 관리하는 데 사용할 수 있는 **kn func** 명령을 사용할 수 있습니다.
- Docker Container Engine 또는 Podman 버전 3.4.7 이상이 설치되어 있습니다.
- OpenShift Container Registry와 같이 사용 가능한 이미지 레지스트리에 액세스할 수 있습니다.
- Quay.io 를 이미지 레지스트리로 사용하는 경우 리포지토리가 비공개가 아닌지 확인하거나 pod가 다른 보안 레지스트리의 이미지를 참조하도록 허용하는 OpenShift Container Platform 설명서를 따라야 합니다.

- OpenShift Container Registry를 사용하는 경우 클러스터 관리자가 레지스트리를 공개해야 합니다.

3.7.2. Podman 설정

고급 컨테이너 관리 기능을 사용하려면 OpenShift Serverless Functions와 함께 Podman을 사용할 수 있습니다. 이를 위해 Podman 서비스를 시작하고 Knative(**kn**) CLI를 구성하여 연결합니다.

절차

1. **${XDG_RUNTIME_DIR}/podman/podman.sock**:의 UNIX 소켓에서 Docker API를 제공하는 Podman 서비스를 시작합니다.

```
$ systemctl start --user podman.socket
```

참고

대부분의 시스템에서 이 소켓은 **/run/user/$(id -u)/podman/podman.sock**에 있습니다.

2. 기능을 구축하는 데 사용되는 환경 변수를 설정합니다.

```
$ export DOCKER_HOST="unix://${XDG_RUNTIME_DIR}/podman/podman.sock"
```

3. 자세한 출력을 보려면 **-v** 플래그를 사용하여 함수 프로젝트 디렉터리 내에서 build 명령을 실행합니다. 로컬 UNIX 소켓에 대한 연결이 표시됩니다.

```
$ kn func build -v
```

3.7.3. macOS에서 Podman 설정

고급 컨테이너 관리 기능을 사용하려면 OpenShift Serverless Functions와 함께 Podman을 사용할 수 있습니다. macOS에서 이 작업을 수행하려면 Podman 머신을 시작하고 Knative(**kn**) CLI를 구성하여 연결합니다.

절차

1. Podman 시스템을 생성합니다.

```
$ podman machine init --memory=8192 --cpus=2 --disk-size=20
```

2. UNIX 소켓에서 Docker API를 제공하는 Podman 시스템을 시작합니다.

```
$ podman machine start
Starting machine "podman-machine-default"
Waiting for VM ...
Mounting volume... /Users/myuser:/Users/user

[...truncated output...]

You can still connect Docker API clients by setting DOCKER_HOST using the
following command in your terminal session:
```

```
 export
DOCKER_HOST='unix:///Users/myuser/.local/share/containers/podman/machine/podman-machine-default/podman.sock'

Machine "podman-machine-default" started successfully
```

참고

대부분의 macOS 시스템에서 이 소켓은 **/Users/myuser/.local/share/containers/podman/machine/podman-default/podman.sock** 에 있습니다.

3. 기능을 구축하는 데 사용되는 환경 변수를 설정합니다.

```
$ export
DOCKER_HOST='unix:///Users/myuser/.local/share/containers/podman/machine/podman-machine-default/podman.sock'
```

4. 자세한 출력을 보려면 **-v** 플래그를 사용하여 함수 프로젝트 디렉터리 내에서 build 명령을 실행합니다. 로컬 UNIX 소켓에 대한 연결이 표시됩니다.

```
$ kn func build -v
```

3.7.4. 다음 단계

- Docker Container Engine 또는 Podman에 대한 자세한 내용은 컨테이너 빌드 툴 옵션을 참조하십시오.
- 함수 시작하기를 참조하십시오.

4장. 서빙

4.1. KNATIVE SERVING 시작하기

4.1.1. 서버리스 애플리케이션

서버리스 애플리케이션은 경로와 구성으로 정의되고 YAML 파일에 포함된 Kubernetes 서비스로 생성 및 배포됩니다. OpenShift Serverless를 사용하여 서버리스 애플리케이션을 배포하려면 Knative **Service** 오브젝트를 생성해야 합니다.

Knative Service 오브젝트 YAML 파일의 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: hello 1
  namespace: default 2
spec:
  template:
    spec:
      containers:
        - image: docker.io/openshift/hello-openshift 3
          env:
            - name: RESPONSE 4
              value: "Hello Serverless!"
```

1. 애플리케이션 이름입니다.
2. 애플리케이션에서 사용하는 네임스페이스입니다.
3. 애플리케이션 이미지입니다.
4. 샘플 애플리케이션에서 출력한 환경 변수입니다.

다음 방법 중 하나를 사용하여 서버리스 애플리케이션을 생성합니다.

- OpenShift Container Platform 웹 콘솔에서 Knative 서비스를 생성합니다.
  자세한 내용은 개발자 화면을 사용하여 애플리케이션 생성을 참조하십시오.
- Knative(**kn**) CLI를 사용하여 Knative 서비스를 생성합니다.
- **oc** CLI를 사용하여 Knative **Service** 오브젝트를 YAML 파일로 생성하고 적용합니다.

4.1.1.1. Knative CLI를 사용하여 서버리스 애플리케이션 생성

Knative(**kn**) CLI를 사용하여 서버리스 애플리케이션을 생성하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다. **kn service create** 명령을 사용하여 기본 서버리스 애플리케이션을 생성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.

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

절차

- Knative 서비스를 생성합니다.

```
$ kn service create <service-name> --image <image> --tag <tag-value>
```

  다음과 같습니다.

  - **--image** 는 애플리케이션의 이미지 URI입니다.
  - **--tag** 는 서비스로 생성된 초기 버전에 태그를 추가하는 데 사용할 수 있는 선택적 플래그입니다.

  명령 예

```
$ kn service create event-display \
    --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

  출력 예

```
Creating service 'event-display' in namespace 'default':

  0.271s The Route is still working to reflect the latest desired specification.
  0.580s Configuration "event-display" is waiting for a Revision to become ready.
  3.857s ...
  3.861s Ingress has not yet been reconciled.
  4.270s Ready to serve.

Service 'event-display' created with latest revision 'event-display-bxshg-1' and URL:
http://event-display-default.apps-crc.testing
```

4.1.1.2. YAML을 사용하여 서버리스 애플리케이션 생성

YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 방식으로 애플리케이션을 재현할 수 있으며 재현 가능한 방식으로 애플리케이션을 설명할 수 있습니다. YAML을 사용하여 서버리스 애플리케이션을 생성하려면 Knative **Service** 오브젝트를 정의하는 YAML 파일을 생성한 다음 **oc apply** 를 사용하여 적용해야 합니다.

서비스가 생성되고 애플리케이션이 배포되면 Knative에서 이 버전의 애플리케이션에 대해 변경할 수 없는 버전을 생성합니다. Knative는 또한 네트워크 프로그래밍을 수행하여 애플리케이션의 경로, 수신, 서비스 및 로드 밸런서를 생성하고 트래픽에 따라 Pod를 자동으로 확장합니다.

사전 요구 사항

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

- OpenShift CLI(**oc**)를 설치합니다.

절차

1. 다음 샘플 코드를 포함하는 YAML 파일을 생성합니다.

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-delivery
  namespace: default
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
          env:
            - name: RESPONSE
              value: "Hello Serverless!"
```

2. YAML 파일이 포함된 디렉터리로 이동한 후 YAML 파일을 적용하여 애플리케이션을 배포합니다.

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

OpenShift Container Platform 웹 콘솔에서 **개발자** 화면으로 전환하거나 Knative(**kn**) CLI 또는 YAML 파일을 사용하려면 OpenShift Container Platform 웹 콘솔의 **Administator** 관점을 사용하여 Knative 구성 요소를 생성할 수 있습니다.

4.1.1.3. 관리자 화면을 사용하여 서버리스 애플리케이션 생성

서버리스 애플리케이션은 경로와 구성으로 정의되고 YAML 파일에 포함된 Kubernetes 서비스로 생성 및 배포됩니다. OpenShift Serverless를 사용하여 서버리스 애플리케이션을 배포하려면 Knative **Service** 오브젝트를 생성해야 합니다.

Knative Service 오브젝트 YAML 파일의 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: hello 1
  namespace: default 2
spec:
  template:
    spec:
      containers:
        - image: docker.io/openshift/hello-openshift 3
          env:
            - name: RESPONSE 4
              value: "Hello Serverless!"
```

1. 애플리케이션 이름입니다.
2. 애플리케이션에서 사용하는 네임스페이스입니다.

3 애플리케이션 이미지입니다.

4 샘플 애플리케이션에서 출력한 환경 변수입니다.

서비스가 생성되고 애플리케이션이 배포되면 Knative에서 이 버전의 애플리케이션에 대해 변경할 수 없는 버전을 생성합니다. Knative는 또한 네트워크 프로그래밍을 수행하여 애플리케이션의 경로, 수신, 서비스 및 로드 밸런서를 생성하고 트래픽에 따라 Pod를 자동으로 확장합니다.

사전 요구 사항

관리자 화면을 사용하여 서버리스 애플리케이션을 생성하려면 다음 단계를 완료해야 합니다.

- OpenShift Serverless Operator 및 Knative Serving이 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 **관리자** 화면에 있습니다.

절차

1. **Serverless → Serving** 페이지로 이동합니다.
2. **생성** 목록에서 **서비스**를 선택합니다.
3. YAML 또는 JSON 정의를 수동으로 입력하거나 파일을 편집기로 끌어서 놓습니다.
4. **생성**을 클릭합니다.

4.1.1.4. 오프라인 모드를 사용하여 서비스 생성

오프라인 모드에서 **kn service** 명령을 실행할 수 있으므로 클러스터에서 변경 사항이 발생하지 않고 로컬 시스템에서 서비스 설명자 파일이 생성됩니다. 설명자 파일이 생성되면 클러스터에 대한 변경 사항을 전파하기 전에 파일을 수정할 수 있습니다.

중요

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

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

사전 요구 사항

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

절차

1. 오프라인 모드에서 로컬 Knative 서비스 설명자 파일을 생성합니다.

```
$ kn service create event-display \
    --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest \
```

```
  --target ./ \
  --namespace test
```

출력 예

```
Service 'event-display' created in namespace 'test'.
```

- **--target ./** 플래그는 오프라인 모드를 활성화하고 새 디렉터리 트리를 저장하는 디렉토리로 **./**를 지정합니다.
  기존 디렉터리를 지정하지 않고 **--target my-service.yaml**과 같은 파일 이름을 사용하면 디렉터리 트리가 생성되지 않습니다. 대신 서비스 설명자 파일 **my-service.yaml**만 현재 디렉터리에 생성됩니다.

  파일 이름에는 **.yaml**, **.yml**, 또는 **.json** 확장을 사용할 수 있습니다. **.json**을 선택하면 JSON 형식으로 서비스 설명자 파일을 생성합니다.

- **--namespace test** 옵션은 새 서비스를 **test** 네임스페이스에 배치합니다.
  --namespace를 사용하지 않고 OpenShift Container Platform 클러스터에 로그인한 경우 현재 네임스페이스에 설명자 파일이 생성됩니다. 그렇지 않으면 설명자 파일이 **default** 네임스페이스에 생성됩니다.

2. 생성된 디렉터리 구조를 확인합니다.

```
$ tree ./
```

출력 예

```
./
└── test
    └── ksvc
        └── event-display.yaml

2 directories, 1 file
```

- **--target**에서 지정된 현재 **./** 디렉터리에는 지정된 네임스페이스를 바탕으로 이름이 지정된 **test/** 디렉터리가 포함되어 있습니다.

- **test/** 디렉터리에는 리소스 유형의 이름에 따라 이름이 지정된 **ksvc** 디렉터리가 포함되어 있습니다.

- **ksvc** 디렉터리에는 지정된 서비스 이름에 따라 이름이 지정된 기술자 파일 **event-display.yaml**이 포함되어 있습니다.

3. 생성된 서비스 기술자 파일을 확인합니다.

```
$ cat test/ksvc/event-display.yaml
```

출력 예

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

```
  namespace: test
spec:
  template:
    metadata:
      annotations:
        client.knative.dev/user-image: quay.io/openshift-knative/knative-eventing-sources-event-
display:latest
      creationTimestamp: null
    spec:
      containers:
      - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
        name: ""
        resources: {}
status: {}
```

4. 새 서비스에 대한 정보를 나열합니다.

```
$ kn service describe event-display --target ./ --namespace test
```

출력 예

```
Name:       event-display
Namespace:  test
Age:
URL:

Revisions:

Conditions:
  OK TYPE    AGE REASON
```

- **target ./** 옵션은 네임스페이스 하위 디렉터리를 포함하는 디렉터리 구조의 루트 디렉터리를 지정합니다.
  또는 **--target** 옵션을 사용하여 YAML 또는 JSON 파일 이름을 직접 지정할 수 있습니다. 허용된 파일 확장자는 **.yaml**, **.yml**, **.json**입니다.

- **--namespace** 옵션은 필요한 서비스 기술자 파일을 포함하는 하위 디렉터리를 **kn**와 통신하는 네임스페이스를 지정합니다.
  --namespace를 사용하지 않고 OpenShift Container Platform 클러스터에 로그인한 경우 **kn**은 현재 네임스페이스를 바탕으로 이름이 지정된 하위 디렉터리에서 서비스를 검색합니다. 그렇지 않으면 **kn**은 **default/** 하위 디렉터리에서 검색합니다.

5. 서비스 설명자 파일을 사용하여 클러스터에 서비스를 생성합니다.

```
$ kn service create -f test/ksvc/event-display.yaml
```

출력 예

```
Creating service 'event-display' in namespace 'test':

  0.058s The Route is still working to reflect the latest desired specification.
  0.098s ...
  0.168s Configuration "event-display" is waiting for a Revision to become ready.
 23.377s ...
```

```
23.419s Ingress has not yet been reconciled.
23.534s Waiting for load balancer to be ready
23.723s Ready to serve.

Service 'event-display' created to latest revision 'event-display-00001' is available at URL:
http://event-display-test.apps.example.com
```

4.1.1.5. 추가 리소스

- Knative Serving CLI 명령
- Knative 서비스의 JSON Web Token 인증 설정

4.1.2. 서버리스 애플리케이션 배포 확인

서버리스 애플리케이션이 성공적으로 배포되었는지 확인하려면 Knative에서 생성한 애플리케이션 URL을 가져와서 해당 URL로 요청을 보낸 후 출력을 관찰해야 합니다. OpenShift Serverless에서는 HTTP 및 HTTPS URL을 모두 사용할 수 있지만 **oc get ksvc** 출력에서는 항상 **http://** 형식을 사용하여 URL을 출력합니다.

4.1.2.1. 서버리스 애플리케이션 배포 확인

서버리스 애플리케이션이 성공적으로 배포되었는지 확인하려면 Knative에서 생성한 애플리케이션 URL을 가져와서 해당 URL로 요청을 보낸 후 출력을 관찰해야 합니다. OpenShift Serverless에서는 HTTP 및 HTTPS URL을 모두 사용할 수 있지만 **oc get ksvc** 출력에서는 항상 **http://** 형식을 사용하여 URL을 출력합니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- **oc** CLI를 설치했습니다.
- Knative 서비스를 생성했습니다.

사전 요구 사항

- OpenShift CLI(**oc**)를 설치합니다.

절차

1. 애플리케이션 URL을 찾습니다.

```
$ oc get ksvc <service_name>
```

출력 예

```
NAME            URL                                        LATESTCREATED         LATESTREADY           READY   REASON
event-delivery  http://event-delivery-default.example.com  event-delivery-4wsd2  event-delivery-4wsd2  True
```

2. 클러스터에 요청한 후 출력을 확인합니다.

HTTP 요청의 예

```
$ curl http://event-delivery-default.example.com
```

HTTPS 요청의 예

```
$ curl https://event-delivery-default.example.com
```

출력 예

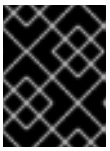
```
Hello Serverless!
```

3. 선택 사항입니다. 인증서 체인에서 자체 서명된 인증서와 관련된 오류가 발생하면 **--insecure** 플래그를 curl 명령에 추가하여 오류를 무시할 수 있습니다.

```
$ curl https://event-delivery-default.example.com --insecure
```

출력 예

```
Hello Serverless!
```



중요

프로덕션 배포에는 자체 서명된 인증서를 사용해서는 안 됩니다. 이 방법은 테스트 목적으로만 사용됩니다.

4. 선택 사항입니다. OpenShift Container Platform 클러스터가 CA(인증 기관)에서 서명한 인증서로 구성되어 있지만 아직 시스템에 전역적으로 구성되지 않은 경우 **curl** 명령으로 이 인증서를 지정할 수 있습니다. 인증서 경로는 **--cacert** 플래그를 사용하여 curl 명령에 전달할 수 있습니다.

```
$ curl https://event-delivery-default.example.com --cacert <file>
```

출력 예

```
Hello Serverless!
```

4.2. 자동 확장

4.2.1. 자동 확장

Knative Serving은 들어오는 수요와 일치하도록 애플리케이션에서 *자동 스케일링 또는 자동 스케일링* 을 제공합니다. 예를 들어 애플리케이션에서 트래픽을 수신하고 0으로 스케일링하는 경우 Knative Serving이 애플리케이션을 복제본 0으로 축소합니다. scale-to-zero이 비활성화된 경우 애플리케이션은 클러스터의 애플리케이션에 대해 구성된 최소 복제본 수로 축소됩니다. 애플리케이션으로의 트래픽이 증가하면 요구에 맞게 복제본을 확장할 수도 있습니다.

Knative 서비스의 자동 스케일링 설정은 클러스터 관리자가 구성하는 전역 설정 또는 개별 서비스에 대해 구성된 개별 설정일 수 있습니다.

서비스의 YAML 파일을 수정하거나 Knative(**kn**) CLI를 사용하여 OpenShift Container Platform 웹 콘솔을 사용하여 서비스 단위 설정을 수정할 수 있습니다.

참고

서비스에 대해 설정한 모든 제한 또는 대상은 애플리케이션의 단일 인스턴스에 대해 측정됩니다. 예를 들어 **target** 주석을 **50** 으로 설정하면 자동 스케일러가 애플리케이션을 스케일링하여 각 버전이 한 번에 50개의 요청을 처리하도록 구성됩니다.

4.2.2. scale bounds

scale bounds는 언제든지 애플리케이션을 제공할 수 있는 최소 및 최대 복제본 수를 결정합니다. 애플리케이션의 스케일 경계를 설정하여 콜드 시작을 방지하거나 컴퓨팅 비용을 제어할 수 있습니다.

4.2.2.1. 최소 스케일 경계

애플리케이션을 제공할 수 있는 최소 복제본 수는 **min-scale** 주석에 따라 결정됩니다. 스케일을 0으로 설정하지 않으면 **min-scale** 값이 기본값으로 **1** 로 설정됩니다.

다음 조건이 충족되는 경우 **min-scale** 값은 **0** 개 복제본으로 설정됩니다.

- **min-scale** 주석이 설정되지 않음
- 0으로 스케일링이 활성화됨
- 클래스 **KPA** 사용

min-scale 주석이 있는 서비스 사양의 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "0"
...
```

4.2.2.1.1. Knative CLI를 사용하여 min-scale 주석 설정

Knative(**kn**) CLI를 사용하여 **min-scale** 주석을 설정하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스가 제공됩니다. **kn service** 명령을 **--scale-min** 플래그와 함께 사용하여 서비스의 **min-scale** 값을 생성하거나 수정할 수 있습니다.

사전 요구 사항

- Knative Serving이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

절차

- **--scale-min** 플래그를 사용하여 서비스의 최소 복제본 수를 설정합니다.

```
$ kn service create <service_name> --image <image_uri> --scale-min <integer>
```

명령 예

```
$ kn service create example-service --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest --scale-min 2
```

4.2.2.2. 최대 스케일링 바인딩

애플리케이션을 제공할 수 있는 최대 복제본 수는 **max-scale** 주석에 따라 결정됩니다. **max-scale** 주석을 설정하지 않으면 생성된 복제본 수에 대한 상한이 없습니다.

max-scale 주석이 있는 서비스 사양의 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/max-scale: "10"
...
```

4.2.2.2.1. Knative CLI를 사용하여 max-scale 주석 설정

Knative(**kn**) CLI를 사용하여 **max-scale** 주석을 설정하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스가 제공됩니다. **kn service** 명령을 **--scale-max** 플래그와 함께 사용하여 서비스의 **max-scale** 값을 생성하거나 수정할 수 있습니다.

사전 요구 사항

- Knative Serving이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

절차

- **--scale-max** 플래그를 사용하여 서비스의 최대 복제본 수를 설정합니다.

```
$ kn service create <service_name> --image <image_uri> --scale-max <integer>
```

명령 예

```
$ kn service create example-service --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest --scale-max 10
```

4.2.3. 동시성

동시성은 지정된 시간에 애플리케이션의 각 복제본에서 처리할 수 있는 동시 요청 수를 결정합니다. 동시성을 *소프트 제한* 또는 *하드 제한* 으로 구성할 수 있습니다.

- 소프트 제한은 엄격하게 적용된 바인딩이 아닌 대상 요청 제한입니다. 예를 들어 트래픽이 갑작스러운 버스트가 있는 경우 소프트 제한 대상을 초과할 수 있습니다.
- 하드 제한은 엄격하게 적용된 상위 바인딩된 요청 제한입니다. 동시성이 하드 제한에 도달하면 초과 요청이 버퍼링되고 요청을 실행하기에 충분한 여유 용량이 있을 때까지 기다려야 합니다.

중요

하드 제한 구성을 사용하는 것은 애플리케이션과 함께 명확한 사용 사례가 있는 경우에만 권장됩니다. 낮은 하드 제한을 지정하면 애플리케이션의 처리량 및 대기 시간에 부정적인 영향을 줄 수 있으며 콜드 시작이 발생할 수 있습니다.

소프트 대상과 하드 제한을 추가하면 자동 스케일러가 소프트 대상 동시 요청 수를 대상으로 하지만 최대 요청 수에 하드 제한 값의 하드 제한을 적용합니다.

하드 제한 값이 소프트 제한 값보다 작으면 실제로 처리할 수 있는 수보다 더 많은 요청을 대상으로 할 필요가 없기 때문에 소프트 제한 값이 조정됩니다.

4.2.3.1. 소프트 동시성 대상 구성

소프트 제한은 엄격하게 적용된 바인딩이 아닌 대상 요청 제한입니다. 예를 들어 트래픽이 갑작스러운 버스트가 있는 경우 소프트 제한 대상을 초과할 수 있습니다. 사양에 **autoscaling.knative.dev/target** 주석을 설정하거나 **kn service** 명령을 올바른 플래그와 함께 사용하여 Knative 서비스의 소프트 동시성 대상을 지정할 수 있습니다.

절차

- 선택 사항: **Service** 사용자 정의 리소스 사양에서 Knative 서비스의 **autoscaling.knative.dev/target** 주석을 설정합니다.

  서비스 사양의 예

  ```
  apiVersion: serving.knative.dev/v1
  kind: Service
  metadata:
    name: example-service
    namespace: default
  spec:
    template:
      metadata:
        annotations:
          autoscaling.knative.dev/target: "200"
  ```

- 선택 사항: **kn service** 명령을 사용하여 **--concurrency-target** 플래그를 지정합니다.

  ```
  $ kn service create <service_name> --image <image_uri> --concurrency-target <integer>
  ```

  동시성 대상이 50개 요청인 서비스를 생성하는 명령의 예

  ```
  $ kn service create example-service --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest --concurrency-target 50
  ```

-

4.2.3.2. 하드 동시성 제한 구성

하드 동시성 제한은 엄격하게 적용된 상위 바인딩된 요청 제한입니다. 동시성이 하드 제한에 도달하면 초과 요청이 버퍼링되고 요청을 실행하기에 충분한 여유 용량이 있을 때까지 기다려야 합니다.
containerConcurrency 사양을 수정하거나 **kn service** 명령을 올바른 플래그와 함께 사용하여 Knative 서비스의 하드 동시성 제한을 지정할 수 있습니다.

절차

- 선택사항: **Service** 사용자 정의 리소스의 사양에 Knative 서비스의 **containerConcurrency** 사양을 설정합니다.

  서비스 사양의 예

  ```
  apiVersion: serving.knative.dev/v1
  kind: Service
  metadata:
    name: example-service
    namespace: default
  spec:
    template:
      spec:
        containerConcurrency: 50
  ```

  기본값은 **0** 이며, 이는 한 번에 하나의 서비스 복제본으로 이동할 수 있는 동시 요청 수에 제한이 없음을 의미합니다.

  값이 **0** 보다 크면 한 번에 하나의 서비스 복제본으로 이동할 수 있는 정확한 요청 수를 지정합니다. 이 예제에서는 50개 요청의 하드 동시성 제한을 활성화합니다.

- 선택 사항: **kn service** 명령을 사용하여 **--concurrency-limit** 플래그를 지정합니다.

  ```
  $ kn service create <service_name> --image <image_uri> --concurrency-limit <integer>
  ```

  동시성 제한이 50개 요청인 서비스를 생성하는 명령의 예

  ```
  $ kn service create example-service --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest --concurrency-limit 50
  ```

4.2.3.3. 동시성 대상 사용률

이 값은 자동 스케일러가 실제로 대상으로 하는 동시성 제한의 백분율을 지정합니다. 이를 통해 복제본이 실행되는 *hotness* 를 지정하여 정의된 하드 제한에 도달하기 전에 자동 스케일러를 확장할 수 있습니다.

예를 들어 **containerConcurrency** 값이 10으로 설정되고 **target-utilization-percentage** 값이 70%로 설정되면 기존 복제본의 평균 동시 요청 수가 7에 도달하면 자동 스케일러는 새 복제본을 생성합니다. 7에서 10까지의 요청은 여전히 기존 복제본으로 전송되지만 **containerConcurrency** 값에 도달한 후 추가 복제본은 필요한 예상에서 시작됩니다.

target-utilization-percentage 주석을 사용하여 구성된 서비스의 예

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

```
metadata:
  name: example-service
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/target-utilization-percentage: "70"
...
```

4.2.4. Scale-to-zero

Knative Serving은 들어오는 수요와 일치하도록 애플리케이션에서 *자동 스케일링 또는 자동 스케일링* 을 제공합니다.

4.2.4.1. scale-to-zero 활성화

enable-scale-to-zero 사양을 사용하여 클러스터의 애플리케이션에 대해 전역적으로 스케일링을 활성화하거나 비활성화할 수 있습니다.

사전 요구 사항

- 클러스터에 OpenShift Serverless Operator 및 Knative Serving이 설치되어 있습니다.
- 클러스터 관리자 권한이 있어야 합니다.
- 기본 Knative Pod Autoscaler를 사용하고 있습니다. Kubernetes Horizontal Pod Autoscaler를 사용하는 경우 제로로 확장 기능을 사용할 수 없습니다.

절차

- **KnativeServing** 사용자 정의 리소스(CR)에서 **enable-scale-to-zero** 사양을 수정합니다.

  KnativeServing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    autoscaler:
      enable-scale-to-zero: "false" 1
```

  1. **enable-scale-to-zero** 사양은 **"true"** 또는 **"false"** 일 수 있습니다. true로 설정하면 scale-to-zero이 활성화됩니다. false로 설정하면 애플리케이션이 구성된 *최소 스케일 바인딩* 으로 축소됩니다. 기본값은 **"true"** 입니다.

4.2.4.2. scale-to-zero 유예 기간 구성

Knative Serving은 애플리케이션에 대해 Pod 0개로 자동 스케일링을 제공합니다. **scale-to-zero-grace-period** 사양을 사용하여 Knative에서 scale-to-zero machinery가 애플리케이션 마지막 복제본을 제거하기 전에 대기하는 상한 시간 제한을 정의할 수 있습니다.

사전 요구 사항

- 클러스터에 OpenShift Serverless Operator 및 Knative Serving이 설치되어 있습니다.
- 클러스터 관리자 권한이 있어야 합니다.
- 기본 Knative Pod Autoscaler를 사용하고 있습니다. Kubernetes Horizontal Pod Autoscaler를 사용하는 경우 scale-to-zero 기능을 사용할 수 없습니다.

절차

- **KnativeServing** CR(사용자 정의 리소스)에서 **scale-to-grace-period** 사양을 수정합니다.

  KnativeServing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    autoscaler:
      scale-to-zero-grace-period: "30s" 1
```

  1 유예 기간(초)입니다. 기본값은 30초입니다.

4.3. 서버리스 애플리케이션 구성

4.3.1. Knative Serving 시스템 배포 구성 덮어쓰기

KnativeServing CR(사용자 정의 리소스)에서 deployments 사양을 수정하여 일부 특정 **배포** 의 기본 구성을 덮어쓸 수 있습니다.

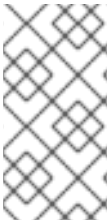
4.3.1.1. 시스템 배포 구성 덮어쓰기

현재는 **리소스,복제본,레이블,주석**, **nodeSelector** 필드에는 기본 구성 설정을 재정의하고 **프로브** 의 준비 및 활성 필드에도 지원됩니다.

다음 예에서 KnativeServing CR은 다음과 같이 Webhook 배포를 덮어씁니다.

- **net-kourier-controller 의 준비 상태 프로브 타임아웃이 10초로 설정됩니다.**
- **배포에 CPU 및 메모리 리소스 제한이 지정되었습니다.**

- 배포에는 3개의 복제본이 있습니다.
- example-label: 레이블 레이블이 추가되었습니다.
- example-annotation: 주석 주석이 추가되었습니다.
- nodeSelector 필드는 disktype: hdd 라벨이 있는 노드를 선택하도록 설정됩니다.



참고

KnativeServing CR 라벨 및 주석 설정은 배포 자체와 결과 Pod에 대한 배포 레이블 및 주석을 재정의합니다.

KnativeServing CR 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: ks
  namespace: knative-serving
spec:
  high-availability:
    replicas: 2
  deployments:
  - name: net-kourier-controller
    readinessProbes: 1
      - container: controller
        timeoutSeconds: 10
  - name: webhook
    resources:
    - container: webhook
      requests:
        cpu: 300m
        memory: 60Mi
      limits:
        cpu: 1000m
        memory: 1000Mi
    replicas: 3
    labels:
      example-label: label
    annotations:
```

```
    example-annotation: annotation
  nodeSelector:
    disktype: hdd
```

1

준비 상태 및 활성 상태 프로브 덮어쓰기를 사용하여 프로브에 지정된 대로 Kubernetes API에 지정된 대로 배포 컨테이너에 있는 프로브의 모든 필드를 재정의할 수 있습니다. 그러나 프로브 처리기와 관련된 필드인 exec,grpc,httpGet, tcpSocket.

추가 리소스

- Kubernetes API 문서의 프로브 구성 섹션

4.3.2. emptyDir 볼륨

emptyDir 볼륨은 Pod를 생성할 때 생성되는 빈 볼륨이며 임시 디스크 공간을 제공하는 데 사용됩니다. emptyDir 볼륨은 포드가 생성될 때 삭제됩니다.

4.3.2.1. EmptyDir 확장 구성

kubernetes.podspec-volumes-emptydir 확장 기능은 emptyDir 볼륨을 Knative Serving과 함께 사용할 수 있는지 여부를 제어합니다. emptyDir 볼륨을 사용하여 활성화하려면 다음 YAML을 포함하도록 KnativeServing CR(사용자 정의 리소스)을 수정해야 합니다.

KnativeServing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    features:
      kubernetes.podspec-volumes-emptydir: enabled
...
```

4.3.3. Serving용 영구 볼륨 클레임

일부 서버리스 애플리케이션에는 영구 데이터 스토리지가 필요합니다. 이를 위해 Knative 서비스에 대한 PVC(영구 볼륨 클레임)를 구성할 수 있습니다.

4.3.3.1. PVC 지원 활성화

절차

1. Knative Serving이 PVC를 사용하고 작성할 수 있도록 하려면 다음 YAML을 포함하도록 KnativeServing CR(사용자 정의 리소스)을 수정합니다.

   쓰기 액세스로 PVC 활성화

```
...
spec:
  config:
    features:
      "kubernetes.podspec-persistent-volume-claim": enabled
      "kubernetes.podspec-persistent-volume-write": enabled
...
```

   - kubernetes.podspec-persistent-volume-claim 확장 기능은 Knative Serving에서 PV(영구 볼륨)를 사용할 수 있는지 여부를 제어합니다.
   - kubernetes.podspec-persistent-volume-write 확장 기능은 쓰기 액세스 권한이 있는 Knative Serving에서 PV를 사용할 수 있는지 여부를 제어합니다.

2. PV를 클레임하려면 PV 구성을 포함하도록 서비스를 수정합니다. 예를 들어 다음 구성을 사용하는 영구 볼륨 클레임이 있을 수 있습니다.

   참고

   요청 중인 액세스 모드를 지원하는 스토리지 클래스를 사용합니다. 예를 들어 ReadWriteMany 액세스 모드에 ocs-storagecluster-cephfs 클래스를 사용할 수 있습니다.

PersistentVolumeClaim 구성

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pv-claim
  namespace: my-ns
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ocs-storagecluster-cephfs
  resources:
    requests:
      storage: 1Gi
```

이 경우 쓰기 액세스 권한이 있는 PV를 요청하려면 서비스를 다음과 같이 수정합니다.

Knative 서비스 PVC 구성

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  namespace: my-ns
...
spec:
 template:
   spec:
     containers:
         ...
         volumeMounts: 1
           - mountPath: /data
             name: mydata
             readOnly: false
     volumes:
       - name: mydata
         persistentVolumeClaim: 2
           claimName: example-pv-claim
           readOnly: false 3
```

1

볼륨 마운트 사양.

2

영구 볼륨 클레임 사양.

3

읽기 전용 액세스를 활성화하는 플래그입니다.

참고

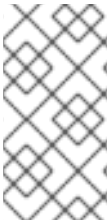
Knative 서비스에서 영구 스토리지를 사용하려면 Knative 컨테이너 사용자에 대한 사용자 권한과 같은 추가 구성이 필요합니다.

4.3.3.2. 추가 리소스

- 영구저장장치 이해

4.3.4. Init 컨테이너

Init 컨테이너 는 Pod의 애플리케이션 컨테이너보다 먼저 실행되는 특수 컨테이너입니다. 일반적으로 애플리케이션에 대한 초기화 논리를 구현하는 데 사용됩니다. 이 논리에는 설정 스크립트를 실행하거나 필요한 구성을 다운로드하는 작업이 포함될 수 있습니다. KnativeServing CR(사용자 정의 리소스)을 수정하여 Knative 서비스의 init 컨테이너 사용을 활성화할 수 있습니다.



참고

Init 컨테이너는 애플리케이션 시작 시간이 길어질 수 있으며 서버리스 애플리케이션에 주의해서 사용해야 합니다. 이 경우 자주 확장 및 축소할 것으로 예상됩니다.

4.3.4.1. init 컨테이너 활성화

사전 요구 사항

- 클러스터에 OpenShift Serverless Operator 및 Knative Serving이 설치되어 있습니다.

- 클러스터 관리자 권한이 있어야 합니다.

절차

- **kubernetes.podspec-init-containers** 플래그를 **KnativeServing CR**에 추가하여 **init** 컨테이너 사용을 활성화합니다.

KnativeServing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    features:
      kubernetes.podspec-init-containers: enabled
...
```

4.3.5. 다이제스트로 이미지 태그 확인

Knative Serving 컨트롤러가 컨테이너 레지스트리에 액세스할 수 있는 경우 **Knative Serving**은 서비스 버전을 생성할 때 이미지 태그를 다이제스트로 확인합니다. 이 *문제를 태그-다이제스트 해결* 이라고 하며 배포에 일관성을 제공하는 데 도움이 됩니다.

4.3.5.1. Tag-to-digest resolution

OpenShift Container Platform의 컨테이너 레지스트리에 대한 컨트롤러 액세스 권한을 부여하려면 보안을 생성한 다음 컨트롤러 사용자 정의 인증서를 구성해야 합니다. **KnativeServing CR**(사용자 정의 리소스)에서 **controller-custom-certs** 사양을 수정하여 컨트롤러 사용자 정의 인증서를 구성할 수 있습니다. 보안은 **KnativeServing CR**과 동일한 네임스페이스에 있어야 합니다.

secret이 **KnativeServing CR**에 포함되어 있지 않은 경우 이 설정은 기본적으로 공개 키 인프라(**PKI**)를 사용합니다. **PKI**를 사용하는 경우 **config-service-sa** 구성 맵을 사용하여 클러스터 전체 인증서가 **Knative Serving** 컨트롤러에 자동으로 삽입됩니다. **OpenShift Serverless Operator**는 **config-service-sa** 구성 맵을 클러스터 전체 인증서로 채우고 구성 맵을 컨트롤러에 볼륨으로 마운트합니다.

4.3.5.1.1. 보안을 사용하여 태그-다이제스트 확인 구성

controller-custom-certs 사양에서 Secret 유형을 사용하는 경우 시크릿이 시크릿 볼륨으로 마운트됩니다. Knative 구성 요소는 보안에 필요한 인증서가 있다고 가정하여 직접 보안을 사용합니다.

사전 요구 사항

- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있습니다.
- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.

절차

1. 보안을 생성합니다.

명령 예

```
$ oc -n knative-serving create secret generic custom-secret --from-file=
<secret_name>.crt=<path_to_certificate>
```

2. Secret 유형을 사용하도록 KnativeServing CR(사용자 정의 리소스)에서 controller-custom-certs 사양을 구성합니다.

KnativeServing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  controller-custom-certs:
    name: custom-secret
    type: Secret
```

4.3.6. TLS 인증 구성

TLS(*Transport Layer Security*)를 사용하여 Knative 트래픽 및 인증을 암호화할 수 있습니다.

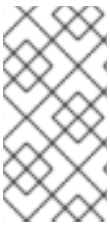
TLS는 Knative Kafka에서 지원되는 유일한 트래픽 암호화 방법입니다. Red Hat은 Knative Kafka 리소스에 SASL 및 TLS를 함께 사용하는 것이 좋습니다.

참고

Red Hat OpenShift Service Mesh 통합을 사용하여 내부 TLS를 활성화하려면 다음 절차에 설명된 내부 암호화 대신 mTLS를 사용하여 서비스 메시를 활성화해야 합니다. mTLS로 서비스 메시를 사용할 때 Knative Serving 메트릭 활성화에 대한 설명서를 참조하십시오.

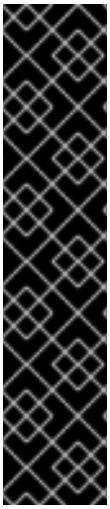
4.3.6.1. 내부 트래픽에 대한 TLS 인증 활성화

OpenShift Serverless는 기본적으로 TLS 엣지 종료를 지원하므로 최종 사용자의 HTTPS 트래픽이 암호화됩니다. 그러나 OpenShift 경로 뒤의 내부 트래픽은 일반 데이터를 사용하여 애플리케이션으로 전달됩니다. 내부 트래픽에 TLS를 활성화하면 구성 요소 간에 전송된 트래픽이 암호화되므로 이러한 트래픽의 보안이 향상됩니다.



참고

Red Hat OpenShift Service Mesh 통합을 사용하여 내부 TLS를 활성화하려면 다음 절차에 설명된 내부 암호화 대신 mTLS를 사용하여 서비스 메시를 활성화해야 합니다.



중요

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

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

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 설치되어 있습니다.
- OpenShift(oc) CLI를 설치했습니다.

절차

1. 사양에 internal-encryption: "true" 필드를 포함하는 Knative 서비스를 생성합니다.

```
...
spec:
  config:
    network:
      internal-encryption: "true"
...
```

2. knative-serving 네임스페이스에서 활성화기 Pod를 다시 시작하여 인증서를 로드합니다.

```
$ oc delete pod -n knative-serving --selector app=activator
```

추가 리소스

- Kafka 브로커의 TLS 인증 구성
- Kafka 채널에 대한 TLS 인증 구성
- mTLS로 서비스 메시를 사용할 때 Knative Serving 메트릭 활성화

4.3.7. 제한적인 네트워크 정책

4.3.7.1. 제한적인 네트워크 정책이 있는 클러스터

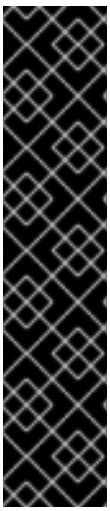
여러 사용자가 액세스할 수 있는 클러스터를 사용하는 경우 클러스터는 네트워크 정책을 사용하여 네트워크를 통해 서로 통신할 수 있는 pod, 서비스 및 네임스페이스를 제어할 수 있습니다. 클러스터에서 제한적인 네트워크 정책을 사용하는 경우 Knative 시스템 Pod가 Knative 애플리케이션에 액세스할 수 없습니다. 예를 들어 네임스페이스에 모든 요청을 거부하는 다음 네트워크 정책이 있는 경우 Knative 시스템 Pod가 Knative 애플리케이션에 액세스할 수 없습니다.

네임스페이스에 대한 모든 요청을 거부하는 NetworkPolicy 오브젝트의 예

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
  namespace: example-namespace
spec:
  podSelector:
  ingress: []
```

4.3.7.2. 제한적인 네트워크 정책을 사용하여 클러스터에서 Knative 애플리케이션과의 통신 활성화

Knative 시스템 **Pod**에서 애플리케이션에 액세스할 수 있도록 하려면 각 **Knative** 시스템 네임스페이스에 레이블을 추가한 다음 이 레이블이 있는 다른 네임스페이스의 네임스페이스에 액세스할 수 있는 애플리케이션 네임스페이스에 **NetworkPolicy** 오브젝트를 생성해야 합니다.



중요

클러스터의 비**Knative** 서비스에 대한 요청을 거부하는 네트워크 정책은 이러한 서비스에 대한 액세스를 계속 차단합니다. 그러나 **Knative** 시스템 네임스페이스에서 **Knative** 애플리케이션으로의 액세스를 허용하면 클러스터의 모든 네임스페이스에서 **Knative** 애플리케이션에 액세스할 수 있습니다.

클러스터의 모든 네임스페이스에서 **Knative** 애플리케이션에 대한 액세스를 허용하지 않으려면 ***Knative** 서비스에 **JSON** 웹 토큰 인증을* 사용할 수 있습니다. **Knative** 서비스에 대한 **JSON** 웹 토큰 인증에는 **Service Mesh**가 필요합니다.

사전 요구 사항

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

절차

1. 애플리케이션에 액세스해야 하는 각 **Knative** 시스템 네임스페이스에 **knative.openshift.io/system-namespace=true** 레이블을 추가합니다.

a. **knative-serving** 네임스페이스에 레이블을 지정합니다.

```
$ oc label namespace knative-serving knative.openshift.io/system-namespace=true
```

b. **knative-serving-ingress** 네임스페이스에 레이블을 지정합니다.

```
$ oc label namespace knative-serving-ingress knative.openshift.io/system-namespace=true
```

c. **knative-eventing** 네임스페이스에 레이블을 지정합니다.

```
$ oc label namespace knative-eventing knative.openshift.io/system-namespace=true
```

d. **knative-kafka** 네임스페이스에 레이블을 지정합니다.

```
$ oc label namespace knative-kafka knative.openshift.io/system-namespace=true
```

2. 애플리케이션 네임스페이스에 **NetworkPolicy** 오브젝트를 생성하여 **knative.openshift.io/system-namespace** 레이블이 있는 네임스페이스에서 액세스를 허용합니다.

NetworkPolicy 오브젝트 예

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: <network_policy_name> 1
  namespace: <namespace> 2
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          knative.openshift.io/system-namespace: "true"
  podSelector: {}
  policyTypes:
  - Ingress
```

1

네트워크 정책의 이름을 제공합니다.

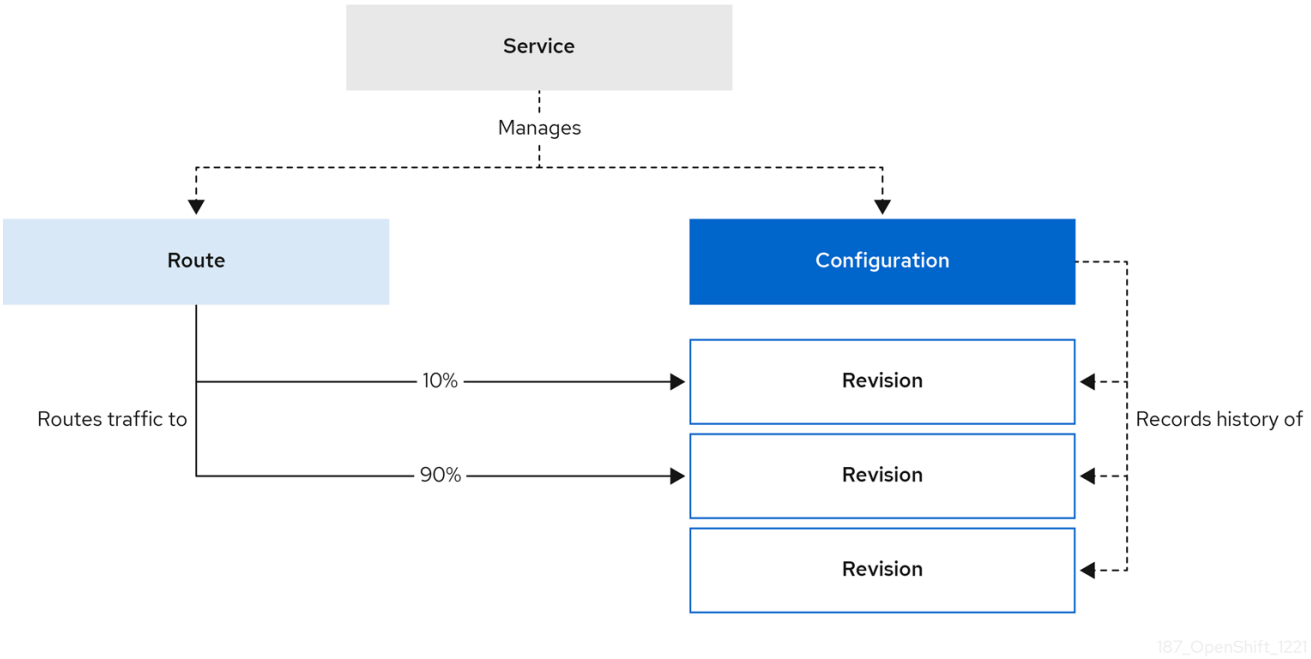
2

애플리케이션이 있는 네임스페이스입니다.

4.4. 트래픽 분할

4.4.1. 트래픽 분할 개요

Knative 애플리케이션에서 트래픽 분할을 생성하여 트래픽을 관리할 수 있습니다. 트래픽 분할은 **Knative** 서비스에서 관리하는 경로의 일부로 구성됩니다.



경로를 구성하면 요청을 다양한 버전의 서비스로 전송할 수 있습니다. 이 라우팅은 **Service** 오브젝트의 트래픽 사양에 따라 결정됩니다.

트래픽 사양 선언은 하나 이상의 버전으로 구성되며, 각각 전체 트래픽의 일부를 처리합니다. 각 버전으로 라우팅되는 트래픽의 백분율은 최대 **100%**를 추가해야 합니다. 이는 **Knative** 검증에 의해 확인됩니다.

트래픽 사양에 지정된 버전은 수정되거나 이름이 지정된 버전이거나 "latest" 버전을 가리킬 수 있으며, 이는 서비스의 모든 버전 목록의 헤드를 추적할 수 있습니다. 새 버전이 생성되는 경우 "latest" 버전은 업데이트하는 부동 참조 유형입니다. 각 버전에는 해당 버전의 추가 액세스 URL을 생성하는 태그가 연결될 수 있습니다.

트래픽 사양은 다음을 통해 수정할 수 있습니다.

- Service 오브젝트의 YAML을 직접 편집합니다.
- Knative(kn) CLI --traffic 플래그 사용.
- OpenShift Container Platform 웹 콘솔 사용.

Knative 서비스를 생성할 때 기본 traffic 사양 설정이 없습니다.

4.4.2. 트래픽 사양 예

다음 예에서는 traffic의 100%가 서비스의 최신 버전으로 라우팅되는 트래픽 사양을 보여줍니다. status에서 latestRevision의 최신 버전의 이름을 볼 수 있습니다.

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
...
  traffic:
  - latestRevision: true
    percent: 100
status:
  ...
  traffic:
  - percent: 100
    revisionName: example-service
```

다음 예에서는 traffic의 100%가 current로 태그가 지정된 버전으로 라우팅되고 해당 버전의 이름이 example-service로 지정된 트래픽 사양을 보여줍니다. 트래픽이 라우팅되지 않더라도 latest로 태그된 버전을 사용할 수 있습니다.

-

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
...
  traffic:
  - tag: current
    revisionName: example-service
    percent: 100
  - tag: latest
    latestRevision: true
    percent: 0
```

다음 예제에서는 **traffic** 사양의 버전 목록을 확장하여 여러 버전으로 트래픽을 분할하는 방법을 보여줍니다. 이 예제에서는 **50%**의 트래픽을 현재 태그된 버전으로, 후보로 태그된 버전에 대한 트래픽의 **50%**를 보냅니다. 트래픽이 라우팅되지 않더라도 **latest**로 태그된 버전을 사용할 수 있습니다.

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
...
  traffic:
  - tag: current
    revisionName: example-service-1
    percent: 50
  - tag: candidate
    revisionName: example-service-2
    percent: 50
  - tag: latest
    latestRevision: true
    percent: 0
```

4.4.3. Knative CLI를 사용하여 트래픽 분할

Knative(kn) CLI를 사용하여 트래픽 분할을 생성하면 **YAML** 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다. **kn service update** 명령을 사용하여 서비스 버전 간에 트래픽을 분할할 수 있습니다.

4.4.3.1. Knative CLI를 사용하여 트래픽 분할 생성

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다.

- Knative(kn) CLI가 설치되어 있습니다.
- Knative 서비스를 생성했습니다.

절차

- 표준 kn service update 명령과 함께 --traffic 태그를 사용하여 서비스 버전 및 라우팅할 트래픽의 백분율을 지정합니다.

  명령 예

```
$ kn service update <service_name> --traffic <revision>=<percentage>
```

  다음과 같습니다.

  - <service_name>은 트래픽 라우팅을 구성하려는 Knative 서비스의 이름입니다.
  - <revision>은 일정 비율의 트래픽을 수신하도록 구성하려는 버전입니다. 버전 이름 또는 --tag 플래그를 사용하여 버전에 할당한 태그를 지정할 수 있습니다.
  - < percentage>는 지정된 버전으로 보낼 트래픽의 백분율입니다.

- 선택 사항: --traffic 플래그는 한 명령에 여러 번 지정할 수 있습니다. 예를 들어 @latest 로 태그된 리비전과 stable 이라는 버전이 있는 경우 다음과 같이 각 버전으로 분할하려는 트래픽 백분율을 지정할 수 있습니다.

  명령 예

```
$ kn service update example-service --traffic @latest=20,stable=80
```

버전이 여러 개 있고 마지막 버전으로 분할되어야 하는 트래픽 비율을 지정하지 않으면 **--traffic** 플래그는 이 값을 자동으로 계산할 수 있습니다. 예를 들어 이름이 **. example** 인 세 번째 버전이 있고 다음 명령을 사용합니다.

명령 예

```
$ kn service update example-service --traffic @latest=10,stable=60
```

트래픽의 나머지 **30%**는 지정되지 않은 경우에도 예제 버전으로 분할됩니다.

4.4.4. 트래픽 분할을 위한 CLI 플래그

Knative(kn) CLI는 **kn service update** 명령의 일부로 서비스의 트래픽 블록에서 트래픽 작업을 지원합니다.

4.4.4.1. Knative CLI 트래픽 분할 플래그

다음 테이블에는 트래픽 분할 플래그, 값 형식, 플래그에서 수행하는 작업이 요약되어 있습니다. 반복 열은 **kn service update** 명령에서 특정 플래그 값을 반복할 수 있는지를 나타냅니다.

플래그	값	작업	반복
--traffic	**RevisionName=Percent**	**RevisionName**에 **Percent** 트래픽 제공	예
--traffic	**Tag=Percent**	**Tag**가 있는 버전에 **Percent** 트래픽 제공	예
--traffic	**@latest=Percent**	최신 준비 버전에 **Percent** 트래픽 제공	아니요
--tag	**RevisionName=Tag**	**RevisionName**에 **Tag** 지정	예

플래그	값	작업	반복
--tag	**@latest=Tag**	최근 준비된 버전에 **Tag** 지정	아니요
--untag	**Tag**	버전에서 **Tag** 제거	예

4.4.4.1.1. 여러 플래그 및 우선 순위

모든 트래픽 관련 플래그는 단일 **kn service update** 명령을 사용하여 지정할 수 있습니다. **kn**은 이러한 플래그의 우선순위를 정의합니다. 명령을 사용할 때 지정된 플래그의 순서는 고려하지 않습니다.

kn에 의해 평가되는 플래그의 우선순위는 다음과 같습니다.

1. **--untag:** 이 플래그가 있는 참조된 버전은 모두 트래픽 블록에서 제거됩니다.
2. **--tag:** 버전에는 트래픽 블록에 지정된 대로 태그가 지정됩니다.
3. **--traffic:** 참조된 버전에는 트래픽 분할의 일부가 할당됩니다.

버전에 태그를 추가한 다음 설정한 태그에 따라 트래픽을 분할할 수 있습니다.

4.4.4.1.2. 개정버전 사용자 정의 URL

kn service update 명령을 사용하여 서비스에 **--tag** 플래그를 할당하면 서비스를 업데이트할 때 생성되는 버전에 대한 사용자 정의 **URL**이 생성됩니다. 사용자 정의 **URL**은 **https://<tag>-<service_name>-<namespace>.<domain >** 또는 **http://<tag>-<service_name>-<namespace>.<domain >** 을 따릅니다.

--tag 및 **--untag** 플래그는 다음 구문을 사용합니다.

- 하나의 값이 필요합니다.
- 서비스의 트래픽 블록에 있는 고유한 태그를 나타냅니다.

- 하나의 명령에서 여러 번 지정할 수 있습니다.

4.4.4.1.2.1. 예: 태그를 버전에 할당

다음 예제에서는 **latest** 태그를 **example-revision**이라는 버전에 할당합니다.

```
$ kn service update <service_name> --tag @latest=example-tag
```

4.4.4.1.2.2. 예: 버전에서 태그 제거

--untag 플래그를 사용하여 사용자 정의 **URL**을 제거하도록 태그를 제거할 수 있습니다.

참고

개정 버전에 해당 태그가 제거되고 트래픽의 **0%**가 할당되면 개정 버전이 트래픽 블록에서 완전히 제거됩니다.

다음 명령은 **example-revision**이라는 버전에서 모든 태그를 제거합니다.

```
$ kn service update <service_name> --untag example-tag
```

4.4.5. 버전 간 트래픽 분할

서버리스 애플리케이션을 생성하면 **OpenShift Container Platform** 웹 콘솔의 개발자 화면의 토폴로지 보기에 애플리케이션이 표시됩니다. 애플리케이션 버전은 노드에서 나타내며 **Knative** 서비스는 노드 주변의 사각형으로 표시됩니다.

코드 또는 서비스 구성을 새로 변경하면 지정된 시간에 코드 스냅샷인 새 버전이 생성됩니다. 서비스의 경우 필요에 따라 서비스를 분할하고 다른 버전으로 라우팅하여 서비스 버전 간 트래픽을 관리할 수 있습니다.

4.4.5.1. OpenShift Container Platform 웹 콘솔을 사용하여 버전 간 트래픽 관리

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다.

- **OpenShift Container Platform** 웹 콘솔에 로그인했습니다.

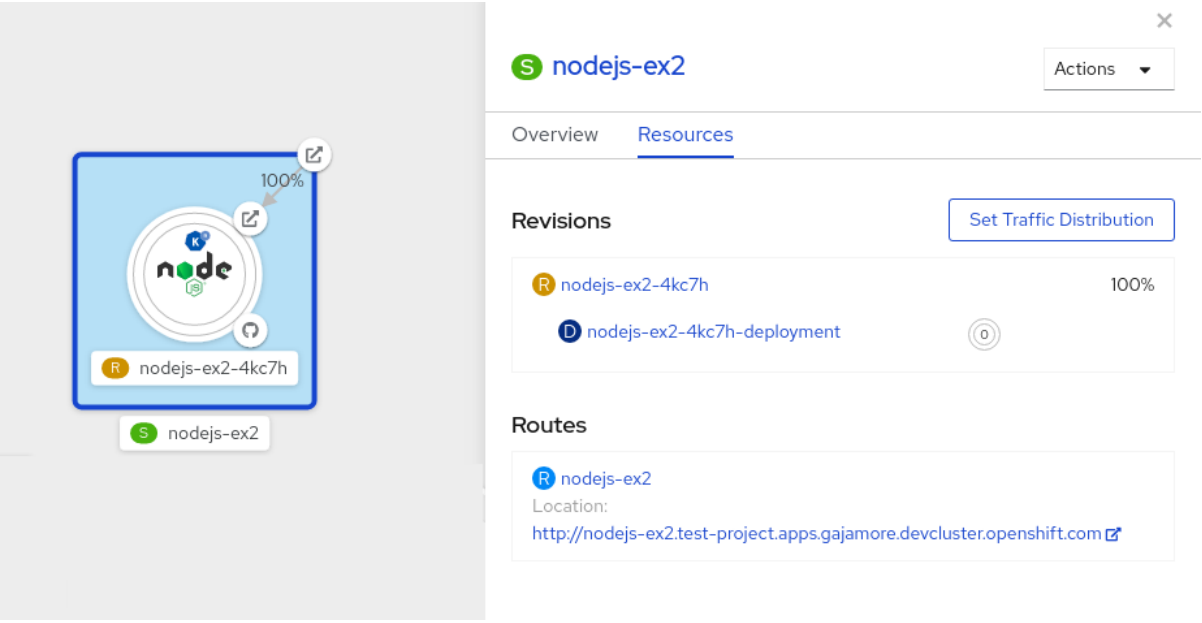
절차

토폴로지 보기에서 애플리케이션의 다양한 버전 간 트래픽을 분할하려면 다음을 수행합니다.

1. **Knative** 서비스를 클릭하여 측면 패널에서 개요를 확인합니다.

2. 리소스 탭을 클릭하여 서비스의 버전 및 경로로 구성된 목록을 확인합니다.

그림 **4.1.** 서버리스 애플리케이션



3. 측면 패널 상단에 **S** 아이콘으로 표시된 서비스를 클릭하여 서비스 세부 정보 개요를 확인합니다.

4. **YAML** 탭을 클릭하고 **YAML** 편집기에서 서비스 구성을 수정한 다음 저장을 클릭합니다. 예를 들면 **timeoutseconds**를 **300**에서 **301**로 변경합니다. 이러한 구성 변경으로 인해 새 버전이 트리거됩니다. 토폴로지 보기에 최신 버전이 표시되고 서비스의 리소스 탭에 두 가지 버전이 표시됩니다.

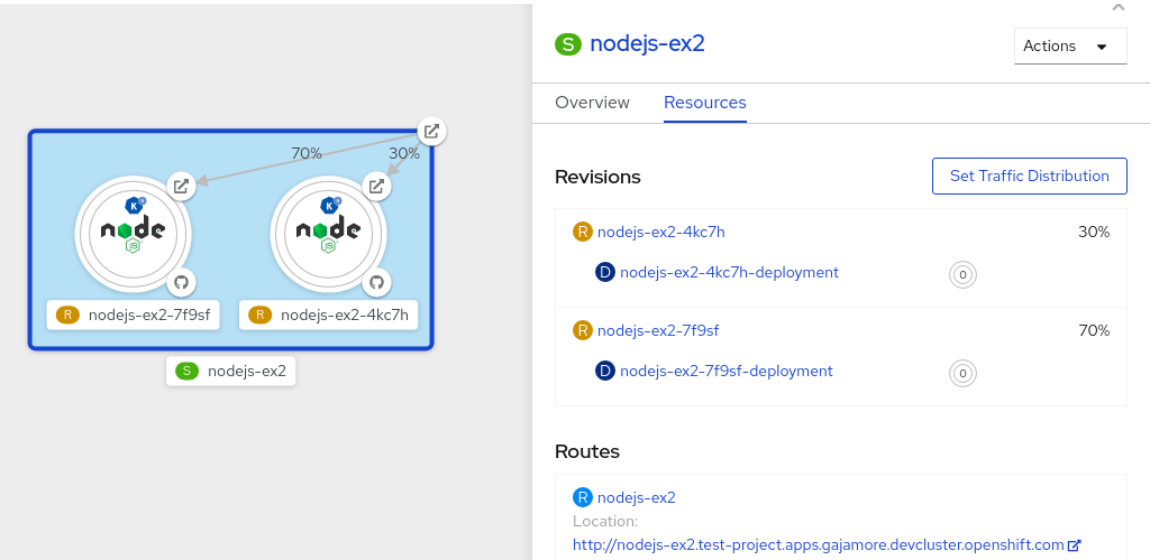
5. 리소스 탭에서 트래픽 분산 설정을 클릭하여 트래픽 분배 대화 상자를 확인합니다.

   a. 분할 필드에 두 버전의 분할 트래픽 백분율 부분을 추가합니다.

b. 두 버전에 대한 사용자 정의 URL을 생성하도록 태그를 추가합니다.

c. 저장을 클릭하여 토폴로지 보기에서 두 버전을 나타내는 두 노드를 확인합니다.

그림 4.2. 서버리스 애플리케이션의 버전



4.4.6. Blue-Green 전략을 사용하여 트래픽 재실행

blue-green 배포 전략을 사용하여 프로덕션 버전에서 새 버전으로 트래픽을 안전하게 다시 라우팅할 수 있습니다.

4.4.6.1. blue-green 배포 전략을 사용하여 트래픽 라우팅 및 관리

사전 요구 사항

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

절차

1. 애플리케이션을 Knative 서비스로 생성하고 배포합니다.
2. 다음 명령의 출력을 확인하여 서비스를 배포할 때 생성된 첫 번째 버전의 이름을 찾습니다.

```
$ oc get ksvc <service_name> -o=jsonpath='{.status.latestCreatedRevisionName}'
```

명령 예

```
$ oc get ksvc example-service -o=jsonpath='{.status.latestCreatedRevisionName}'
```

출력 예

```
$ example-service-00001
```

3. 다음 YAML을 서비스 spec에 추가하여 인바운드 트래픽을 버전으로 전송합니다.

```
...
spec:
  traffic:
    - revisionName: <first_revision_name>
      percent: 100 # All traffic goes to this revision
...
```

4. 다음 명령을 실행하여 얻은 URL 출력에서 앱을 볼 수 있는지 확인합니다.

```
$ oc get ksvc <service_name>
```

5. 서비스의 template 사양에서 하나 이상의 필드를 수정하고 재배포하여 애플리케이션의 두 번째 버전을 배포합니다. 예를 들어 서비스의 image 또는 env 환경 변수를 수정할 수 있습니다. 서비스 YAML 파일을 적용하거나 Knative(kn) CLI를 설치한 경우 kn service update 명령을 사용하여 서비스를 재배포할 수 있습니다.

6. 다음 명령을 실행하여 서비스를 재배포할 때 생성된 두 번째 최신 버전의 이름을 찾습니다.

```
$ oc get ksvc <service_name> -o=jsonpath='{.status.latestCreatedRevisionName}'
```

이때 서비스의 첫 번째 버전과 두 번째 버전이 모두 배포되고 실행됩니다.

7. 다른 모든 트래픽을 첫 번째 버전으로 전송하면서 두 번째 버전에 대한 새 테스트 끝점을 생성하도록 기존 서비스를 업데이트합니다.

테스트 끝점을 사용하여 업데이트된 서비스 사양의 예

```
...
spec:
  traffic:
    - revisionName: <first_revision_name>
      percent: 100 # All traffic is still being routed to the first revision
    - revisionName: <second_revision_name>
      percent: 0 # No traffic is routed to the second revision
      tag: v2 # A named route
...
```

YAML 리소스를 다시 적용하여 이 서비스를 재배포하면 애플리케이션의 두 번째 버전이 준비됩니다. 기본 URL의 두 번째 버전으로 라우팅되는 트래픽이 없으며 Knative는 새로 배포된 버전을 테스트하기 위해 v2라는 새 서비스를 생성합니다.

8. 다음 명령을 실행하여 두 번째 버전에 대한 새 서비스의 URL을 가져옵니다.

```
$ oc get ksvc <service_name> --output jsonpath="{.status.traffic[*].url}"
```

이 URL을 사용하여 트래픽을 라우팅하기 전에 애플리케이션의 새 버전이 예상대로 작동하는지 확인할 수 있습니다.

9. 트래픽의 50%가 첫 번째 버전으로 전송되고 50%가 두 번째 버전으로 전송되도록 기존 서비스를 다시 업데이트합니다.

버전 간에 트래픽을 50/50으로 분할하는 업데이트된 서비스 사양 분할 예

```
...
```

```
spec:
  traffic:
    - revisionName: <first_revision_name>
      percent: 50
    - revisionName: <second_revision_name>
      percent: 50
      tag: v2
...
```

10. 모든 트래픽을 새 버전의 앱으로 라우팅할 준비가 되면 두 번째 버전으로 트래픽의 **100%**를 보내도록 서비스를 다시 업데이트합니다.

두 번째 버전으로 모든 트래픽을 전송하는 업데이트된 서비스 사양의 예

```
...
spec:
  traffic:
    - revisionName: <first_revision_name>
      percent: 0
    - revisionName: <second_revision_name>
      percent: 100
      tag: v2
...
```

작은 정보

버전을 롤백하지 않으려는 경우 첫 번째 버전을 트래픽의 **0%**로 설정하는 대신 제거할 수 있습니다. 라우팅할 수 없는 버전 오브젝트는 가비지 수집됩니다.

11. 첫 번째 버전의 **URL**을 방문하여 이전 버전의 앱으로 더 이상 트래픽이 전송되지 않는지 확인합니다.

4.5. 외부 및 인그레스 라우팅

4.5.1. 라우팅 개요

Knative에서는 OpenShift Container Platform TLS 종료를 활용하여 Knative 서비스에 대한 라우팅을 제공합니다. Knative 서비스가 생성되면 서비스에 대해 OpenShift Container Platform 경로가 자동으로 생성됩니다. 이 경로는 OpenShift Serverless Operator에서 관리합니다. OpenShift Container Platform 경로는 OpenShift Container Platform 클러스터와 동일한 도메인을 통해 Knative 서비스를 표시합니다.

OpenShift Container Platform 라우팅에 대한 Operator의 제어를 비활성화하여 대신 TLS 인증서를 직접 사용하도록 Knative 경로를 구성할 수 있습니다.

또한 Knative 경로를 OpenShift Container Platform 경로와 함께 사용하면 트래픽 분할과 같은 세분화된 라우팅 기능을 추가로 제공할 수 있습니다.

4.5.1.1. 추가 리소스

- 경로별 주석

4.5.2. 레이블 및 주석 사용자 정의

OpenShift Container Platform 경로는 사용자 정의 레이블 및 주석을 사용할 수 있으며 Knative 서비스의 metadata 사양을 수정하여 구성할 수 있습니다. 사용자 정의 레이블 및 주석은 서비스에서 Knative 경로로 전달된 다음 Knative Ingress로 전달되고 마지막으로 OpenShift Container Platform 경로로 전달됩니다.

4.5.2.1. OpenShift Container Platform 경로에 대한 레이블 및 주석 사용자 정의

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving 구성 요소가 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- OpenShift CLI(oc)를 설치합니다.

절차

1. OpenShift Container Platform 경로에 전달할 레이블 또는 주석이 포함된 Knative 서비스를 생성합니다.
   - YAML을 사용하여 서비스를 생성하려면 다음을 수행합니다.

YAML을 사용하여 생성한 서비스 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: <service_name>
  labels:
    <label_name>: <label_value>
  annotations:
    <annotation_name>: <annotation_value>
...
```

- Knative(kn) CLI를 사용하여 서비스를 생성하려면 다음을 입력합니다.

kn 명령을 사용하여 생성된 서비스 예

```
$ kn service create <service_name> \
  --image=<image> \
  --annotation <annotation_name>=<annotation_value> \
  --label <label_value>=<label_value>
```

2. 다음 명령의 출력을 확인하여 추가한 주석 또는 레이블을 사용하여 OpenShift Container Platform 경로가 생성되었는지 확인합니다.

확인을 위한 명령 예

```
$ oc get routes.route.openshift.io \
    -l serving.knative.openshift.io/ingressName=<service_name> \ 1
    -l serving.knative.openshift.io/ingressNamespace=<service_namespace> \ 2
    -n knative-serving-ingress -o yaml \
       | grep -e "<label_name>: \"<label_value>\""  -e "<annotation_name>:
<annotation_value>" 3
```

1

서비스 이름을 사용합니다.

2

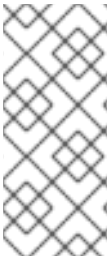
서비스가 생성된 네임스페이스를 사용합니다.

3

레이블 및 주석 이름과 값의 값을 사용합니다.

4.5.3. Knative 서비스의 경로 구성

OpenShift Container Platform에서 TLS 인증서를 사용하도록 Knative 서비스를 구성하려면 OpenShift Serverless Operator의 서비스 경로 자동 생성 기능을 비활성화하고 대신 해당 서비스에 대한 경로를 수동으로 생성해야 합니다.



참고

다음 절차를 완료하면 knative-serving-ingress 네임스페이스의 기본 OpenShift Container Platform 경로가 생성되지 않습니다. 그러나 애플리케이션의 Knative 경로는 이 네임스페이스에 계속 생성됩니다.

4.5.3.1. Knative 서비스에 대한 OpenShift Container Platform 경로 구성

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving 구성 요소가 OpenShift Container Platform 클러스터에 설치되어 있어야 합니다.
- OpenShift CLI(oc)를 설치합니다.

절차

1. serving.knative.openshift.io/disableRoute=true 주석이 포함된 Knative 서비스를 생성합니다.

중요

serving.knative.openshift.io/disableRoute=true 주석은 OpenShift Serverless에서 경로를 자동으로 생성하지 않도록 지시합니다. 그러나 서비스는 여전히 URL을 표시하며 Ready 상태에 도달합니다. 이 URL은 URL의 호스트 이름과 동일한 호스트 이름으로 자체 경로를 생성할 때까지 외부에서 작동하지 않습니다.

a. Knative 서비스 리소스를 생성합니다.

리소스 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    serving.knative.openshift.io/disableRoute: "true"
spec:
  template:
    spec:
      containers:
      - image: <image>
...
```

b. Service 리소스를 적용합니다.

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

c. 선택 사항입니다. kn service create 명령을 사용하여 Knative 서비스를 생성합니다.

kn 명령 예제

```
$ kn service create <service_name> \
  --image=gcr.io/knative-samples/helloworld-go \
  --annotation serving.knative.openshift.io/disableRoute=true
```

2. 서비스에 OpenShift Container Platform 경로가 생성되지 않았는지 확인합니다.

명령 예

```
$ $ oc get routes.route.openshift.io \
  -l serving.knative.openshift.io/ingressName=$KSERVICE_NAME \
  -l serving.knative.openshift.io/ingressNamespace=$KSERVICE_NAMESPACE \
  -n knative-serving-ingress
```

다음 출력이 표시됩니다.

```
No resources found in knative-serving-ingress namespace.
```

3. knative-serving-ingress 네임스페이스에 Route 리소스를 생성합니다.

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 600s (1)
  name: <route_name> (2)
  namespace: knative-serving-ingress (3)
spec:
  host: <service_host> (4)
  port:
    targetPort: http2
  to:
    kind: Service
    name: kourier
    weight: 100
  tls:
    insecureEdgeTerminationPolicy: Allow
    termination: edge (5)
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
```

```
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE-----
  caCertificate: |-
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE----
wildcardPolicy: None
```

1. OpenShift Container Platform 경로에 대한 타임아웃 값입니다. max-revision-timeout-seconds 설정과 동일한 값으로 설정해야 합니다(기본값: 600s).
2. OpenShift Container Platform 경로의 이름입니다.
3. OpenShift Container Platform 경로의 네임스페이스입니다. knative-serving-ingress여야 합니다.
4. 외부 액세스를 위한 호스트 이름입니다. 이 값을 <service_name>-<service_namespace>.<domain>으로 설정할 수 있습니다.
5. 사용할 인증서입니다. 현재는 edge 종료만 지원됩니다.

4. Route 리소스를 적용합니다.

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

4.5.4. 글로벌 HTTPS 리디렉션

HTTPS 리디렉션은 들어오는 HTTP 요청에 대한 리디렉션을 제공합니다. 이러한 리디렉션된 HTTP 요청은 암호화됩니다. KnativeServing CR(사용자 정의 리소스)에 httpProtocol 사양을 구성하여 클러스터의 모든 서비스에 대해 HTTPS 리디렉션을 활성화할 수 있습니다.

4.5.4.1. HTTPS 리디렉션 글로벌 설정

HTTPS 리디렉션을 활성화하는 KnativeServing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    network:
      httpProtocol: "redirected"
...
```

4.5.5. 외부 경로에 대한 URL 스키마

외부 경로의 URL 스키마는 강화된 보안을 위해 기본적으로 HTTPS로 설정됩니다. 이 스키마는 KnativeServing CR(사용자 정의 리소스) 사양의 default-external-scheme 키로 결정됩니다.

4.5.5.1. 외부 경로의 URL 스키마 설정

기본 사양

```
...
spec:
  config:
    network:
      default-external-scheme: "https"
...
```

default-external-scheme 키를 수정하여 HTTP를 사용하도록 기본 사양을 덮어쓸 수 있습니다.

HTTP 덮어쓰기 사양

```
...
spec:
  config:
```

```
  network:
    default-external-scheme: "http"
...
```

4.5.6. 서비스별 HTTPS 리디렉션

networking.knative.dev/http-option 주석을 구성하여 서비스의 HTTPS 리디렉션을 활성화하거나 비활성화할 수 있습니다.

4.5.6.1. 서비스의 HTTPS 리디렉션

다음 예제에서는 Knative Service YAML 오브젝트에서 이 주석을 사용하는 방법을 보여줍니다.

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example
  namespace: default
  annotations:
    networking.knative.dev/http-option: "redirected"
spec:
  ...
```

4.5.7. 클러스터 로컬 가용성

기본적으로 Knative 서비스는 공용 IP 주소에 게시됩니다. 공용 IP 주소에 게시된다는 것은 Knative 서비스가 공용 애플리케이션이 되어 공개적으로 액세스할 수 있는 URL이 있음을 의미합니다.

공개적으로 액세스할 수 있는 URL은 클러스터 외부에서 액세스할 수 있습니다. 그러나 개발자는 클러스터 내부에서만 액세스할 수 있는 백엔드 서비스(*비공개 서비스*)를 빌드해야 할 수 있습니다. 개발자는 클러스터의 개별 서비스에 networking.knative.dev/visibility=cluster-local 레이블을 지정하여 비공개로 설정할 수 있습니다.

중요

OpenShift Serverless 1.15.0 및 최신 버전의 경우 serving.knative.dev/visibility 레이블을 더 이상 사용할 수 없습니다. 대신 networking.knative.dev/visibility 레이블을 사용하려면 기존 서비스를 업데이트해야 합니다.

4.5.7.1. 클러스터 가용성 설정을 클러스터 로컬로 설정

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- Knative 서비스를 생성했습니다.

절차

- networking.knative.dev/visibility=cluster-local 레블을 추가하여 서비스 가시성을 설정합니다.

```
$ oc label ksvc <service_name> networking.knative.dev/visibility=cluster-local
```

검증

- 다음 명령을 입력하고 출력을 검토하여 서비스의 URL이 http://<service_name>.<namespace>.svc.cluster.local 형식인지 확인합니다.

```
$ oc get ksvc
```

출력 예

```
NAME    URL                                        LATESTCREATED
LATESTREADY    READY   REASON
hello   http://hello.default.svc.cluster.local    hello-tx2g7
hello-tx2g7    True
```

4.5.7.2. 클러스터 로컬 서비스에 대한 TLS 인증 활성화

클러스터 로컬 서비스의 경우 Kourier 로컬 게이트웨이 kourier-internal 이 사용됩니다. Kourier 로컬 게이트웨이에 대해 TLS 트래픽을 사용하려면 로컬 게이트웨이에서 자체 서버 인증서를 구성해야 합니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 설치되어 있습니다.
- 관리자 권한이 있어야 합니다.
- OpenShift(oc) CLI를 설치했습니다.

절차

1. knative-serving-ingress 네임스페이스에 서버 인증서를 배포합니다.

```
$ export san="knative"
```

참고

이러한 인증서가 < app_name>.<namespace>.svc.cluster.local 에 대한 요청을 제공할 수 있도록 SAN(Subject Alternative Name) 검증이 필요합니다.

2. 루트 키 및 인증서를 생성합니다.

```
$ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 \
    -subj '/O=Example/CN=Example' \
    -keyout ca.key \
    -out ca.crt
```

3. SAN 검증을 사용하는 서버 키를 생성합니다.

```
$ openssl req -out tls.csr -newkey rsa:2048 -nodes -keyout tls.key \
  -subj "/CN=Example/O=Example" \
  -addext "subjectAltName = DNS:$san"
```

4. 서버 인증서를 생성합니다.

```
$ openssl x509 -req -extfile <(printf "subjectAltName=DNS:$san") \
  -days 365 -in tls.csr \
  -CA ca.crt -CAkey ca.key -CAcreateserial -out tls.crt
```

5. Kourier 로컬 게이트웨이의 시크릿을 구성합니다.

a. 이전 단계에서 생성한 인증서에서 knative-serving-ingress 네임스페이스에 보안을 배포합니다.

```
$ oc create -n knative-serving-ingress secret tls server-certs \
   --key=tls.key \
   --cert=tls.crt --dry-run=client -o yaml | oc apply -f -
```

b. Kourier 게이트웨이에서 생성한 보안을 사용하도록 KnativeServing CR(사용자 정의 리소스) 사양을 업데이트합니다.

KnativeServing CR의 예

```
...
spec:
  config:
    kourier:
      cluster-cert-secret: server-certs
...
```

Kourier 컨트롤러는 서비스를 재시작하지 않고 인증서를 설정하므로 Pod를 다시 시작할 필요가 없습니다.

클라이언트의 ca.crt 를 마운트하고, 포트 443 을 통해 TLS를 사용하여 Kourier 내부 서비스에 액세스할 수 있습니다.

4.5.8. Kourier Gateway 서비스 유형

Kourier 게이트웨이는 기본적으로 ClusterIP 서비스 유형으로 노출됩니다. 이 서비스 유형은 KnativeServing CR(사용자 정의 리소스)의 서비스 유형 Ingress 사양에 따라 결정됩니다.

기본 사양

```
...
spec:
  ingress:
    kourier:
      service-type: ClusterIP
...
```

4.5.8.1. Kourier Gateway 서비스 유형 설정

서비스 유형 사양을 수정하여 로드 밸런서 서비스 유형을 대신 사용하도록 기본 서비스 유형을 덮어쓸 수 있습니다.

LoadBalancer 덮어쓰기 사양

```
...
spec:
  ingress:
    kourier:
      service-type: LoadBalancer
...
```

4.5.9. HTTP2 및 gRPC 사용

OpenShift Serverless에서는 비보안 또는 엣지 종료 경로만 지원합니다. 비보안 또는 엣지 종료 경로에서는 OpenShift Container Platform에서 HTTP2를 지원하지 않습니다. 또한 이러한 경로는 gRPC가 HTTP2에 의해 전송되기 때문에 gRPC를 지원하지 않습니다. 애플리케이션에서 이러한 프로토콜을 사용하는 경우 수신 게이트웨이를 사용하여 애플리케이션을 직접 호출해야 합니다. 이를 위해서는 수신 게이트웨이의 공용 주소와 애플리케이션의 특정 호스트를 찾아야 합니다.

4.5.9.1. HTTP2 및 gRPC를 사용하여 서버리스 애플리케이션과 상호 작용

중요

이 방법은 **LoadBalancer** 서비스 유형을 사용하여 **Kourier Gateway**를 노출해야 합니다. **KnativeServing CRD**(사용자 정의 리소스 정의)에 다음 **YAML**을 추가하여 구성할 수 있습니다.

```
...
spec:
  ingress:
    kourier:
      service-type: LoadBalancer
...
```

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다.
- **OpenShift CLI**(**oc**)를 설치합니다.
- **Knative** 서비스를 생성했습니다.

절차

1. 애플리케이션 호스트를 찾습니다. *서버리스 애플리케이션 배포 확인*에 있는 지침을 참조하십시오.
2. 수신 게이트웨이의 공용 주소를 찾습니다.

```
$ oc -n knative-serving-ingress get svc kourier
```

출력 예

```
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)
AGE
kourier   LoadBalancer   172.30.51.103   a83e86291bcdd11e993af02b7a65e514-
33544245.us-east-1.elb.amazonaws.com   80:31380/TCP,443:31390/TCP   67m
```

공용 주소는 **EXTERNAL-IP** 필드에 있으며 이 경우 **a83e86291bcdd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com**입니다.

3. **HTTP** 요청의 호스트 헤더를 애플리케이션의 호스트로 수동으로 설정하되 요청 자체는 수신 게이트웨이의 공개 주소로 지정합니다.

```
$ curl -H "Host: hello-default.example.com" a83e86291bcdd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com
```

출력 예

```
Hello Serverless!
```

수신 게이트웨이에 대한 요청을 직접 지시하는 동안 애플리케이션 호스트에 기관을 설정하여 **gRPC** 요청을 생성할 수도 있습니다.

```
grpc.Dial(
    "a83e86291bcdd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com:80",
    grpc.WithAuthority("hello-default.example.com:80"),
    grpc.WithInsecure(),
)
```

참고

위 예제와 같이 각 포트(기본값 **80**)를 두 호스트 모두에 추가해야 합니다.

4.6. KNATIVE 서비스에 대한 액세스 구성

4.6.1. Knative 서비스의 JSON Web Token 인증 설정

OpenShift Serverless에는 현재 사용자 정의 권한 부여 기능이 없습니다. 배포에 사용자 정의 권한 부여를 추가하려면 **OpenShift Serverless**를 **Red Hat OpenShift Service Mesh**와 통합한 다음 **Knative** 서비스에 대해 **JSON** 웹 토큰(**JWT**) 인증 및 사이드카 삽입을 구성해야 합니다.

4.6.2. Service Mesh 2.x에서 JSON 웹 토큰 인증 사용

Service Mesh 2.x 및 OpenShift Serverless를 사용하여 Knative 서비스와 함께 JSON 웹 토큰(JWT) 인증을 사용할 수 있습니다. 이렇게 하려면 ServiceMeshMemberRoll 오브젝트의 멤버인 애플리케이션 네임스페이스에서 인증 요청 및 정책을 생성해야 합니다. 서비스에 대한 사이드카 삽입도 활성화해야 합니다.

4.6.2.1. Service Mesh 2.x 및 OpenShift Serverless에 대한 JSON 웹 토큰 인증 구성

중요

knative-serving 및 knative-serving-ingress와 같은 시스템 네임스페이스의 Pod에 Kourier가 활성화되어 있는 경우 사이드카를 삽입할 수 없습니다.

이러한 네임스페이스의 Pod에 사이드카를 삽입해야 하는 경우 *Service Mesh와 OpenShift Serverless 통합에 대한 OpenShift Serverless* 문서를 참조하십시오.

사전 요구 사항

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

절차

1. 서비스에 sidecar.istio.io/inject="true" 주석을 추가합니다.

서비스의 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: <service_name>
spec:
```

```
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true" 1
        sidecar.istio.io/rewriteAppHTTPProbers: "true" 2
...
```

1 sidecar.istio.io/inject="true" 주석을 추가합니다.

2 OpenShift Serverless 버전 1.14.0 이상은 기본적으로 Knative 서비스에 대한 준비 상태 프로브로 사용하므로 Knative 서비스에서 주석 sidecar.istio.io/rewriteAppHTTPProbers: "true" 를 설정해야 합니다.

2. Service 리소스를 적용합니다.

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

3. ServiceMeshMemberRoll 오브젝트의 멤버인 각 서버리스 애플리케이션 네임스페이스에 RequestAuthentication 리소스를 생성합니다.

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: <namespace>
spec:
  jwtRules:
  - issuer: testing@secure.istio.io
    jwksUri: https://raw.githubusercontent.com/istio/istio/release-
1.8/security/tools/jwt/samples/jwks.json
```

4. RequestAuthentication 리소스를 적용합니다.

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

5. 다음 AuthorizationPolicy 리소스를 생성하여 ServiceMeshMemberRoll 오브젝트의 멤버인 각 서버리스 애플리케이션 네임스페이스의 시스템 Pod에서 RequestAuthenticaton 리소스

에 대한 액세스를 허용합니다.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allowlist-by-paths
 namespace: <namespace>
spec:
 action: ALLOW
 rules:
 - to:
  - operation:
      paths:
      - /metrics (1)
      - /healthz (2)
```

(1) 시스템 Pod별 지표를 수집하는 애플리케이션의 경로입니다.

(2) 시스템 Pod별로 검색할 애플리케이션의 경로입니다.

6. AuthorizationPolicy 리소스를 적용합니다.

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

7. ServiceMeshMemberRoll 오브젝트의 멤버인 각 서버리스 애플리케이션 네임스페이스에서 다음 AuthorizationPolicy 리소스를 생성합니다.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: require-jwt
 namespace: <namespace>
spec:
 action: ALLOW
 rules:
 - from:
  - source:
     requestPrincipals: ["testing@secure.istio.io/testing@secure.istio.io"]
```

8. AuthorizationPolicy 리소스를 적용합니다.

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

검증

1. Knative 서비스 URL을 가져오기 위해 curl 요청을 사용하면 해당 요청이 거부됩니다.

명령 예

```
$ curl http://hello-example-1-default.apps.mycluster.example.com/
```

출력 예

```
RBAC: access denied
```

2. 유효한 JWT로 요청을 확인합니다.

a. 유효한 JWT 토큰을 가져옵니다.

```
$ TOKEN=$(curl https://raw.githubusercontent.com/istio/istio/release-
1.8/security/tools/jwt/samples/demo.jwt -s) && echo "$TOKEN" | cut -d '.' -f2 - |
base64 --decode -
```

b. curl 요청 헤더에 유효한 토큰을 사용하여 서비스에 액세스합니다.

```
$ curl -H "Authorization: Bearer $TOKEN"  http://hello-example-1-
default.apps.example.com
```

그러면 요청이 허용됩니다.

출력 예

```
Hello OpenShift!
```

4.6.3. Service Mesh 1.x에서 JSON 웹 토큰 인증 사용

Service Mesh 1.x 및 OpenShift Serverless를 사용하여 Knative 서비스와 함께 JSON 웹 토큰(JWT) 인증을 사용할 수 있습니다. 이렇게 하려면 ServiceMeshMemberRoll 오브젝트의 멤버인 애플리케이션 네임스페이스에 정책을 만들어야 합니다. 서비스에 대한 사이드카 삽입도 활성화해야 합니다.

4.6.3.1. Service Mesh 1.x 및 OpenShift Serverless에 대한 JSON 웹 토큰 인증 구성

중요

knative-serving 및 knative-serving-ingress와 같은 시스템 네임스페이스의 Pod에 Kourier가 활성화되어 있는 경우 사이드카를 삽입할 수 없습니다.

이러한 네임스페이스의 Pod에 사이드카를 삽입해야 하는 경우 *Service Mesh와 OpenShift Serverless 통합에 대한 OpenShift Serverless* 문서를 참조하십시오.

사전 요구 사항

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

절차

1. 서비스에 sidecar.istio.io/inject="true" 주석을 추가합니다.

서비스의 예

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

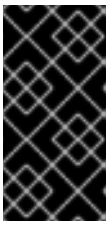
1 sidecar.istio.io/inject="true" 주석을 추가합니다.

2 OpenShift Serverless 버전 1.14.0 이상은 기본적으로 Knative 서비스에 대한 준비 상태 프로브로 사용하므로 Knative 서비스에서 주석 sidecar.istio.io/rewriteAppHTTPProbers: "true" 를 설정해야 합니다.

2. Service 리소스를 적용합니다.

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

3. 유효한 JWT(JSON 웹 토큰)가 있는 요청만 허용하는 ServiceMeshMemberRoll 오브젝트의 멤버인 서버리스 애플리케이션 네임스페이스에 정책을 생성합니다.



중요

/metrics 및 /healthz 경로는 knative-serving 네임스페이스의 시스템 Pod에서 액세스하므로 excludedPaths에 포함되어야 합니다.

```
apiVersion: authentication.istio.io/v1alpha1
kind: Policy
metadata:
  name: default
```

```
  namespace: <namespace>
spec:
  origins:
  - jwt:
      issuer: testing@secure.istio.io
      jwksUri: "https://raw.githubusercontent.com/istio/istio/release-
1.6/security/tools/jwt/samples/jwks.json"
      triggerRules:
      - excludedPaths:
        - prefix: /metrics (1)
        - prefix: /healthz (2)
  principalBinding: USE_ORIGIN
```

(1) 시스템 Pod별 지표를 수집하는 애플리케이션의 경로입니다.

(2) 시스템 Pod별로 검색할 애플리케이션의 경로입니다.

4. Policy 리소스를 적용합니다.

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

검증

1. Knative 서비스 URL을 가져오기 위해 curl 요청을 사용하면 해당 요청이 거부됩니다.

```
$ curl http://hello-example-default.apps.mycluster.example.com/
```

출력 예

```
Origin authentication failed.
```

2. 유효한 JWT로 요청을 확인합니다.

   a. 유효한 JWT 토큰을 가져옵니다.

```
$ TOKEN=$(curl https://raw.githubusercontent.com/istio/istio/release-
1.6/security/tools/jwt/samples/demo.jwt -s) && echo "$TOKEN" | cut -d '.' -f2 - |
base64 --decode -
```

b. curl 요청 헤더에 유효한 토큰을 사용하여 서비스에 액세스합니다.

```
$ curl http://hello-example-default.apps.mycluster.example.com/ -H "Authorization:
Bearer $TOKEN"
```

그러면 요청이 허용됩니다.

출력 예

```
Hello OpenShift!
```

4.7. KNATIVE 서비스의 사용자 정의 도메인 구성

4.7.1. Knative 서비스의 사용자 정의 도메인 구성

Knative 서비스에는 클러스터 구성에 따라 기본 도메인 이름이 자동으로 할당됩니다. 예를 들면 <service_name>-<namespace>.example.com 입니다. 보유한 사용자 정의 도메인 이름을 Knative 서비스에 매핑하여 Knative 서비스의 도메인을 사용자 지정할 수 있습니다.

서비스에 대한 DomainMapping 리소스를 생성하여 이 작업을 수행할 수 있습니다. 또한 여러 개의 DomainMapping 리소스를 생성하여 여러 도메인에 매핑하고 하위 도메인을 단일 서비스에 매핑할 수도 있습니다.

4.7.2. 사용자 정의 도메인 매핑

보유한 사용자 정의 도메인 이름을 Knative 서비스에 매핑하여 Knative 서비스의 도메인을 사용자 지정할 수 있습니다. 사용자 정의 도메인 이름을 CR(사용자 정의 리소스)에 매핑하려면 Knative 서비스 또는 Knative 경로와 같이 주소 지정 가능 대상 CR에 매핑하는 DomainMapping CR을 생성해야 합니다.

4.7.2.1. 사용자 정의 도메인 매핑 생성

보유한 사용자 정의 도메인 이름을 Knative 서비스에 매핑하여 Knative 서비스의 도메인을 사용자 지정할 수 있습니다. 사용자 정의 도메인 이름을 CR(사용자 정의 리소스)에 매핑하려면 Knative 서비스 또는 Knative 경로와 같이 주소 지정 가능 대상 CR에 매핑하는 DomainMapping CR을 생성해야 합니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- OpenShift CLI(oc)를 설치합니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Knative 서비스를 생성했으며 해당 서비스에 매핑할 사용자 정의 도메인을 제어할 수 있습니다.

참고

사용자 정의 도메인에서 OpenShift Container Platform 클러스터의 IP 주소를 참조해야 합니다.

절차

1. 매핑하려는 대상 CR과 동일한 네임스페이스에 DomainMapping CR을 포함하는 YAML 파일을 생성합니다.

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: <domain_name> 1
 namespace: <namespace> 2
spec:
 ref:
   name: <target_name> 3
   kind: <target_type> 4
   apiVersion: serving.knative.dev/v1
```

1. 대상 CR에 매핑할 사용자 정의 도메인 이름입니다.

2

DomainMapping CR 및 대상 CR의 네임스페이스입니다.

3

사용자 정의 도메인에 매핑할 대상 CR의 이름입니다.

4

사용자 지정 도메인에 매핑되는 CR 유형입니다.

서비스 도메인 매핑 예

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: example-domain
 namespace: default
spec:
 ref:
   name: example-service
   kind: Service
   apiVersion: serving.knative.dev/v1
```

경로 도메인 매핑 예

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: example-domain
 namespace: default
spec:
 ref:
   name: example-route
   kind: Route
   apiVersion: serving.knative.dev/v1
```

2. DomainMapping CR을 YAML 파일로 적용합니다.

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

4.7.3. Knative CLI를 사용하는 Knative 서비스의 사용자 정의 도메인

보유한 사용자 정의 도메인 이름을 Knative 서비스에 매핑하여 Knative 서비스의 도메인을 사용자 지정할 수 있습니다. Knative(kn) CLI를 사용하여 Knative 서비스 또는 Knative 경로와 같이 주소 지정 가능 대상 CR에 매핑되는 DomainMapping CR(사용자 정의 리소스)을 생성할 수 있습니다.

4.7.3.1. Knative CLI를 사용하여 사용자 정의 도메인 매핑 생성

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- Knative 서비스 또는 경로를 생성했으며 CR에 매핑할 사용자 정의 도메인을 제어할 수 있습니다.

참고

사용자 정의 도메인에서 OpenShift Container Platform 클러스터의 DNS를 가리켜야 합니다.

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

절차

- 현재 네임스페이스의 CR에 도메인을 매핑합니다.

```
$ kn domain create <domain_mapping_name> --ref <target_name>
```

명령 예

```
$ kn domain create example-domain-map --ref example-service
```

--ref 플래그는 도메인 매핑을 위해 주소 지정 가능한 대상 **CR**을 지정합니다.

--ref 플래그를 사용할 때 접두사가 지정되어 있지 않은 경우 대상이 현재 네임스페이스의 **Knative** 서비스라고 가정합니다.

- 지정된 네임스페이스의 **Knative** 서비스에 도메인을 매핑합니다.

```
$ kn domain create <domain_mapping_name> --ref
<ksvc:service_name:service_namespace>
```

명령 예

```
$ kn domain create example-domain-map --ref ksvc:example-service:example-
namespace
```

- 도메인을 **Knative** 경로에 매핑합니다.

```
$ kn domain create <domain_mapping_name> --ref <kroute:route_name>
```

명령 예

```
$ kn domain create example-domain-map --ref kroute:example-route
```

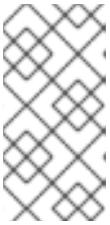
4.7.4. 개발자 관점을 사용한 도메인 매핑

보유한 사용자 정의 도메인 이름을 **Knative** 서비스에 매핑하여 **Knative** 서비스의 도메인을 사용자 지정할 수 있습니다. **OpenShift Container Platform** 웹 콘솔의 개발자 화면을 사용하여 **DomainMapping CR**(사용자 정의 리소스)을 **Knative** 서비스에 매핑할 수 있습니다.

4.7.4.1. 개발자 화면을 사용하여 사용자 정의 도메인을 서비스에 매핑

사전 요구 사항

- 웹 콘솔에 로그인했습니다.
- 개발자 화면에 있습니다.
- **OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다. 클러스터 관리자가 완료해야 합니다.
- 프로젝트를 생성했거나 **OpenShift Container Platform**에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- **Knative** 서비스를 생성했으며 해당 서비스에 매핑할 사용자 정의 도메인을 제어할 수 있습니다.



참고

사용자 정의 도메인에서 **OpenShift Container Platform** 클러스터의 **IP** 주소를 참조해야 합니다.

절차

1. 토폴로지 페이지로 이동합니다.
2. 도메인에 매핑할 서비스를 마우스 오른쪽 버튼으로 클릭하고 서비스 이름이 포함된 편집 옵션을 선택합니다. 예를 들어 서비스 이름이 **example-service** 인 경우 **Edit example-service** 옵션을 선택합니다.

3. 고급 옵션 섹션에서 고급 라우팅 옵션 표시를 클릭합니다.

   a. 서비스에 매핑할 도메인 매핑 CR이 이미 존재하는 경우 도메인 매핑 목록에서 선택할 수 있습니다.

   b. 새 도메인 매핑 CR을 생성하려면 박스에 도메인 이름을 입력하고 Create 옵션을 선택합니다. 예를 들어 example.com 을 입력하면 생성 옵션은 Create "example.com" 입니다.

4. 저장을 클릭하여 서비스에 대한 변경 사항을 저장합니다.

검증

1. 토폴로지 페이지로 이동합니다.
2. 생성한 서비스를 클릭합니다.
3. 서비스 정보 창의 리소스 탭에서 도메인 매핑 아래에 나열된 서비스에 매핑된 도메인을 확인할 수 있습니다.

4.7.5. 관리자 관점을 사용한 도메인 매핑

OpenShift Container Platform 웹 콘솔의 개발자 화면으로 전환하거나 Knative(kn) CLI 또는 YAML 파일을 사용하지 않으려면 OpenShift Container Platform 웹 콘솔의 관리자 화면을 사용할 수 있습니다.

4.7.5.1. 관리자 화면을 사용하여 사용자 정의 도메인을 서비스에 매핑

Knative 서비스에는 클러스터 구성에 따라 기본 도메인 이름이 자동으로 할당됩니다. 예를 들면 <service_name>-<namespace>.example.com 입니다. 보유한 사용자 정의 도메인 이름을 Knative 서비스에 매핑하여 Knative 서비스의 도메인을 사용자 지정할 수 있습니다.

서비스에 대한 DomainMapping 리소스를 생성하여 이 작업을 수행할 수 있습니다. 또한 여러 개의 DomainMapping 리소스를 생성하여 여러 도메인에 매핑하고 하위 도메인을 단일 서비스에 매핑할 수도 있습니다.

클러스터 관리자 권한이 있는 경우 OpenShift Container Platform 웹 콘솔의 관리자 화면을 사용하

여 DomainMapping CR(사용자 정의 리소스)을 생성할 수 있습니다.

사전 요구 사항

- 웹 콘솔에 로그인했습니다.
- 관리자 화면에 있습니다.
- OpenShift Serverless Operator를 설치했습니다.
- Knative Serving이 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Knative 서비스를 생성했으며 해당 서비스에 매핑할 사용자 정의 도메인을 제어할 수 있습니다.

참고

사용자 정의 도메인에서 OpenShift Container Platform 클러스터의 IP 주소를 참조해야 합니다.

절차

1. CustomResourceDefinitions 로 이동하여 검색 상자를 사용하여 DomainMapping CRD(사용자 정의 리소스 정의)를 찾습니다.
2. DomainMapping CRD를 클릭한 다음 Instances (인스턴스) 탭으로 이동합니다.
3. 도메인 매핑 생성을 클릭합니다.
4. 인스턴스에 대한 다음 정보가 포함되도록 DomainMapping CR의 YAML을 수정합니다.

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: <domain_name> 1
 namespace: <namespace> 2
spec:
 ref:
   name: <target_name> 3
   kind: <target_type> 4
   apiVersion: serving.knative.dev/v1
```

1 대상 CR에 매핑할 사용자 정의 도메인 이름입니다.

2 DomainMapping CR 및 대상 CR의 네임스페이스입니다.

3 사용자 정의 도메인에 매핑할 대상 CR의 이름입니다.

4 사용자 지정 도메인에 매핑되는 CR 유형입니다.

Knative 서비스에 대한 도메인 매핑 예

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: custom-ksvc-domain.example.com
 namespace: default
spec:
 ref:
   name: example-service
   kind: Service
   apiVersion: serving.knative.dev/v1
```

검증

- curl 요청을 사용하여 사용자 정의 도메인에 액세스합니다. 예를 들면 다음과 같습니다.

명령 예

```
$ curl custom-ksvc-domain.example.com
```

출력 예

```
Hello OpenShift!
```

4.7.6. TLS 인증서를 사용하여 매핑된 서비스 보안

4.7.6.1. TLS 인증서를 사용하여 사용자 정의 도메인으로 서비스 보안

Knative 서비스의 사용자 정의 도메인을 구성한 후 TLS 인증서를 사용하여 매핑된 서비스를 보호할 수 있습니다. 이렇게 하려면 Kubernetes TLS 시크릿을 생성한 다음 생성한 TLS 시크릿을 사용하도록 DomainMapping CR을 업데이트해야 합니다.

참고

Ingress에 net-istio 를 사용하고 security.dataPlane.mtls: true 를 사용하여 SMCP 를 통해 mTLS를 활성화하는 경우 Service Mesh는 OpenShift Serverless에 대한 DomainMapping 을 허용하지 않는 *.local 호스트에 대한 DestinationRule 을 배포합니다.

이 문제를 해결하려면 security.dataPlane.mtls: true 를 사용하는 대신 PeerAuthentication 을 배포하여 mTLS를 활성화합니다.

사전 요구 사항

- Knative 서비스에 대한 사용자 정의 도메인을 구성하고 작동 중인 DomainMapping CR이 있습니다.

- 인증 기관 공급자 또는 자체 서명된 인증서의 **TLS** 인증서가 있습니다.
- 인증 기관 공급자 또는 자체 서명된 인증서에서 인증서 및 키 파일을 가져왔습니다.
- **OpenShift CLI(oc)**를 설치합니다.

절차

1. **Kubernetes TLS** 시크릿을 생성합니다.

```
$ oc create secret tls <tls_secret_name> --cert=<path_to_certificate_file> --key=
<path_to_key_file>
```

2. **Red Hat OpenShift Service Mesh**를 **OpenShift Serverless** 설치의 수신으로 사용하는 경우 다음을 사용하여 **Kubernetes TLS** 시크릿에 레이블을 지정합니다.

```
"networking.internal.knative.dev/certificate-uid": "<value>"
```

cert-manager와 같은 타사 시크릿 공급자를 사용하는 경우, **Kubernetes TLS** 보안에 자동으로 레이블을 지정하도록 시크릿 관리자를 구성할 수 있습니다. **cert-manager** 사용자는 제공된 보안 템플릿을 사용하여 올바른 레이블이 있는 보안을 자동으로 생성할 수 있습니다. 이 경우 시크릿 필터링은 키를 기반으로만 수행되지만 이 값은 시크릿에 포함된 인증서 **ID**와 같은 유용한 정보를 제공할 수 있습니다.

참고

cert-manager Operator for Red Hat OpenShift는 기술 프리뷰 기능입니다. 자세한 내용은 **Red Hat OpenShift**용 **cert-manager Operator** 설치 설명서를 참조하십시오.

3. 생성한 **TLS** 보안을 사용하도록 **DomainMapping CR**을 업데이트합니다.

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
  name: <domain_name>
```

```
  namespace: <namespace>
spec:
  ref:
    name: <service_name>
    kind: Service
    apiVersion: serving.knative.dev/v1
# TLS block specifies the secret to be used
  tls:
    secretName: <tls_secret_name>
```

검증

1. DomainMapping CR 상태가 True 이고 출력의 URL 열에 스키마 https 가 있는 매핑된 도메인이 표시되는지 확인합니다.

```
$ oc get domainmapping <domain_name>
```

출력 예

```
NAME                URL                         READY   REASON
example.com         https://example.com         True
```

2. 선택 사항: 서비스가 공개적으로 노출되는 경우 다음 명령을 실행하여 서비스를 사용할 수 있는지 확인합니다.

```
$ curl https://<domain_name>
```

인증서가 자체 서명된 경우 curl 명령에 -k 플래그를 추가하여 확인을 건너뜁니다.

4.8. KNATIVE 서비스의 고가용성 구성

4.8.1. Knative 서비스의 고가용성

고가용성(HA)은 중단이 발생하는 경우 API가 작동하도록 하는 데 도움이 되는 Kubernetes API의 표준 기능입니다. HA 배포에서 활성 컨트롤러가 충돌하거나 삭제되면 다른 컨트롤러를 쉽게 사용할 수 있습니다. 이 컨트롤러는 현재 사용할 수 없는 컨트롤러에서 서비스하고 있는 API를 대신 처리합니다.

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

4.8.2. Knative 서비스의 고가용성

HA(고가용성)는 기본적으로 **Knative Serving activator, autoscaler ,autoscaler -hpa,controller,webhook,kourier-control, kourier-gateway** 구성 요소에 기본적으로 사용할 수 있습니다. 이 구성 요소는 기본적으로 각각 두 개의 복제본을 보유하도록 구성되어 있습니다. **KnativeServing CR**(사용자 정의 리소스)의 **spec.high-availability.replicas** 값을 수정하여 이러한 구성 요소의 복제본 수를 변경할 수 있습니다.

4.8.2.1. Knative Serving의 고가용성 복제본 구성

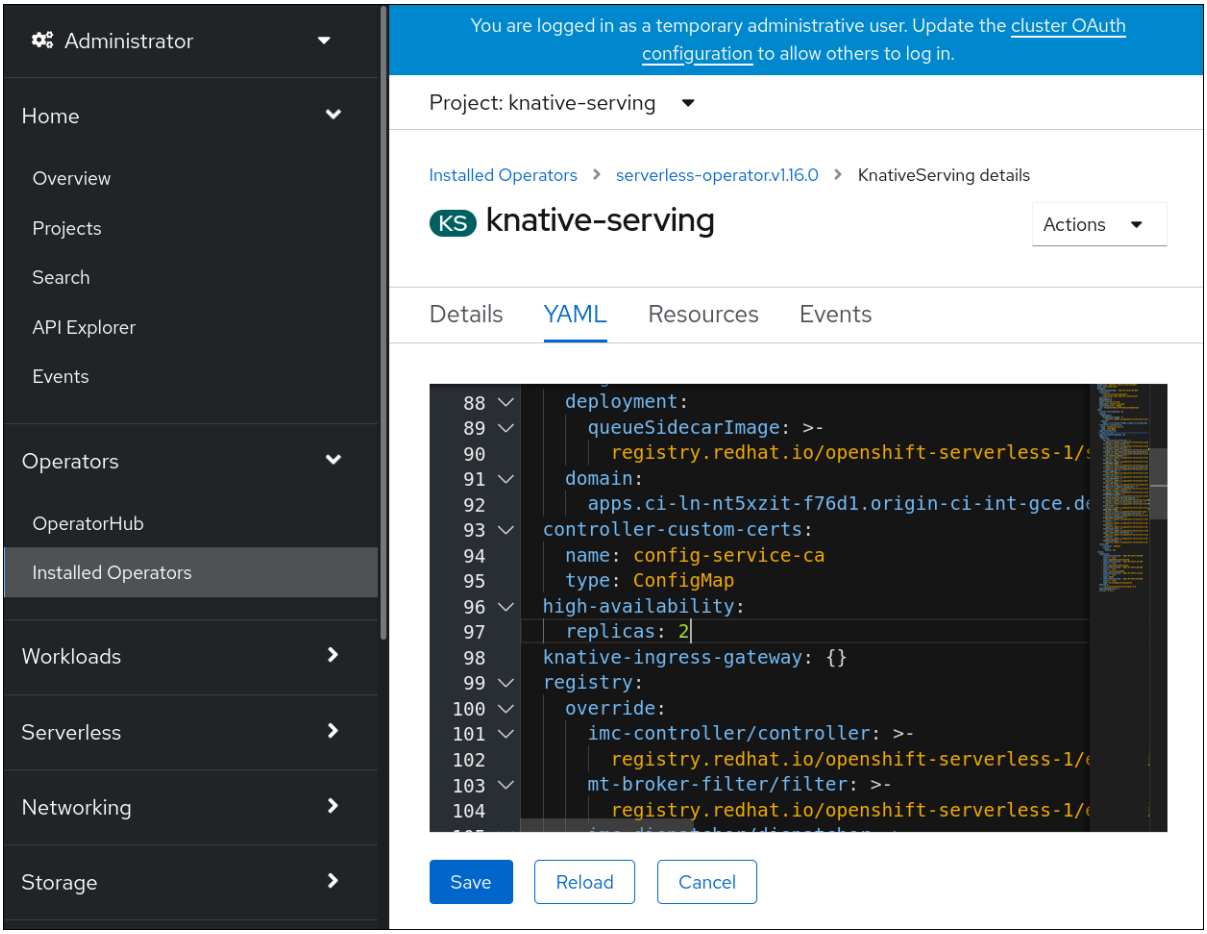
적격 배포 리소스에 대한 최소 **3**개의 복제본을 지정하려면 사용자 정의 리소스의 **spec.high-availability.replicas** 필드 값을 **3** 으로 설정합니다.

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 **OpenShift Container Platform** 계정에 액세스할 수 있습니다.
- **OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다.

절차

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



5. **KnativeServing CR의 복제본 수를 수정합니다.**

YAML의 예

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

5장. EVENTING

5.1. KNATIVE EVENTING

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

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

Knative Eventing에서는 다음 유스 케이스를 지원합니다.

소비자를 생성하지 않고 이벤트 게시

이벤트를 **HTTP POST**에 브로커로 보내고 바인딩을 사용하여 이벤트를 생성하는 애플리케이션에서 대상 구성을 분리할 수 있습니다.

게시자를 생성하지 않고 이벤트 사용

트리거를 사용하면 이벤트 특성을 기반으로 브로커의 이벤트를 사용할 수 있습니다. 애플리케이션은 이벤트를 **HTTP POST**로 수신합니다.

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

주소 지정 가능 리소스

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

호출 가능한 리소스

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

5.2. 이벤트 소스

5.2.1. 이벤트 소스

Knative *이벤트* 소스는 클라우드 이벤트를 생성하거나 가져오는 Kubernetes 오브젝트일 수 있으며 해당 이벤트를 *싱크* 라고 하는 다른 끝점으로 중계할 수 있습니다. 이벤트에 대응하는 분산 시스템을 개발하는 데 소싱 이벤트가 중요합니다.

OpenShift Container Platform 웹 콘솔의 개발자 화면, Knative(kn) CLI 또는 YAML 파일을 적용하여 Knative 이벤트 소스를 생성 및 관리할 수 있습니다.

현재 OpenShift Serverless에서는 다음 이벤트 소스 유형을 지원합니다.

API 서버 소스

Kubernetes API 서버 이벤트를 Knative로 가져옵니다. API 서버 소스는 Kubernetes 리소스를 생성, 업데이트 또는 삭제할 때마다 새 이벤트를 보냅니다.

ping 소스

지정된 cron 일정에 고정된 페이로드를 사용하여 이벤트를 생성합니다.

Kafka 이벤트 소스

Kafka 클러스터를 이벤트 소스로 싱크에 연결합니다.

사용자 지정 이벤트 소스를 생성할 수도 있습니다.

5.2.2. 관리자 관점에서 이벤트 소스

이벤트에 대응하는 분산 시스템을 개발하는 데 소싱 이벤트가 중요합니다.

5.2.2.1. 관리자 화면을 사용하여 이벤트 소스 생성

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

사전 요구 사항

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

- 웹 콘솔에 로그인한 후 관리자 화면에 있습니다.
- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있습니다.

절차

1. OpenShift Container Platform 웹 콘솔의 관리자 화면에서 Serverless → Eventing으로 이동합니다.
2. 생성 목록에서 이벤트 소스를 선택합니다. 그러면 이벤트 소스 페이지로 이동합니다.
3. 생성할 이벤트 소스 유형을 선택합니다.

5.2.3. API 서버 소스 생성

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

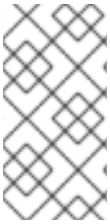
5.2.3.1. 웹 콘솔을 사용하여 API 서버 소스 생성

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

사전 요구 사항

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

- OpenShift CLI(oc)가 설치되어 있습니다.



절차

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

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

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

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

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

1 2 3 4

이 네임스페이스를 이벤트 소스 설치를 위해 선택한 네임스페이스로 변경합니다.

2. YAML 파일을 적용합니다.

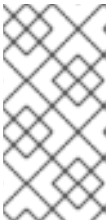
```
$ oc apply -f <filename>
```

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

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

5. ApiServerSource를 선택한 다음 이벤트 소스 생성을 클릭합니다. 이벤트 소스 생성 페이지가 표시됩니다.

6. 양식 보기 또는 YAML 보기를 사용하여 ApiServerSource 설정을 구성합니다.



참고

양식 보기와 YAML 보기를 전환할 수 있습니다. 다른 보기로 전환해도 데이터는 유지됩니다.

a. APIVERSION으로 v1을, KIND로 Event를 입력합니다.

b. 생성한 서비스 계정의 서비스 계정 이름을 선택합니다.

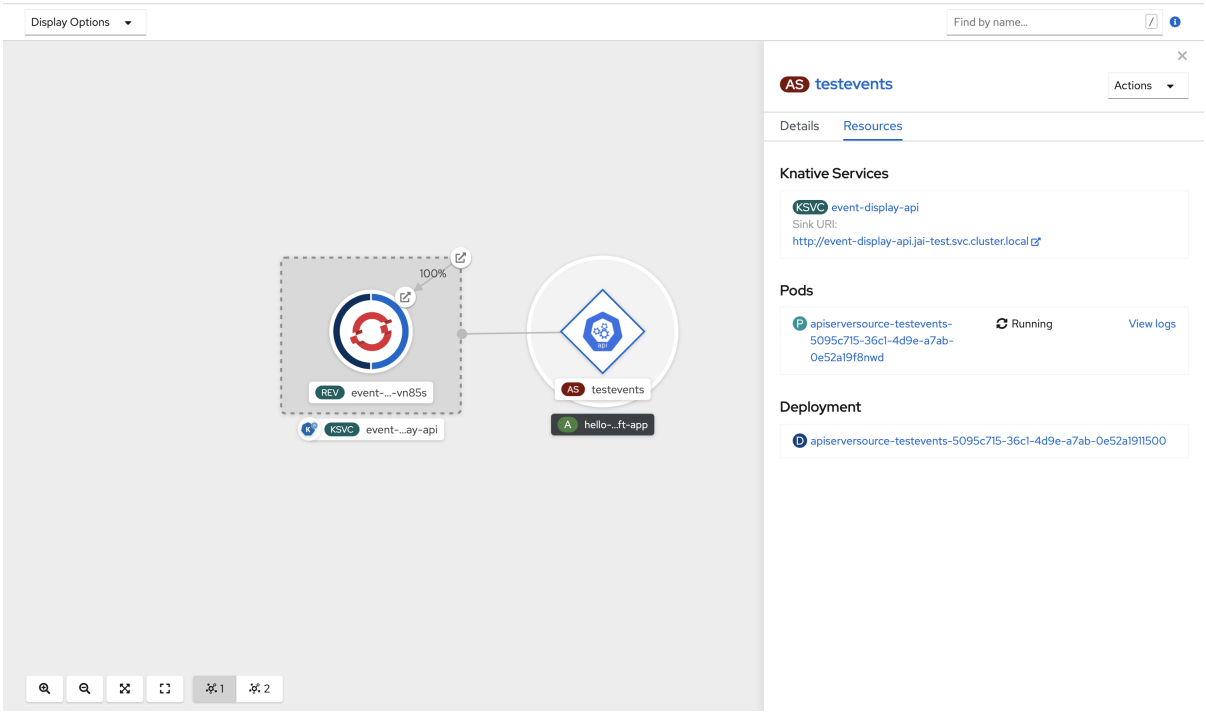
c. 이벤트 소스로 싱크를 선택합니다. 싱크는 채널, 브로커 또는 서비스와 같은 리소스이거나 URI일 수 있습니다.

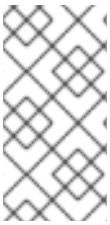
7. 생성을 클릭합니다.

검증

-

API 서버 소스를 생성한 후에는 토폴로지 보기에서 싱크된 서비스에 연결된 것을 확인할 수 있습니다.





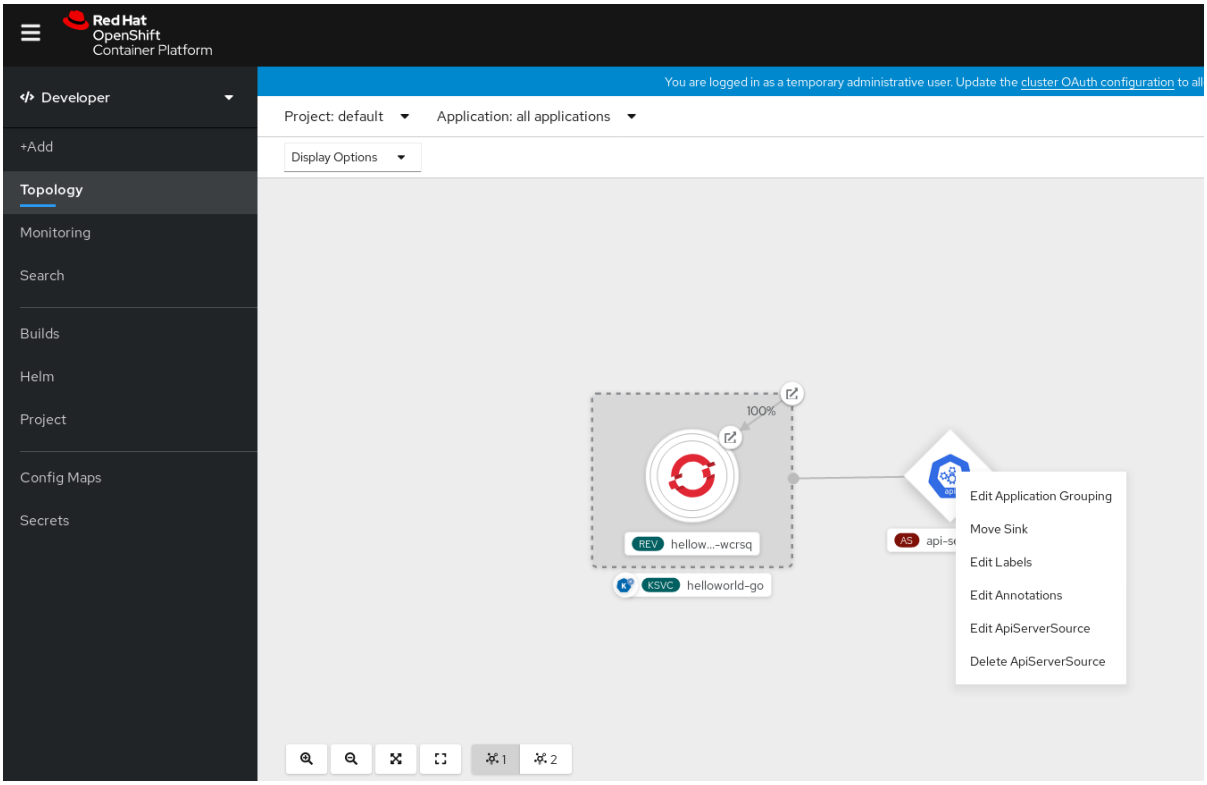
참고

URI 싱크를 사용하는 경우 **URI** 싱크 → **URI** 편집을 마우스 오른쪽 버튼으로 클릭하여 **URI**를 수정합니다.

API 서버 소스 삭제

1. 토폴로지 보기로 이동합니다.

2. **API** 서버 소스를 마우스 오른쪽 버튼으로 클릭하고 **ApiServerSource** 삭제를 선택합니다.

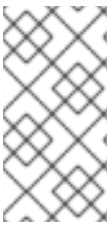


5.2.3.2. Knative CLI를 사용하여 API 서버 소스 생성

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

사전 요구 사항

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



절차

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

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

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

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

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

1 2 3 4 이 네임스페이스를 이벤트 소스 설치를 위해 선택한 네임스페이스로 변경합니다.

2. YAML 파일을 적용합니다.

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

3.

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

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

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

```
$ kn service create <service_name> --image quay.io/openshift-knative/knative-
eventing-sources-event-display:latest
```

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

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

6. 기본 네임스페이스에서 Pod를 시작하여 이벤트를 생성합니다.

```
$ oc create deployment hello-node --image quay.io/openshift-knative/knative-eventing-
sources-event-display:latest
```

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

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

출력 예

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

```
OK TYPE                  AGE REASON
++ Ready                 3m
++ Deployed              3m
++ SinkProvided          3m
++ SufficientPermissions 3m
++ EventTypesProvided    3m
```

검증

메시지 덤퍼 기능 로그를 확인하면 **Kubernetes** 이벤트가 **Knative**로 전송되었는지 확인할 수 있습니다.

1. **Pod**를 가져옵니다.

```
$ oc get pods
```

2. **Pod**의 메시지 덤퍼 기능 로그를 확인합니다.

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

출력 예

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

```
    "name": "hello-node.159d7608e3a3572c",
    "namespace": "default",
    ....
  },
  "reason": "Started",
  ...
}
```

API 서버 소스 삭제

1. 트리거를 삭제합니다.

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

2. 이벤트 소스를 삭제합니다.

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

3. 서비스 계정, 클러스터 역할, 클러스터 바인딩을 삭제합니다.

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

5.2.3.2.1. Knative CLI 싱크 플래그

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

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

싱크 플래그를 사용하는 명령의 예

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
  --subject "Job:batch/v1:app=heartbeat-cron" \
```

```
--sink http://event-display.svc.cluster.local \ 1
--ce-override "sink=bound"
```

1

http://event-display.svc.cluster.local 의 svc 는 싱크가 Knative 서비스인지 확인합니다. 기타 기본 싱크 접두사에는 channel, 및 broker가 포함됩니다.

5.2.3.3. YAML 파일을 사용하여 API 서버 소스 생성

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

사전 요구 사항

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

절차

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

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

```
apiVersion: v1
```

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

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

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

1 2 3 4

이 네임스페이스를 이벤트 소스 설치를 위해 선택한 네임스페이스로 변경합니다.

2. YAML 파일을 적용합니다.

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

3. API 서버 소스를 YAML 파일로 만듭니다.

```
apiVersion: sources.knative.dev/v1alpha1
kind: ApiServerSource
metadata:
```

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

4. ApiServerSource YAML 파일을 적용합니다.

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

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

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

6. Service YAML 파일을 적용합니다.

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

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

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: event-display-trigger
  namespace: default
spec:
  broker: default
  subscriber:
```

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

8. Trigger YAML 파일을 적용합니다.

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

9. 기본 네임스페이스에서 Pod를 시작하여 이벤트를 생성합니다.

```
$ oc create deployment hello-node --image=quay.io/openshift-knative/knative-eventing-sources-event-display
```

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

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

출력 예

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

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

검증

Kubernetes 이벤트가 **Knative**로 전송되었는지 확인하려면 메시지 덤퍼 기능 로그를 확인하면 됩니다.

1. 다음 명령을 입력하여 **pod**를 가져옵니다.

```
$ oc get pods
```

2. 다음 명령을 입력하여 **pod** 메시지 덤퍼 기능 로그를 표시합니다.

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

출력 예

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

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

API 서버 소스 삭제

1. 트리거를 삭제합니다.

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

2. 이벤트 소스를 삭제합니다.

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

3. 서비스 계정, 클러스터 역할, 클러스터 바인딩을 삭제합니다.

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

5.2.4. ping 소스 생성

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

5.2.4.1. 웹 콘솔을 사용하여 ping 소스 생성

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

사전 요구 사항

- **OpenShift Container Platform** 웹 콘솔에 로그인했습니다.
-

OpenShift Serverless Operator, Knative Serving 및 Knative Eventing이 클러스터에 설치되어 있습니다.

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

절차

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

   a. 개발자 화면에서 +추가 → YAML로 이동합니다.

   b. 예제 YAML을 복사합니다.

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

   c. 생성을 클릭합니다.

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

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

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

   c. Ping 소스를 선택한 다음 이벤트 소스 생성을 클릭합니다. 이벤트 소스 생성 페이지가

표시됩니다.

참고

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

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

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

f. 싱크를 선택합니다. 리소스 또는 **URI**일 수 있습니다. 이 예제에서는 이전 단계에서 생성한 **event-display** 서비스를 리소스 싱크로 사용합니다.

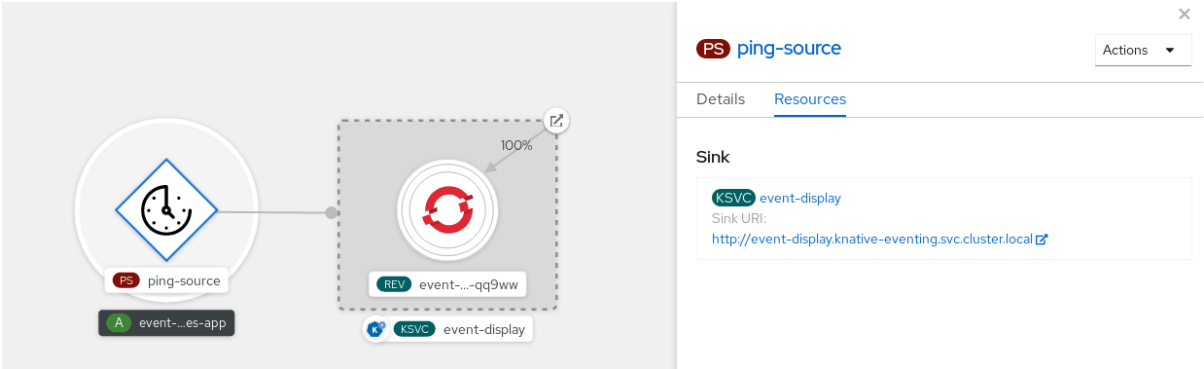
g. 생성을 클릭합니다.

검증

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

1. 개발자 화면에서 토폴로지로 이동합니다.

2. **ping** 소스 및 싱크를 확인합니다.



ping 소스 삭제

1. 토폴로지 보기로 이동합니다.

2. API 서버 소스를 마우스 오른쪽 버튼으로 클릭하고 Ping 소스 삭제를 선택합니다.

5.2.4.2. Knative CLI를 사용하여 ping 소스 생성

kn 소스 ping create 명령을 사용하여 Knative(kn) CLI를 사용하여 ping 소스를 생성할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Serving 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- Knative(kn) CLI가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 선택 사항: 이 프로세스에 대한 확인 단계를 사용하려면 OpenShift CLI(oc)를 설치합니다.

절차

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

```
$ kn service create event-display \
  --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

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

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

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

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

출력 예

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

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

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

검증

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

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

1. 새 Pod가 생성되었는지 확인합니다.

```
$ watch oc get pods
```

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

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

출력 예

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

ping 소스 삭제

- ping 소스를 삭제합니다.

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

5.2.4.2.1. Knative CLI 싱크 플래그

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

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

싱크 플래그를 사용하는 명령의 예

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

1

http://event-display.svc.cluster.local 의 svc 는 싱크가 Knative 서비스인지 확인합니다. 기타 기본 싱크 접두사에는 channel, 및 broker가 포함됩니다.

5.2.4.3. YAML을 사용하여 ping 소스 생성

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

PingSource 개체의 예

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

1

CRON 표현식을 사용하여 지정한 이벤트 스케줄입니다.

2

JSON으로 인코딩된 데이터 문자열로 표시된 이벤트 메시지 본문입니다.

3

이는 이벤트 소비자에 대한 세부 정보입니다. 이 예제에서는 event-display라는 Knative 서비스를 사용하고 있습니다.

사전 요구 사항

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

절차

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

   a. 서비스 YAML 파일을 생성합니다.

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

   b. 서비스를 생성합니다.

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

2.

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

a. **ping** 소스에 대한 **YAML** 파일을 생성합니다.

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

b. **ping** 소스를 생성합니다.

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

3. 다음 명령을 입력하여 컨트롤러가 올바르게 매핑되는지 확인합니다.

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

출력 예

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

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

검증

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

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

1. 새 Pod가 생성되었는지 확인합니다.

```
$ watch oc get pods
```

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

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

출력 예

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

ping 소스 삭제

- ping 소스를 삭제합니다.

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

명령 예

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

5.2.5. Kafka 소스

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

5.2.5.1. 웹 콘솔을 사용하여 Kafka 이벤트 소스 생성

Knative Kafka가 클러스터에 설치되면 웹 콘솔을 사용하여 Kafka 소스를 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔을 사용하면 간단하고 직관적인 사용자 인터페이스를 사용하여 Kafka 소스를 생성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing, KnativeKafka 사용자 정의 리소스가 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인했습니다.
- 가져오려는 Kafka 메시지를 생성하는 Red Hat AMQ Streams(Kafka) 클러스터에 액세스할 수 있습니다.

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

절차

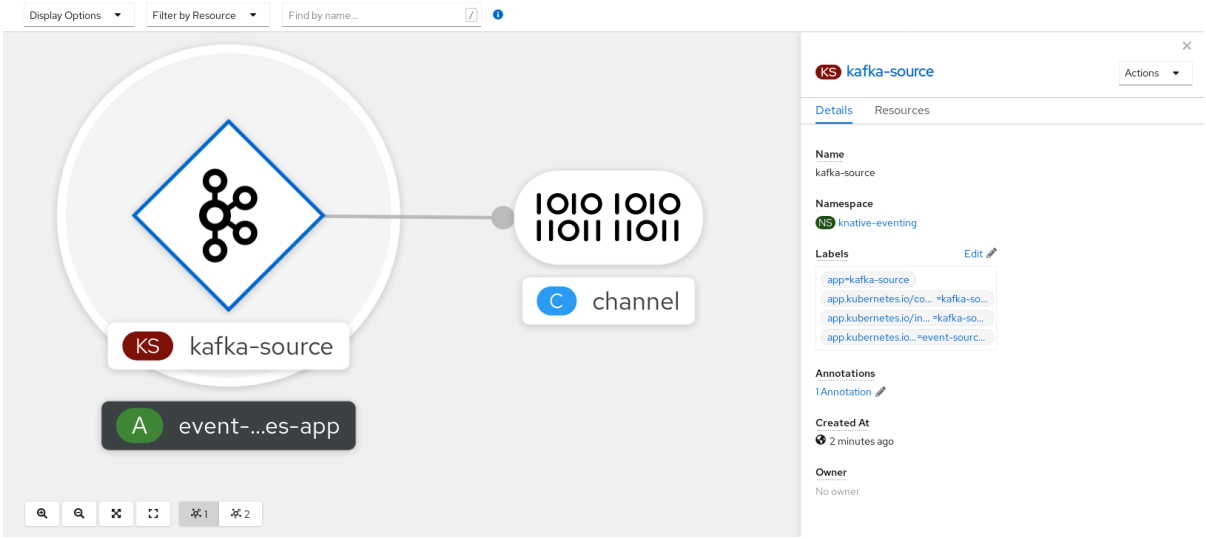
1. 개발자 화면에서 **+추가** 페이지로 이동하여 이벤트 소스를 선택합니다.
2. 이벤트 소스 페이지의 유형 섹션에서 **Kafka** 소스를 선택합니다.
3. **Kafka** 소스 설정을 구성합니다.
   a. 쉼표로 구분된 부트스트랩 서버 목록을 추가합니다.
   b. 쉼표로 구분된 주제 목록을 추가합니다.
   c. 소비자 그룹을 추가합니다.
   d. 생성한 서비스 계정의 서비스 계정 이름을 선택합니다.
   e. 이벤트 소스로 싱크를 선택합니다. 싱크는 채널, 브로커 또는 서비스와 같은 리소스이거나 **URI**일 수 있습니다.
   f. **Kafka** 이벤트 소스로 이름을 입력합니다.
4. 생성을 클릭합니다.

검증

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

1. 개발자 화면에서 토폴로지로 이동합니다.

2. Kafka 이벤트 소스 및 싱크를 확인합니다.



5.2.5.2. Knative CLI를 사용하여 Kafka 이벤트 소스 생성

kn source kafka create 명령을 사용하여 Knative(kn) CLI를 사용하여 Kafka 소스를 생성할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다.

사전 요구 사항

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

절차

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

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

2. KafkaSource CR을 생성합니다.

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

참고

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

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

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

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

출력 예

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

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

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

검증 단계

1. Kafka 인스턴스를 트리거하여 메시지를 항목에 보냅니다.

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

프롬프트에 메시지를 입력합니다. 이 명령은 다음을 가정합니다.

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

2. 로그를 보고 메시지가 도착했는지 확인합니다.

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

출력 예

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

```
  traceparent: 00-161ff3815727d8755848ec01c866d1cd-7ff3916c44334678-00
Data,
  Hello!
```

5.2.5.2.1. Knative CLI 싱크 플래그

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

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

싱크 플래그를 사용하는 명령의 예

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

1

http://event-display.svc.cluster.local 의 **svc** 는 싱크가 **Knative** 서비스인지 확인합니다. 기타 기본 싱크 접두사에는 **channel**, 및 **broker**가 포함됩니다.

5.2.5.3. YAML을 사용하여 Kafka 이벤트 소스 생성

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

사전 요구 사항

-

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

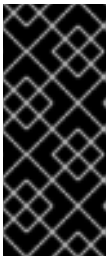
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 가져오려는 Kafka 메시지를 생성하는 Red Hat AMQ Streams(Kafka) 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(oc)를 설치합니다.

절차

1. KafkaSource 오브젝트를 YAML 파일로 생성합니다.

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

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



중요

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

KafkaSource 오브젝트의 예

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

2. KafkaSource YAML 파일을 적용합니다.

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

검증

- 다음 명령을 입력하여 Kafka 이벤트 소스가 생성되었는지 확인합니다.

```
$ oc get pods
```

출력 예

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

5.2.5.4. Kafka 소스에 대한 SASL 인증 구성

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

사전 요구 사항

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

절차

1. **선택한 네임스페이스에서 인증서 파일을 시크릿으로 생성합니다.**

```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
  --from-file=ca.crt=caroot.pem \
  --from-literal=password="SecretPassword" \
```

```
--from-literal=saslType="SCRAM-SHA-512" \ 1
--from-literal=user="my-sasl-user"
```

1

SASL 유형은 PLAIN,SCRAM-SHA-256 또는 SCRAM-SHA-512 일 수 있습니다.

2. 다음 사양 구성이 포함되도록 Kafka 소스를 생성하거나 수정합니다.

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

1

Apache Kafka용 Red Hat OpenShift Streams와 같은 퍼블릭 클라우드 Kafka 서비스를 사용하는 경우에는 caCert 사양이 필요하지 않습니다.

5.2.6. 사용자 정의 이벤트 소스

Knative에 포함되지 않은 이벤트 생산자 또는 CloudEvent 형식에 없는 이벤트를 내보내는 생산자에서 이벤트를 수신해야 하는 경우 사용자 정의 이벤트 소스를 생성하여 이 작업을 수행할 수 있습니다. 다

음 방법 중 하나를 사용하여 사용자 지정 이벤트 소스를 생성할 수 있습니다.

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

5.2.6.1. 싱크 바인딩

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

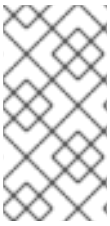
SinkBinding 오브젝트는 싱크의 **PodTemplateSpec**에 환경 변수를 삽입합니다. 즉, 애플리케이션 코드가 이벤트 대상을 찾기 위해 **Kubernetes API**와 직접 상호 작용할 필요가 없습니다. 이러한 환경 변수는 다음과 같습니다.

K_SINK

확인된 싱크의 **URL**입니다.

K_CE_OVERRIDES

아웃바운드 이벤트에 대한 덮어쓰기를 지정하는 **JSON** 오브젝트입니다.



참고

현재 **SinkBinding** 오브젝트는 서비스에 대한 사용자 정의 리비전 이름을 지원하지 않습니다.

5.2.6.1.1. YAML을 사용하여 싱크 바인딩 생성

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

사전 요구 사항

-

OpenShift Serverless Operator, Knative Serving 및 Knative Eventing이 클러스터에 설치되어 있습니다.

- OpenShift CLI(oc)를 설치합니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

절차

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

   a. 서비스 YAML 파일을 생성합니다.

   서비스 YAML 파일 예

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

   b. 서비스를 생성합니다.

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

2. 이벤트를 서비스로 보내는 싱크 바인딩 인스턴스를 생성합니다.

a. 싱크 바인딩 YAML 파일을 생성합니다.

서비스 YAML 파일 예

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

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

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

b. 싱크 바인딩을 생성합니다.

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

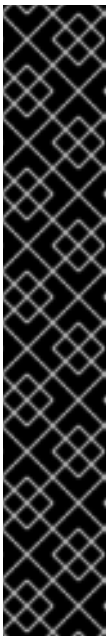
3. CronJob 오브젝트를 생성합니다.

a. cron 작업 YAML 파일을 생성합니다.

cron 작업 YAML 파일의 예

-

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



중요

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

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

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

b. cron 작업을 생성합니다.

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

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

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

출력 예

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

검증

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

1. 명령을 입력합니다.

```
$ oc get pods
```

2. 명령을 입력합니다.

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

출력 예

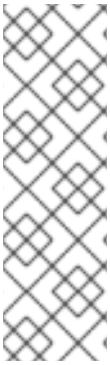
```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.eventing.samples.heartbeat
  source: https://knative.dev/eventing-contrib/cmd/heartbeats/#event-test/mypod
  id: 2b72d7bf-c38f-4a98-a433-608fbcdd2596
  time: 2019-10-18T15:23:20.809775386Z
  contenttype: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

5.2.6.1.2. Knative CLI를 사용하여 싱크 바인딩 생성

kn source binding create 명령을 사용하여 Knative(kn) CLI를 사용하여 싱크 바인딩을 생성할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다.

사전 요구 사항

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



참고

다음 절차에 따라 **YAML** 파일을 생성해야 합니다.

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

절차

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

```
$ kn service create event-display --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

2. 이벤트를 서비스로 보내는 싱크 바인딩 인스턴스를 생성합니다.

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

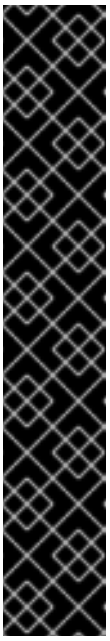
3. **CronJob** 오브젝트를 생성합니다.

   a. **cron** 작업 **YAML** 파일을 생성합니다.

cron 작업 **YAML** 파일의 예

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

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



중요

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

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

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

b. **cron** 작업을 생성합니다.

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

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

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

출력 예

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

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

검증

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

- 다음 명령을 입력하여 메시지 덤퍼 기능 로그를 확인합니다.

```
$ oc get pods
```

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

출력 예

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

```
  heart: yes
  the: 42
Data,
  {
    "id": 1,
    "label": ""
  }
```

5.2.6.1.2.1. Knative CLI 싱크 플래그

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

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

싱크 플래그를 사용하는 명령의 예

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

1

http://event-display.svc.cluster.local 의 **svc** 는 싱크가 **Knative** 서비스인지 확인합니다. 기타 기본 싱크 접두사에는 **channel**, 및 **broker**가 포함됩니다.

5.2.6.1.3. 웹 콘솔을 사용하여 싱크 바인딩 생성

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

사전 요구 사항

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

절차

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

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

   c. 생성을 클릭합니다.
2. 이벤트 소스로 사용되고 1분마다 이벤트를 전송하는 CronJob 리소스를 생성합니다.
   a. 개발자 화면에서 +추가 → YAML로 이동합니다.
   b. 예제 YAML을 복사합니다.

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

1

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

c. 생성을 클릭합니다.

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

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

b. 선택 사항: 이벤트 소스에 대한 공급자가 여러 개인 경우 공급자 목록에서 필요한 공급

자를 선택하여 해당 공급자의 사용 가능한 이벤트 소스를 필터링합니다.

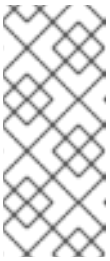
c. 싱크 바인딩을 선택한 다음 이벤트 소스 생성을 클릭합니다. 이벤트 소스 생성 페이지가 표시됩니다.

참고

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

d. apiVersion 필드에 batch/v1을 입력합니다.

e. 유형 필드에 Job을 입력합니다.



참고

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

f. 싱크를 선택합니다. 리소스 또는 URI일 수 있습니다. 이 예제에서는 이전 단계에서 생성한 event-display 서비스를 리소스 싱크로 사용합니다.

g. 레이블 일치 섹션에서 다음을 수행합니다.

i. 이름 필드에 app을 입력합니다.

ii. 값 필드에 heartbeat-cron을 입력합니다.

참고

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

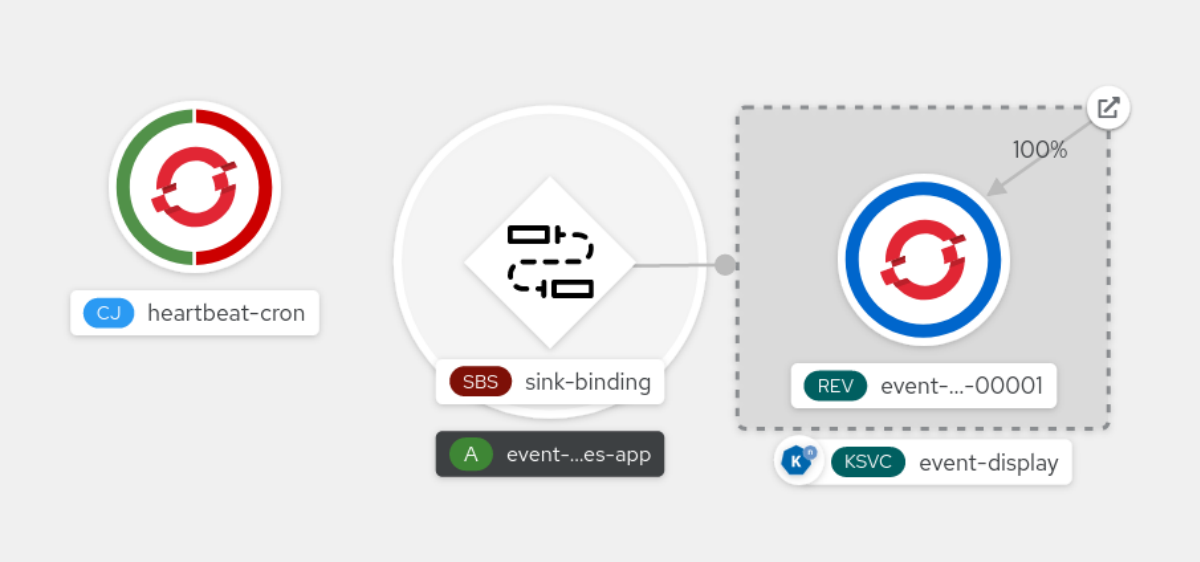
h. 생성을 클릭합니다.

검증

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

1. 개발자 화면에서 토폴로지로 이동합니다.

2. 싱크 바인딩, 싱크 및 하트비트 **cron** 작업을 확인합니다.



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

4. **event-display** 서비스 **pod**의 로그를 검색하여 **heartbeats cron** 작업에서 생성한 이벤트를 확인합니다.

5.2.6.1.4. 싱크 바인딩 참조

싱크 바인딩을 생성하여 **PodSpecable** 오브젝트를 이벤트 소스로 사용할 수 있습니다. SinkBinding 개체를 만들 때 여러 매개 변수를 구성할 수 있습니다.You can configure multiple parameters when creating a SinkBinding object.

SinkBinding 오브젝트는 다음 매개 변수를 지원합니다.

필드	설명	필수 또는 선택적
apiVersion	API 버전을 지정합니다(예: **sources.knative.dev/v1**).	필수 항목
kind	이 리소스 개체를 **SinkBinding** 오브젝트로 식별합니다.	필수 항목
metadata	**SinkBinding** 오브젝트를 고유하게 식별하는 메타데이터를 지정합니다. 예를 들어 **이름**.	필수 항목
spec	이 **SinkBinding** 오브젝트에 대한 구성 정보를 지정합니다.	필수 항목
spec.sink	싱크로 사용하기 위해 URI로 확인되는 오브젝트에 대한 참조입니다.	필수 항목
spec.subject	런타임 계약이 바인딩 구현으로 보강되는 리소스를 참조합니다.	필수 항목
spec.ceOverrides	싱크로 전송된 이벤트에 대한 출력 형식 및 수정을 제어하기 위해 재정의를 정의합니다.	선택 사항

5.2.6.1.4.1. subject 매개변수

Subject 매개변수는 런타임 계약이 바인딩 구현을 통해 보강되는 리소스를 참조합니다. 주체 정의에 대해 여러 필드를 구성할 수 있습니다.

주체 정의는 다음 필드를 지원합니다.

필드	설명	필수 또는 선택적
apiVersion	참조의 API 버전입니다.	필수 항목

필드	설명	필수 또는 선택적
kind	일종의 참조입니다.	필수 항목
namespace	참조의 네임스페이스입니다. 생략하는 경우 기본값은 오브젝트의 네임스페이스입니다.	선택 사항
name	참조의 이름입니다.	**선택기** 를 구성하는 경우 사용하지 마십시오.
선택기	참조의 선택기입니다.	**이름을** 구성하는 경우에는 사용하지 마십시오.
selector.matchExpressions	라벨 선택기 요구 사항 목록입니다.	**matchExpressions** 또는 **matchLabels** 중 하나만 사용하십시오.
selector.matchExpressions.key	선택기가 적용되는 레이블 키입니다.	**matchExpressions** 를 사용하는 경우 필수 항목입니다.
selector.matchExpressions.operator	값 집합에 대한 키의 관계를 나타냅니다.Represents a key's relationship to a set of values. 유효한 연산자는 **in**,**NotIn**,**Exists** 및 **DoesNotExist** 입니다.	**matchExpressions** 를 사용하는 경우 필수 항목입니다.
selector.matchExpressions.values	문자열 값의 배열입니다. **operator** 매개변수 값이 **In** 또는 **NotIn** 인 경우 values 배열은 비어 있지 않아야 합니다. **operator** 매개변수 값이 **Exists** 또는 **DoesNotExist** 인 경우 값 배열은 비어 있어야 합니다. 이 배열은 전략적 병합 패치에서 교체됩니다.	**matchExpressions** 를 사용하는 경우 필수 항목입니다.
selector.matchLabels	키-값 쌍의 맵입니다. **matchLabels** 맵의 각 키-값 쌍은 **matchExpressions** 의 요소와 동일합니다. 여기서 키 필드는 **matchLabels.** <key> , **operator** 는 **In** 이며, **값** 배열에는 **matchLabels.<value** >만 포함됩니다.	**matchExpressions** 또는 **matchLabels** 중 하나만 사용하십시오.

subject 매개변수 예

다음 YAML을 고려할 때 default 네임스페이스에 mysubject 라는 Deployment 오브젝트가 선택됩니다.

-

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

다음 YAML을 지정하면 default 네임스페이스에서 working=example 레이블이 있는 Job 오브젝트가 모두 선택됩니다.

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

다음 YAML을 지정하면 default 네임스페이스의 working=example 또는 working=sample 라벨이 있는 모든 Pod 오브젝트가 선택됩니다.

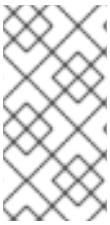
```
apiVersion: sources.knative.dev/v1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  subject:
    apiVersion: v1
    kind: Pod
    namespace: default
    selector:
      - matchExpression:
        key: working
        operator: In
        values:
          - example
          - sample
 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

5.2.6.1.4.2. CloudEvent 덮어쓰기

ceOverrides 정의는 싱크로 전송된 **CloudEvent**의 출력 형식 및 수정 사항을 제어하는 덮어쓰기를 제공합니다. **ceOverrides** 정의에 대해 여러 필드를 구성할 수 있습니다.

ceOverrides 정의는 다음 필드를 지원합니다.

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



참고

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

CloudEvent 덮어쓰기 예

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

제목에 대해 **K_CE_OVERRIDES** 환경 변수를 설정합니다.

출력 예

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

5.2.6.1.4.3. include 라벨

싱크 바인딩을 사용하려면 **resources** 또는 리소스가 포함된 네임스페이스에 **bindings.knative.dev/include: "true"** 레이블을 할당해야 합니다. 리소스 정의에 라벨이 포함되지 않은 경우 클러스터 관리자는 다음을 실행하여 네임스페이스에 연결할 수 있습니다.

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

5.2.6.2. 컨테이너 소스

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

5.2.6.2.1. 컨테이너 이미지 생성을 위한 지침

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

컨테이너 이미지의 예

다음은 하트비트 컨테이너 이미지의 예입니다.

```
package main

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

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

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

```
)

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

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

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

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

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

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

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

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

func main() {
 flag.Parse()

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

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

 var ceOverrides *duckv1.CloudEventOverrides
 if len(env.CEOverrides) > 0 {
```

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

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

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

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

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

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

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

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

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

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

  if env.OneShot {
   return
  }

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

다음은 이전 하트비트 컨테이너 이미지를 참조하는 컨테이너 소스의 예입니다.

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

5.2.6.2.2. Knative CLI를 사용하여 컨테이너 소스 생성 및 관리

kn 소스 컨테이너 명령을 사용하여 Knative(kn) CLI를 사용하여 컨테이너 소스를 생성하고 관리할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다.

컨테이너 소스 생성

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

컨테이너 소스 삭제

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

컨테이너 소스 설명

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

기존 컨테이너 소스 나열

```
$ kn source container list
```

YAML 형식으로 기존 컨테이너 소스 나열

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

컨테이너 소스 업데이트

이 명령은 기존 컨테이너 소스의 이미지 URI를 업데이트합니다.

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

5.2.6.2.3. 웹 콘솔을 사용하여 컨테이너 소스 생성

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

사전 요구 사항

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

절차

1. 개발자 화면에서 +추가 → 이벤트 소스로 이동합니다. 이벤트 소스 페이지가 표시됩니다.
2. 컨테이너 소스를 선택한 다음 이벤트 소스 생성을 클릭합니다. 이벤트 소스 생성 페이지가 표시됩니다.
3. 양식 보기 또는 YAML 보기를 사용하여 컨테이너 소스 설정을 구성합니다.

참고

양식 보기와 YAML 보기를 전환할 수 있습니다. 다른 보기로 전환해도 데이터는 유지됩니다.

   a. 이미지 필드에 컨테이너 소스에서 생성한 컨테이너에서 실행할 이미지의 URI를 입력

합니다.

b. 이름 필드에 이미지 이름을 입력합니다.

c. 선택 사항: 인수 필드에 컨테이너에 전달할 인수를 입력합니다.

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

e. 싱크 섹션에서 컨테이너 소스의 이벤트가 라우팅되는 싱크를 추가합니다. 양식 보기를 사용하는 경우 다음 옵션 중에서 선택할 수 있습니다.

i. 리소스에서 채널, 브로커 또는 서비스를 이벤트 소스의 싱크로 사용합니다.

ii. URI를 선택하여 컨테이너 소스의 이벤트가 라우팅되는 위치를 지정합니다.

4. 컨테이너 소스 구성을 완료한 후 생성을 클릭합니다.

5.2.6.2.4. 컨테이너 소스 참조

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

ContainerSource 개체는 다음 필드를 지원합니다.

필드	설명	필수 또는 선택적
apiVersion	API 버전을 지정합니다(예: **sources.knative.dev/v1**).	필수 항목
kind	이 리소스 개체를 **ContainerSource** 개체로 식별합니다.	필수 항목

필드	설명	필수 또는 선택적
metadata	**ContainerSource** 개체를 고유하게 식별하는 메타데이터를 지정합니다. 예를 들어 **이름**.	필수 항목
spec	이 **ContainerSource** 개체의 구성 정보를 지정합니다.	필수 항목
spec.sink	싱크로 사용하기 위해 URI로 확인되는 오브젝트에 대한 참조입니다.	필수 항목
spec.template	**ContainerSource** 오브젝트의 **템플릿** 사양입니다.	필수 항목
spec.ceOverrides	싱크로 전송된 이벤트에 대한 출력 형식 및 수정을 제어하기 위해 재정의를 정의합니다.	선택 사항

템플릿 매개변수 예

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

5.2.6.2.4.1. CloudEvent 덮어쓰기

ceOverrides 정의는 싱크로 전송된 **CloudEvent**의 출력 형식 및 수정 사항을 제어하는 덮어쓰기를 제공합니다. **ceOverrides** 정의에 대해 여러 필드를 구성할 수 있습니다.

ceOverrides 정의는 다음 필드를 지원합니다.

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

참고

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

CloudEvent 덮어쓰기 예

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

제목에 대해 **K_CE_OVERRIDES** 환경 변수를 설정합니다.

출력 예

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

5.2.7. 개발자 화면을 사용하여 이벤트 소스를 싱크에 연결

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

5.2.7.1. 개발자 화면을 사용하여 이벤트 소스를 싱크에 연결

사전 요구 사항

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

절차

1. +추가 → 이벤트 소스로 이동하여 모든 유형의 이벤트 소스를 생성하고 생성할 이벤트 소스 유형을 선택합니다.
2. 이벤트 소스 생성 양식 보기의 싱크 섹션에서 리소스 목록에서 싱크를 선택합니다.
3. 생성을 클릭합니다.

검증

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

1. 개발자 화면에서 토폴로지로 이동합니다.

2. 이벤트 소스를 표시하고 연결된 싱크를 클릭하여 오른쪽 패널에서 싱크 세부 정보를 확인합니다.

5.3. 이벤트 싱크

5.3.1. 이벤트 싱크

이벤트 소스를 생성할 때 소스에서 이벤트를 보내는 싱크를 지정할 수 있습니다. 싱크는 다른 리소스에서 들어오는 이벤트를 수신할 수 있는 주소 지정 가능 또는 호출 가능한 리소스입니다. **Knative** 서비스, 채널 및 브로커는 모두 싱크의 예입니다.

주소 지정 오브젝트는 **HTTP**를 통해 제공되는 이벤트를 **status.address.url** 필드에 정의된 주소로 수신 및 승인합니다. 특수한 경우 **Kubernetes** 서비스 오브젝트도 주소 지정 가능한 인터페이스입니다.

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

5.3.1.1. Knative CLI 싱크 플래그

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

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

싱크 플래그를 사용하는 명령의 예

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

1

http://event-display.svc.cluster.local 의 **svc** 는 싱크가 **Knative** 서비스인지 확인합니다. 기타 기본 싱크 접두사에는 **channel**, 및 **broker**가 포함됩니다.

작은 정보

kn(kn) CLI 명령에 **--sink** 플래그와 함께 사용할 수 있는 **CR**을 사용자 정의 하여 구성할 수 있습니다.

5.3.2. Kafka 싱크

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

5.3.2.1. Kafka 싱크 사용

Kafka 항목에 이벤트를 전송하는 **Kafka** 싱크라는 이벤트 싱크를 생성할 수 있습니다. **YAML** 파일을 사용하여 **Knative** 리소스를 생성하면 선언적 방식으로 애플리케이션을 재현할 수 있으며 재현 가능한 방식으로 애플리케이션을 설명할 수 있습니다. 기본적으로 **Kafka** 싱크는 구조화된 모드보다 효율적인 바이너리 콘텐츠 모드를 사용합니다. **YAML**을 사용하여 **Kafka** 싱크를 생성하려면 **KafkaSink** 오브젝트를 정의하는 **YAML** 파일을 생성한 다음 **oc apply** 명령을 사용하여 적용해야 합니다.

사전 요구 사항

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

절차

1. KafkaSink 오브젝트 정의를 YAML 파일로 생성합니다.

Kafka 싱크 YAML

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

2. Kafka 싱크를 생성하려면 KafkaSink YAML 파일을 적용합니다.

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

3. 싱크가 사양에 지정되도록 이벤트 소스를 구성합니다.

API 서버 소스에 연결된 Kafka 싱크의 예

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

-

1

이벤트 소스의 이름입니다.

2

이벤트 소스의 네임스페이스입니다.

3

이벤트 소스에 대한 서비스 계정입니다.

4

Kafka 싱크 이름입니다.

5.3.2.2. Kafka 싱크에 대한 보안 구성

TLS(*Transport Layer Security*)는 Apache Kafka 클라이언트와 서버에서 Knative와 Kafka 간의 트래픽 암호화 및 인증에 사용됩니다. TLS는 Knative Kafka에서 지원되는 유일한 트래픽 암호화 방법입니다.

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

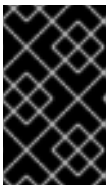
사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing, KnativeKafka 사용자 정의 리소스 (CR)가 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- Kafka 싱크는 KnativeKafka CR에서 활성화되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

- Kafka 클러스터 CA 인증서가 .pem 파일로 저장되어 있습니다.
- Kafka 클러스터 클라이언트 인증서 및 키가 .pem 파일로 저장되어 있습니다.
- OpenShift(oc) CLI를 설치했습니다.
- 사용할 SASL 메커니즘을 선택했습니다(예: PLAIN,SCRAM-SHA-256 또는 SCRAM-SHA-512).

절차

1. KafkaSink 오브젝트와 동일한 네임스페이스에 인증서 파일을 시크릿으로 생성합니다.



중요

인증서 및 키는 PEM 형식이어야 합니다.

- 암호화 없이 SASL을 사용한 인증의 경우:

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

- TLS를 사용한 SASL 및 암호화를 사용한 인증의 경우:

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

1 퍼블릭 클라우드 관리 Kafka 서비스를 사용하는 경우 시스템의 루트 CA 세트(예: Red Hat OpenShift Streams for Apache Kafka)를 사용하도록 ca.crt 를 생략할 수 있습니다.

- TLS를 사용한 인증 및 암호화의 경우:

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

1 퍼블릭 클라우드 관리 Kafka 서비스를 사용하는 경우 시스템의 루트 CA 세트(예: Red Hat OpenShift Streams for Apache Kafka)를 사용하도록 ca.crt 를 생략할 수 있습니다.

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

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

3. KafkaSink 오브젝트를 적용합니다.

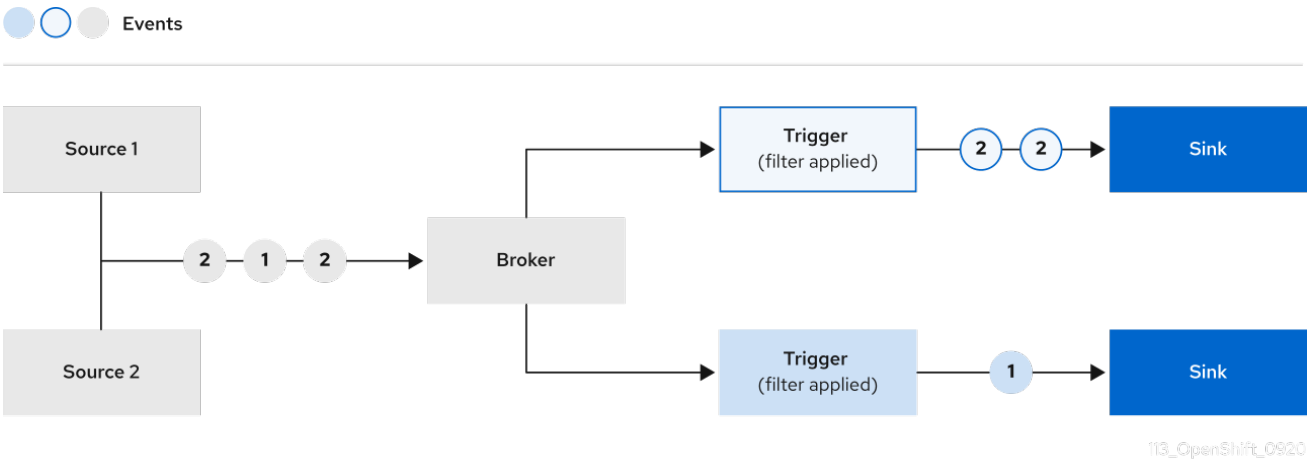
```
$ oc apply -f <filename>
```

5.4. 브로커

5.4.1. 브로커

브로커는 트리거와 함께 이벤트 소스에서 이벤트 싱크로 이벤트를 전달하는 데 사용할 수 있습니다. 이벤트는 HTTP POST 요청으로 이벤트 소스에서 브로커로 전송됩니다. 이벤트가 브로커에 입력되면 트리

거를 사용하여 CloudEvent 속성으로 필터링하고 이벤트 싱크에 HTTP POST 요청으로 보낼 수 있습니다.



5.4.2. 브로커 유형

클러스터 관리자는 클러스터에 대한 기본 브로커 구현을 설정할 수 있습니다. 브로커를 생성할 때 Broker 오브젝트에 세트 구성을 제공하지 않으면 default 브로커 구현이 사용됩니다.

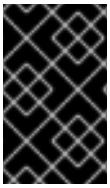
5.4.2.1. 개발을 위한 기본 브로커 구현

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

Kafka를 사용하여 네트워크 홉을 줄이려면 Kafka 브로커 구현을 사용합니다. KafkaChannel 채널 구현에서 지원하도록 채널 기반 브로커를 구성하지 마십시오.

5.4.2.2. 프로덕션 지원 Kafka 브로커 구현

프로덕션 지원 Knative Eventing 배포의 경우 Red Hat은 Knative Kafka 브로커 구현을 사용하는 것이 좋습니다. Kafka 브로커는 Knative 브로커의 Apache Kafka 네이티브 구현이며 Kafka 인스턴스로 CloudEvent를 직접 보냅니다.



중요

Kafka 브로커에 대해 연방 정보 처리 표준(FIPS) 모드가 비활성화되어 있습니다.

Kafka 브로커는 이벤트를 저장하고 라우팅하기 위해 Kafka와 네이티브 통합을 수행합니다. 이를 통

해 브로커에 대한 Kafka와 보다 효율적으로 통합하고 다른 브로커 유형에 대해 트리거 모델을 트리거할 수 있으며 네트워크 홉을 줄일 수 있습니다. Kafka 브로커 구현의 다른 이점은 다음과 같습니다.

- at-least-once 제공 보장
- CloudEvents 파티셔닝 확장에 따른 이벤트 전달
- 컨트롤 플레인 고가용성
- 수평으로 확장 가능한 데이터 플레인

Knative Kafka 브로커는 바이너리 콘텐츠 모드를 사용하여 들어오는 CloudEvents를 Kafka 레코드로 저장합니다. 즉, CloudEvent의 data 사양이 Kafka 레코드의 값에 해당하는 반면 모든 CloudEvent 속성 및 확장이 Kafka 레코드의 헤더로 매핑됩니다.

5.4.3. 브로커 생성

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

클러스터 관리자가 Kafka를 기본 브로커 유형으로 사용하도록 OpenShift Serverless 배포를 구성한 경우 기본 설정을 사용하여 브로커를 생성하여 Kafka 기반 브로커를 생성합니다.

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

5.4.3.1. Knative CLI를 사용하여 브로커 생성

브로커는 트리거와 함께 이벤트 소스에서 이벤트 싱크로 이벤트를 전달하는 데 사용할 수 있습니다. Knative(kn) CLI를 사용하여 브로커를 생성하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다. kn broker create 명령을 사용하여 브로커를 생성할 수 있습니다.

사전 요구 사항

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

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

절차

- 브로커를 생성합니다.

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

검증

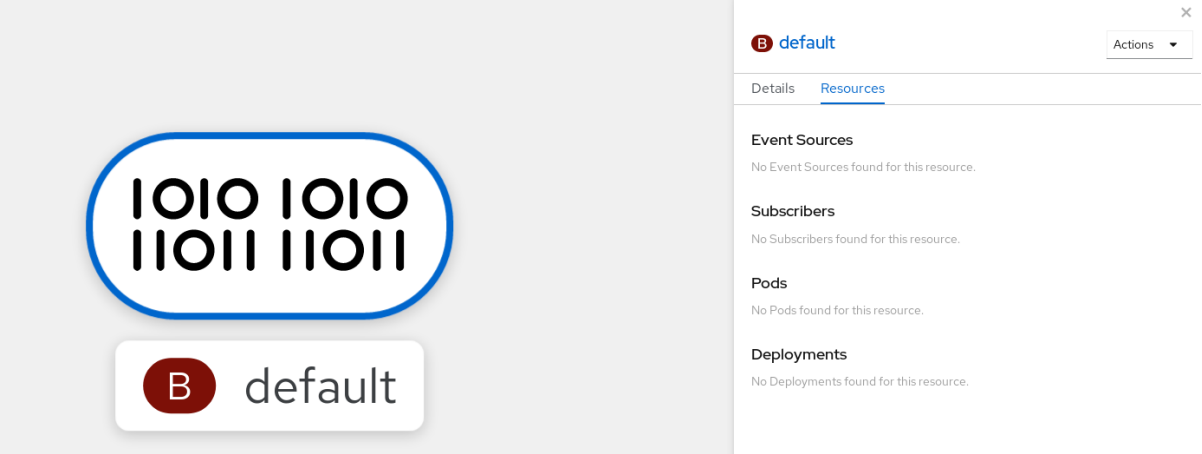
1. kn 명령을 사용하여 기존 브로커를 모두 나열합니다.

```
$ kn broker list
```

출력 예

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

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



5.4.3.2. 트리거에 주석을 달아 브로커 생성

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

중요

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

사전 요구 사항

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

프로세스

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

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
```

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

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

2. Trigger YAML 파일을 적용합니다.

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

검증

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

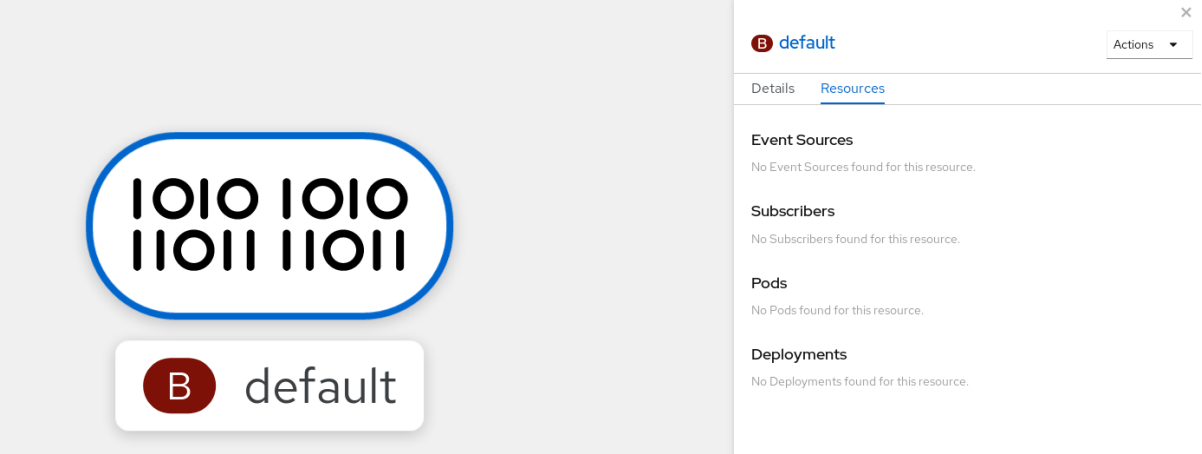
1. oc 명령을 사용하여 브로커를 가져옵니다.

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

출력 예

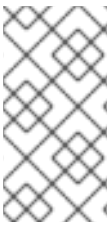
```
NAME    READY    REASON    URL                                                    AGE
default  True             http://broker-ingress.knative-
eventing.svc.cluster.local/test/default  3m56s
```

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



5.4.3.3. 네임스페이스에 라벨을 지정하여 브로커 생성

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



참고

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

사전 요구 사항

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

프로세스

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

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

검증

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

1. oc 명령을 사용하여 브로커를 가져옵니다.

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

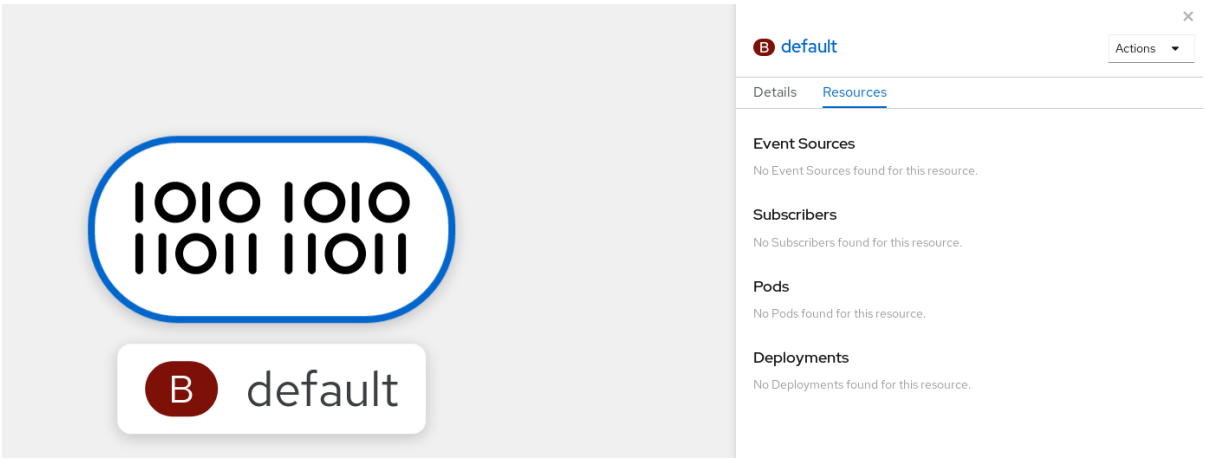
명령 예

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

출력 예

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

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



5.4.3.4. 삽입을 통해 생성된 브로커 삭제

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

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.

절차

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

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

주석을 제거하면 Knative에서 브로커를 삭제한 후 다시 생성하지 않습니다.

2. 선택한 네임스페이스에서 브로커를 삭제합니다.

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

검증

- oc 명령을 사용하여 브로커를 가져옵니다.

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

명령 예

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

출력 예

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

5.4.3.5. 웹 콘솔을 사용하여 브로커 생성

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

사전 요구 사항

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

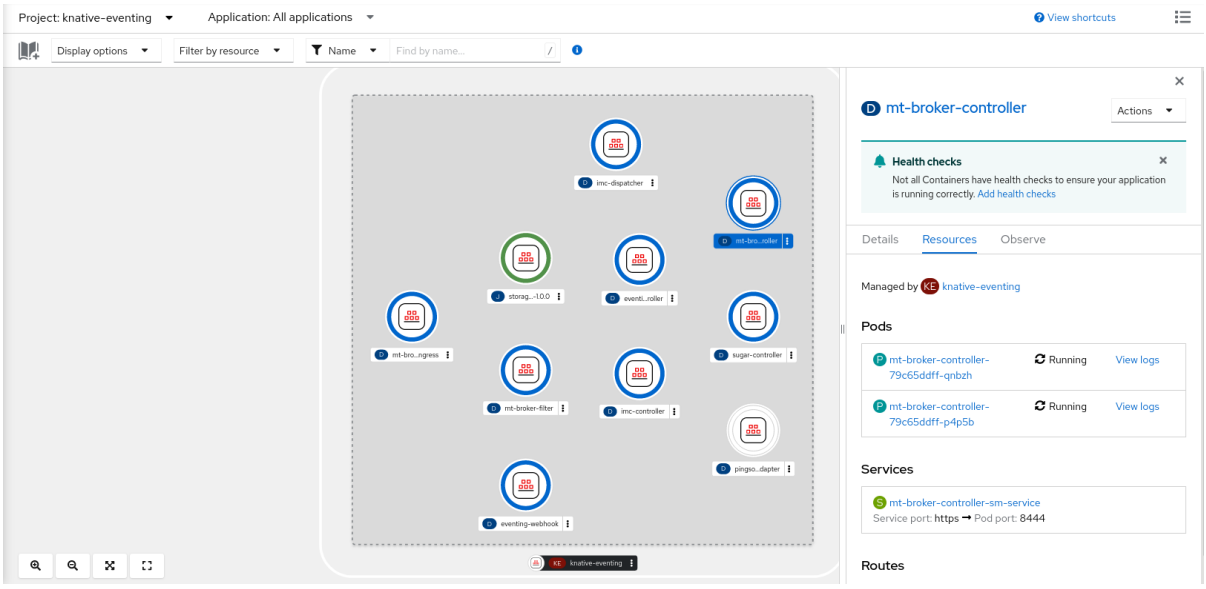
절차

1. 개발자 화면에서 +추가 → 브로커 로 이동합니다. 브로커 페이지가 표시됩니다.
2. 선택 사항: 브로커 이름을 업데이트합니다. 이름을 업데이트하지 않으면 생성된 브로커의 이름은 default 입니다.
3. 생성을 클릭합니다.

검증

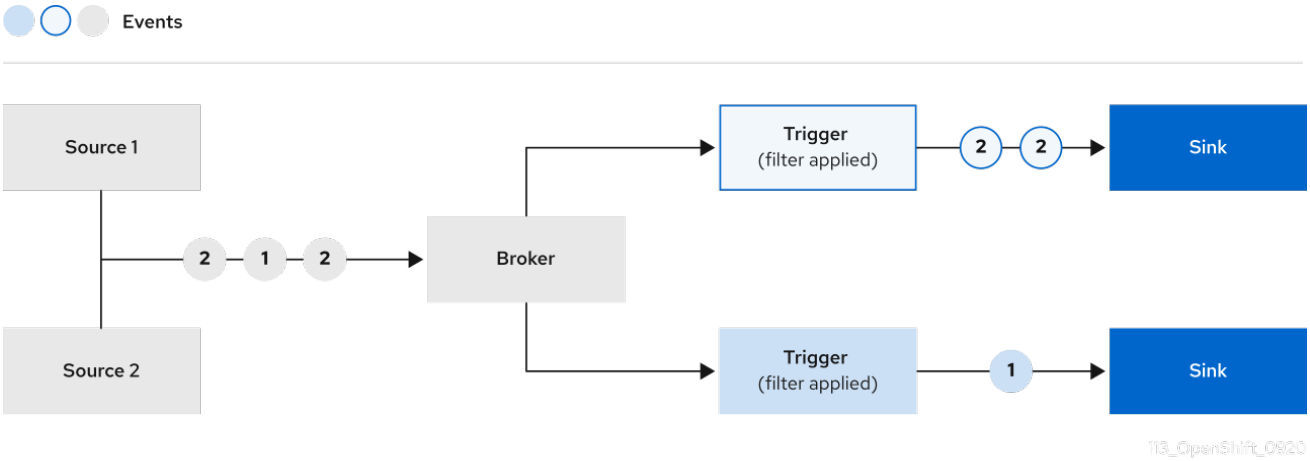
토폴로지 페이지에서 브로커 구성 요소를 확인하여 브로커가 생성되었는지 확인할 수 있습니다.

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



5.4.3.6. 관리자 화면을 사용하여 브로커 생성

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



사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Eventing**이 **OpenShift Container Platform** 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 관리자 화면에 있습니다.
- **OpenShift Container Platform**에 대한 클러스터 관리자 권한이 있습니다.

절차

1. OpenShift Container Platform 웹 콘솔의 관리자 화면에서 Serverless → Eventing으로 이동합니다.

2. 생성 목록에서 브로커를 선택합니다. 그러면 브로커 생성 페이지로 이동합니다.

3. 선택 사항: 브로커의 YAML 구성을 수정합니다.

4. 생성을 클릭합니다.

5.4.3.7. 다음 단계

- 이벤트가 이벤트 싱크로 전달되지 않는 경우 적용되는 이벤트 전달 매개변수를 구성합니다. 이벤트 전달 매개변수 구성 예를 참조하십시오.

5.4.3.8. 추가 리소스

- 기본 브로커 클래스 구성
- 트리거이벤트 소스
- 이벤트 전달

5.4.4. 기본 브로커 지원 채널 구성

채널 기반 브로커를 사용하는 경우 브로커의 기본 지원 채널 유형을 InMemoryChannel 또는 KafkaChannel로 설정할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform에 대한 관리자 권한이 있습니다.

- OpenShift Serverless Operator 및 Knative Eventing을 클러스터에 설치했습니다.
- OpenShift(oc) CLI를 설치했습니다.
- Kafka 채널을 기본 지원 채널 유형으로 사용하려면 클러스터에 KnativeKafka CR도 설치해야 합니다.

절차

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

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

(1) spec.config에서 수정된 구성을 추가할 구성 맵을 지정할 수 있습니다.

(2) 기본 지원 채널 유형 구성입니다. 이 예에서 클러스터의 기본 채널 구현은 KafkaChannel 입니다.

(3) 브로커를 지원하는 Kafka 채널의 파티션 수입니다.

(4) 브로커를 지원하는 Kafka 채널의 복제 요소.

2. 업데이트된 KnativeEventing CR을 적용합니다.

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

5.4.5. 기본 브로커 클래스 구성

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

사전 요구 사항

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

절차

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

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

```
      brokerClass: MTChannelBasedBroker
      apiVersion: v1
      kind: ConfigMap
      name: config-br-default-channel (8)
      namespace: knative-eventing (9)
...
```

(1)

Knative Eventing의 기본 브로커 클래스입니다.

(2)

spec.config에서 수정된 구성을 추가할 구성 맵을 지정할 수 있습니다.

(3)

config-br-defaults 구성 맵은 spec.config 설정 또는 브로커 클래스를 지정하지 않는 브로커의 기본 설정을 지정합니다.

(4)

클러스터 전체 기본 브로커 클래스 구성입니다. 이 예에서 클러스터의 기본 브로커 클래스 구현은 Kafka 입니다.

(5)

kafka-broker-config 구성 맵은 Kafka 브로커의 기본 설정을 지정합니다. "추가 리소스" 섹션의 Kafka 브로커 설정 구성을 참조하십시오.

(6)

kafka-broker-config 구성 맵이 존재하는 네임스페이스입니다.

(7)

네임스페이스 범위의 기본 브로커 클래스 구성입니다. 이 예에서 my-namespace 네임스페이스의 기본 브로커 클래스 구현은 MTChannelBasedBroker 입니다. 여러 네임스페이스에 기본 브로커 클래스 구현을 지정할 수 있습니다.

(8)

config-br-default-channel 구성 맵은 브로커의 기본 지원 채널을 지정합니다. "추가 리소스" 섹션의 "기본 브로커 백업 채널 구성"을 참조하십시오.

(9)

config-br-default-channel 구성 맵이 존재하는 네임스페이스입니다.

중요

네임스페이스별 기본값을 구성하면 클러스터 수준 설정을 덮어씁니다.

5.4.6. Kafka 브로커

프로덕션 지원 **Knative Eventing** 배포의 경우 **Red Hat**은 **Knative Kafka** 브로커 구현을 사용하는 것이 좋습니다. **Kafka** 브로커는 **Knative** 브로커의 **Apache Kafka** 네이티브 구현이며 **Kafka** 인스턴스로 **CloudEvent**를 직접 보냅니다.

중요

Kafka 브로커에 대해 연방 정보 처리 표준**(FIPS)** 모드가 비활성화되어 있습니다.

Kafka 브로커는 이벤트를 저장하고 라우팅하기 위해 **Kafka**와 네이티브 통합을 수행합니다. 이를 통해 브로커에 대한 **Kafka**와 보다 효율적으로 통합하고 다른 브로커 유형에 대해 트리거 모델을 트리거할 수 있으며 네트워크 홉을 줄일 수 있습니다. **Kafka** 브로커 구현의 다른 이점은 다음과 같습니다.

- **at-least-once** 제공 보장
- **CloudEvents** 파티셔닝 확장에 따른 이벤트 전달
- 컨트롤 플레인 고가용성
- 수평으로 확장 가능한 데이터 플레인

Knative Kafka 브로커는 바이너리 콘텐츠 모드를 사용하여 들어오는 **CloudEvents**를 **Kafka** 레코드로 저장합니다. 즉, **CloudEvent**의 **data** 사양이 **Kafka** 레코드의 값에 해당하는 반면 모든 **CloudEvent** 속성 및 확장이 **Kafka** 레코드의 헤더로 매핑됩니다.

5.4.6.1. 기본 브로커 유형으로 구성되지 않은 경우 Kafka 브로커 생성

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

5.4.6.1.1. YAML을 사용하여 Kafka 브로커 생성

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

사전 요구 사항

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

절차

1. Kafka 기반 브로커를 YAML 파일로 생성합니다.

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

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

2

Knative Kafka 브로커의 기본 구성 맵입니다. 이 구성 맵은 클러스터 관리자가 Kafka 브로커 기능을 클러스터에서 활성화할 때 생성됩니다.

2. Kafka 기반 브로커 YAML 파일을 적용합니다.

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

5.4.6.1.2. 외부 관리 Kafka 주제를 사용하는 Kafka 브로커 생성

자체 내부 주제를 생성하지 않고 Kafka 브로커를 사용하려면 대신 외부적으로 관리되는 Kafka 주제를 사용할 수 있습니다. 이를 위해서는 kafka.eventing.knative.dev/external.topic 주석을 사용하는 Kafka Broker 오브젝트를 생성해야 합니다.

사전 요구 사항

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

절차

1. Kafka 기반 브로커를 YAML 파일로 생성합니다.

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

1

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

2

사용하려는 **Kafka** 항목의 이름입니다.

2. **Kafka** 기반 브로커 **YAML** 파일을 적용합니다.

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

5.4.6.2. Kafka 브로커 설정 구성

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

사전 요구 사항

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

절차

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

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

```
  namespace: <namespace> 2
data:
  default.topic.partitions: <integer> 3
  default.topic.replication.factor: <integer> 4
  bootstrap.servers: <list_of_servers> 5
```

1

구성 맵 이름입니다.

2

구성 맵이 존재하는 네임스페이스입니다.

3

Kafka 브로커의 주제 파티션 수입니다. 이렇게 하면 이벤트를 브로커로 얼마나 빨리 보낼 수 있는지가 제어됩니다. 파티션 수가 많을 경우 더 많은 컴퓨팅 리소스가 필요합니다.

4

주제 메시지의 복제 요소. 이는 데이터 손실을 방지합니다. 복제 요인이 증가하려면 더 많은 컴퓨팅 리소스와 더 많은 스토리지가 필요합니다.

5

쉼표로 구분된 부트스트랩 서버 목록입니다. 이는 **OpenShift Container Platform** 클러스터 내부 또는 외부에 있을 수 있으며 브로커가 이벤트를 수신하고 이벤트를 보내는 **Kafka** 클러스터 목록입니다.

중요

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

Kafka 브로커 구성 맵의 예

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kafka-broker-config
  namespace: knative-eventing
data:
```

```
  default.topic.partitions: "10"
  default.topic.replication.factor: "3"
  bootstrap.servers: "my-cluster-kafka-bootstrap.kafka:9092"
```

2. 구성 맵을 적용합니다.

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

3. Kafka Broker 오브젝트의 구성 맵을 지정합니다.

Broker 오브젝트의 예

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

1. 브로커 이름입니다.

2. 브로커가 존재하는 네임스페이스입니다.

3.

4

구성 맵 이름입니다.

5

구성 맵이 존재하는 네임스페이스입니다.

4. 브로커를 적용합니다.

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

5.4.6.3. Knative Kafka 브로커의 보안 구성

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

참고

SASL과 TLS를 모두 함께 활성화하는 것이 좋습니다.

5.4.6.3.1. Kafka 브로커의 TLS 인증 구성

TLS(*Transport Layer Security*)는 Apache Kafka 클라이언트와 서버에서 Knative와 Kafka 간의 트래픽 암호화 및 인증에 사용됩니다. TLS는 Knative Kafka에서 지원되는 유일한 트래픽 암호화 방법입니다.

사전 요구 사항

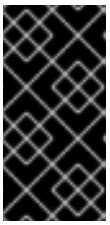
- OpenShift Container Platform에 대한 클러스터 또는 전용 관리자 권한이 있어야 합니다.
- OpenShift Serverless Operator, Knative Eventing, KnativeKafka CR이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

- Kafka 클러스터 CA 인증서가 .pem 파일로 저장되어 있습니다.
- Kafka 클러스터 클라이언트 인증서 및 키가 .pem 파일로 저장되어 있습니다.
- OpenShift CLI(oc)를 설치합니다.

절차

1. knative-eventing 네임스페이스에 인증서 파일을 시크릿으로 생성합니다.

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



중요

키 이름 ca.crt,user.crt, user.key를 사용합니다. 이 값은 변경하지 마십시오.

2. KnativeKafka CR을 편집하고 브로커 사양의 보안에 대한 참조를 추가합니다.

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

5.4.6.3.2. Kafka 브로커에 대한 SASL 인증 구성

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

사전 요구 사항

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

절차

1. knative-eventing 네임스페이스에 인증서 파일을 시크릿으로 생성합니다.

```
$ oc create secret -n knative-eventing generic <secret_name> \
  --from-literal=protocol=SASL_SSL \
  --from-literal=sasl.mechanism=<sasl_mechanism> \
  --from-file=ca.crt=caroot.pem \
  --from-literal=password="SecretPassword" \
  --from-literal=user="my-sasl-user"
```

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

```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
```

```
  --from-literal=tls.enabled=true \
  --from-literal=password="SecretPassword" \
  --from-literal=saslType="SCRAM-SHA-512" \
  --from-literal=user="my-sasl-user"
```

2. KnativeKafka CR을 편집하고 브로커 사양의 보안에 대한 참조를 추가합니다.

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

5.4.6.4. 추가 리소스

- Red Hat AMQ Streams 설명서
- Kafka의 TLS 및 SASL

5.4.7. 브로커 관리

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

5.4.7.1. Knative CLI를 사용하여 기존 브로커 나열

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

사전 요구 사항

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

- Knative(kn) CLI가 설치되어 있습니다.

절차

- 기존 브로커를 모두 나열합니다.

```
$ kn broker list
```

출력 예

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

5.4.7.2. Knative CLI를 사용하여 기존 브로커 설명

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

사전 요구 사항

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

절차

- 기존 브로커를 설명합니다.

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

기본 브로커를 사용하는 명령의 예

```
$ kn broker describe default
```

출력 예

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

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

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

5.4.8. 이벤트 전달

이벤트가 이벤트 싱크로 전달되지 않는 경우 적용되는 이벤트 전달 매개변수를 구성할 수 있습니다. **dead letter sink**를 포함하여 이벤트 전달 매개변수를 구성하면 이벤트 싱크로 전달되지 못한 이벤트를 다시 수행할 수 있습니다. 그렇지 않으면 전달되지 않은 이벤트가 삭제됩니다.

5.4.8.1. 채널 및 브로커에 대한 이벤트 전달 동작 패턴

다양한 채널 및 브로커 유형에는 이벤트 전달을 위해 그 뒤에 자체 동작 패턴이 있습니다.

5.4.8.1.1. Knative Kafka 채널 및 브로커

이벤트가 **Kafka** 채널 또는 브로커 수신자에게 성공적으로 전달되면 수신자는 **202** 상태 코드로 응답합니다. 즉, 이벤트가 **Kafka** 주제 내부에 안전하게 저장되어 손실되지 않습니다.

수신자가 다른 상태 코드로 응답하는 경우 이벤트는 안전하게 저장되지 않으며 문제를 해결하기 위해 사용자가 단계를 수행해야 합니다.

5.4.8.2. 구성 가능한 이벤트 전달 매개변수

이벤트 전달을 위해 다음 매개변수를 구성할 수 있습니다.

dead letter sink

이벤트가 전달되지 않으면 지정된 이벤트 싱크에 저장되도록 **deadLetterSink** 전달 매개변수를 구성할 수 있습니다. **dead letter sink**에 저장되지 않은 이벤트가 삭제됩니다. **dead letter sink**는 **Knative** 서비스, **Kubernetes** 서비스 또는 **URI**와 같은 **Knative Eventing** 싱크 계약을 준수하는 주소 지정 가능한 오브젝트입니다.

retries

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

back off delay

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

back off policy

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

5.4.8.3. 이벤트 전달 매개변수 구성 예

브로커,트리거,채널, 서브스크립션 오브젝트에 대한 이벤트 전달 매개변수를 구성할 수 있습니다. 브로커 또는 채널에 대한 이벤트 전달 매개변수를 구성하는 경우 이러한 매개변수는 해당 오브젝트에 대해 생성된 트리거 또는 서브스크립션으로 전파됩니다. 브로커 또는 채널의 설정을 재정의하도록 트리거 또는 서브스크립션에 대한 이벤트 전달 매개변수를 설정할 수도 있습니다.

Broker 오브젝트의 예

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

```
metadata:
...
spec:
  delivery:
    deadLetterSink:
      ref:
        apiVersion: eventing.knative.dev/v1alpha1
        kind: KafkaSink
        name: <sink_name>
    backoffDelay: <duration>
    backoffPolicy: <policy_type>
    retry: <integer>
...
```

Trigger 오브젝트의 예

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

Channel 오브젝트의 예

```
apiVersion: messaging.knative.dev/v1
kind: Channel
metadata:
...
spec:
  delivery:
    deadLetterSink:
```

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

Subscription 개체 예

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

5.4.8.4. 트리거의 이벤트 전달 순서 구성

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

사전 요구 사항

- **OpenShift Serverless Operator, Knative Eventing, Knative Kafka**가 **OpenShift Container Platform** 클러스터에 설치되어 있습니다.

- Kafka 브로커는 클러스터에서 사용할 수 있도록 활성화되며 Kafka 브로커를 생성했습니다.

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

- OpenShift(oc) CLI를 설치했습니다.

절차

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

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

지원되는 소비자 제공 보장은 다음과 같습니다.

순서가 지정되지 않음

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

ordered

순서가 지정된 소비자는 CloudEvent 구독자의 성공적인 응답을 기다린 후 파티션의 다음 메시지를 전달하기 전에 대기 중인 소비자별 소비자입니다.

기본 주문 보장은 순서가 지정되지 않습니다.

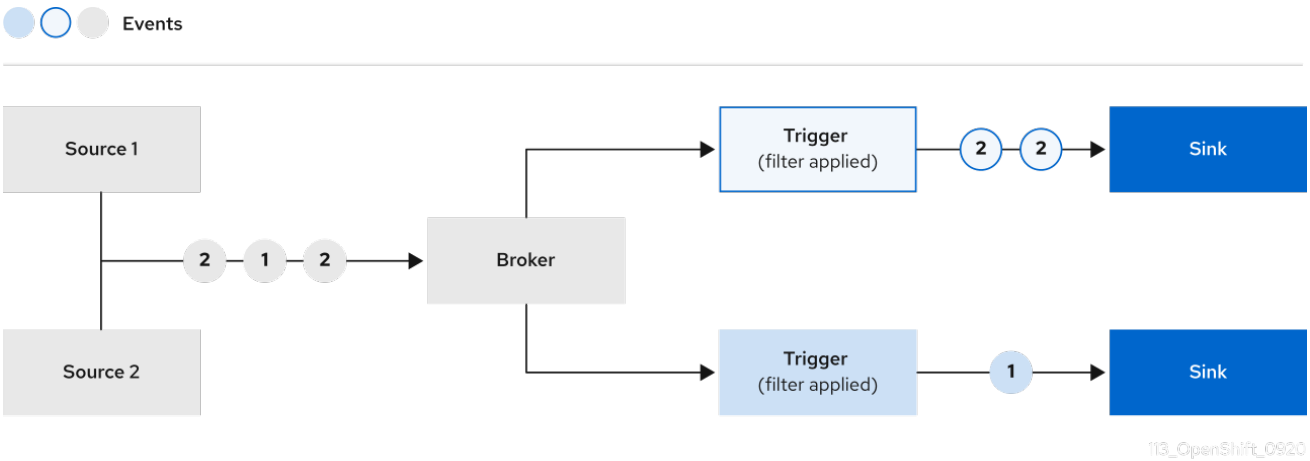
2. Trigger 오브젝트를 적용합니다.

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

5.5. TRIGGER

5.5.1. 트리거 개요

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



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

5.5.1.1. 트리거의 이벤트 전달 순서 구성

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

사전 요구 사항

- **OpenShift Serverless Operator, Knative Eventing, Knative Kafka**가 **OpenShift Container Platform** 클러스터에 설치되어 있습니다.
- **Kafka** 브로커는 클러스터에서 사용할 수 있도록 활성화되며 **Kafka** 브로커를 생성했습니다.
- 프로젝트를 생성했거나 **OpenShift Container Platform**에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- **OpenShift(oc) CLI**를 설치했습니다.

절차

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

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

지원되는 소비자 제공 보장은 다음과 같습니다.

순서가 지정되지 않음

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

ordered

순서가 지정된 소비자는 **CloudEvent** 구독자의 성공적인 응답을 기다린 후 파티션의 다음 메시지를 전달하기 전에 대기 중인 소비자별 소비자입니다.

기본 주문 보장은 순서가 지정되지 않습니다.

2. **Trigger** 오브젝트를 적용합니다.

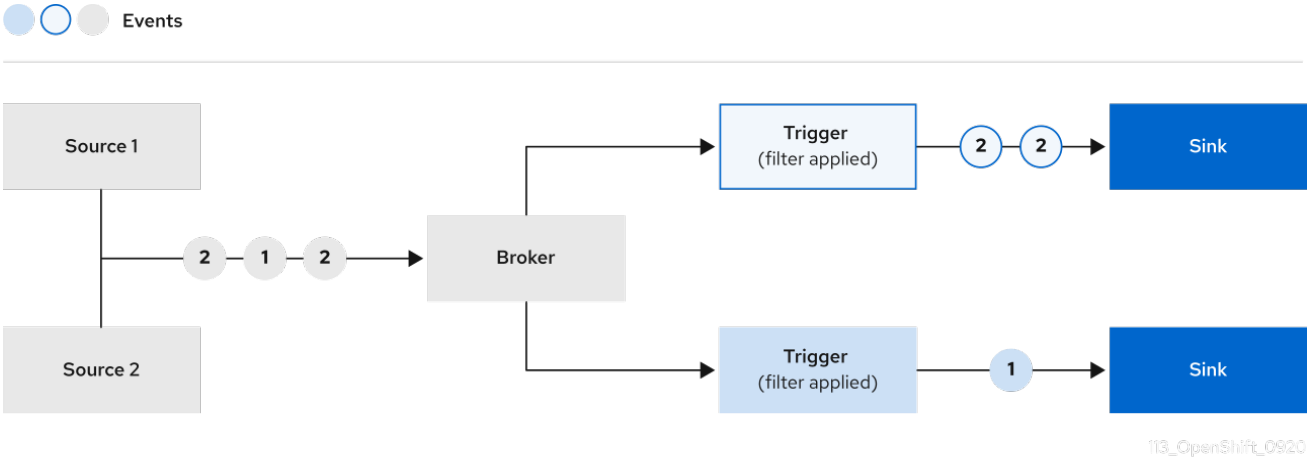
```
$ oc apply -f <filename>
```

5.5.1.2. 다음 단계

- 이벤트가 이벤트 싱크로 전달되지 않는 경우 적용되는 이벤트 전달 매개변수를 구성합니다. 이벤트 전달 매개변수 구성 예를 참조하십시오.

5.5.2. 트리거 생성

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



5.5.2.1. 관리자 화면을 사용하여 트리거 생성

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

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Eventing**이 **OpenShift Container Platform** 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 관리자 화면에 있습니다.
- **OpenShift Container Platform**에 대한 클러스터 관리자 권한이 있습니다.
- **Knative** 브로커를 생성했습니다.
- 구독자로 사용할 **Knative** 서비스를 생성했습니다.

절차

1. **OpenShift Container Platform** 웹 콘솔의 관리자 화면에서 **Serverless → Eventing**으로 이동합니다.
2. 브로커 탭에서 트리거를 추가할 브로커의 옵션 메뉴

를 선택합니다.

3. 목록에서 트리거 추가를 클릭합니다.

4. 트리거 추가 대화 상자에서 트리거에 대한 구독자를 선택합니다. 구독자는 브로커에서 이벤트를 수신하는 **Knative** 서비스입니다.

5. 추가를 클릭합니다.

5.5.2.2. 개발자 화면을 사용하여 트리거 생성

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

사전 요구 사항

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

절차

1. 개발자 화면에서 토폴로지 페이지로 이동합니다.

2. 트리거를 생성할 브로커 위로 마우스 커서를 이동한 후 화살표를 끕니다. 트리거 추가 옵션이 표시됩니다.

3. 트리거 추가를 클릭합니다.

4. **Subscriber** 목록에서 싱크를 선택합니다.

5. 추가를 클릭합니다.

검증

- 서브스크립션이 생성되면 **Topology** 페이지에서 브로커를 이벤트 싱크에 연결하는 선으로 표시할 수 있습니다.

트리거 삭제

1. 개발자 화면에서 토폴로지 페이지로 이동합니다.

2. 삭제할 트리거를 클릭합니다.

3. 작업 컨텍스트 메뉴에서 트리거 삭제를 선택합니다.

5.5.2.3. Knative CLI를 사용하여 트리거 생성

kn trigger create 명령을 사용하여 트리거를 생성할 수 있습니다.

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Eventing**이 **OpenShift Container Platform** 클러스터에 설치되어 있습니다.

- **Knative(kn) CLI**가 설치되어 있습니다.

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

절차

- 트리거를 생성합니다.

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

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

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

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

5.5.3. 명령줄의 트리거 나열

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

5.5.3.1. Knative CLI를 사용하여 트리거 나열

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

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Eventing**이 **OpenShift Container Platform** 클러스터에 설치되어 있습니다.
- **Knative(kn) CLI**가 설치되어 있습니다.

절차

1. 사용 가능한 트리거 목록을 인쇄합니다.

```
$ kn trigger list
```

출력 예

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

2. 선택 사항: JSON 형식으로 된 트리거 목록을 인쇄합니다.

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

5.5.4. 명령줄에서 트리거 설명

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

5.5.4.1. Knative CLI를 사용하여 트리거 설명

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

사전 요구 사항

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

절차

- 명령을 입력합니다.

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

출력 예

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

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

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

5.5.5. 트리거를 싱크에 연결

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

Kafka 싱크에 연결된 **Trigger** 오브젝트의 예

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

```
apiVersion: eventing.knative.dev/v1alpha1
kind: KafkaSink
name: <kafka_sink_name> 2
```

1

싱크에 연결되어 있는 트리거의 이름입니다.

2

KafkaSink 오브젝트의 이름입니다.

5.5.6. 명령줄에서 트리거 필터링

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

5.5.6.1. Knative CLI를 사용하여 트리거로 이벤트 필터링

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

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

여러 특성을 사용하여 이벤트를 필터링할 수도 있습니다. 다음 예제에서는 **type, source** 및 **extension** 특성을 사용하여 이벤트를 필터링하는 방법을 보여줍니다.

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

5.5.7. 명령줄에서 트리거 업데이트

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

5.5.7.1. Knative CLI를 사용하여 트리거 업데이트

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

사전 요구 사항

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

프로세스

- 태그를 업데이트합니다.

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

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

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

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

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

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

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

5.5.8. 명령줄에서 트리거 삭제

Knative(kn) CLI를 사용하여 트리거를 삭제하면 간소화되고 직관적인 사용자 인터페이스가 제공됩니

다.

5.5.8.1. Knative CLI를 사용하여 트리거 삭제

kn trigger delete 명령을 사용하여 트리거를 삭제할 수 있습니다.

사전 요구 사항

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

프로세스

- 트리거를 삭제합니다.

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

검증

1. 기존 트리거를 나열합니다.

```
$ kn trigger list
```

2. 트리거가 더 이상 존재하지 않는지 확인합니다.

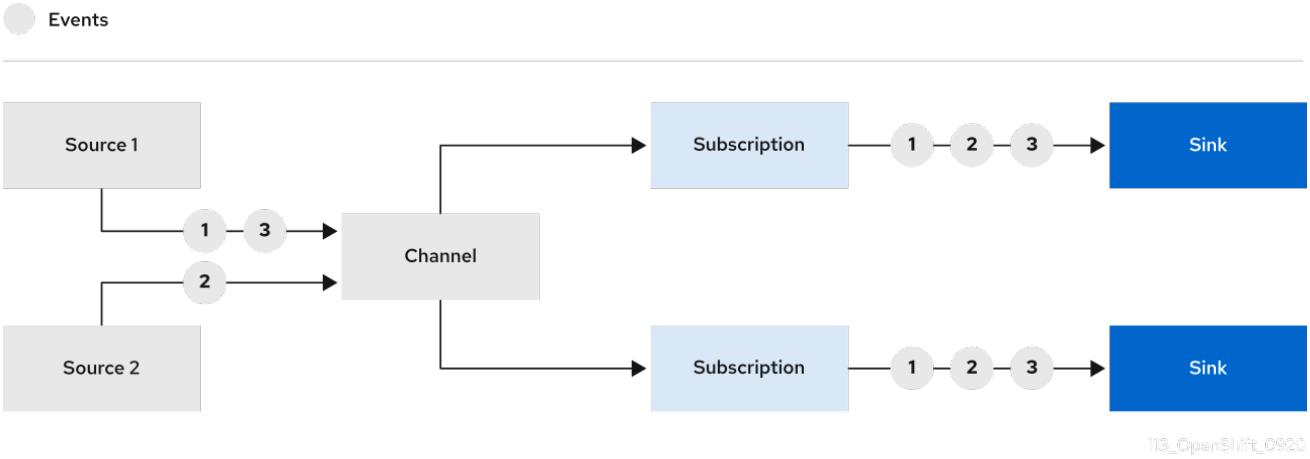
출력 예

```
No triggers found.
```

5.6. 채널

5.6.1. 채널 및 서브스크립션

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

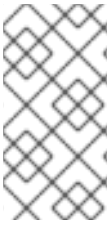


지원되는 Channel 오브젝트를 인스턴스화하여 채널을 생성하고 Subscription 오브젝트에서 delivery 사양을 수정하여 재전송 시도 횟수를 구성할 수 있습니다.

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

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

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



참고

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

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

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

5.6.1.1. 채널 구현 유형

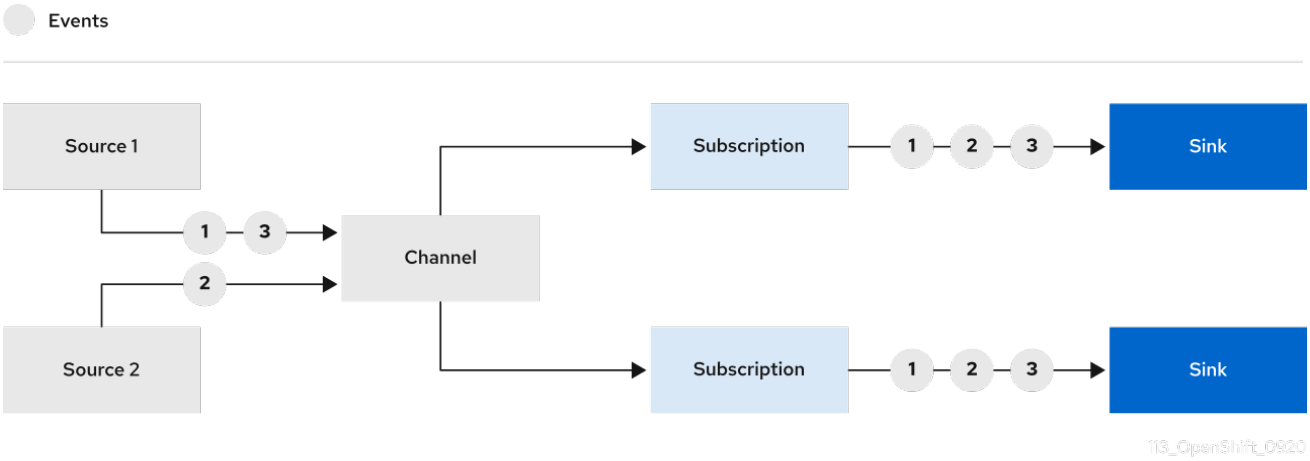
InMemoryChannel 및 KafkaChannel 채널 구현은 OpenShift Serverless에서 개발을 위해 사용할 수 있습니다.

다음은 InMemoryChannel 유형 채널에 대한 제한 사항입니다.

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

5.6.2. 채널 생성

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



지원되는 **Channel** 오브젝트를 인스턴스화하여 채널을 생성하고 **Subscription** 오브젝트에서 **delivery** 사양을 수정하여 재전송 시도 횟수를 구성할 수 있습니다.

5.6.2.1. 관리자 화면을 사용하여 채널 생성

클러스터에 **Knative Eventing**을 설치한 후 관리자 화면을 사용하여 채널을 생성할 수 있습니다.

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Eventing**이 **OpenShift Container Platform** 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 관리자 화면에 있습니다.
- **OpenShift Container Platform**에 대한 클러스터 관리자 권한이 있습니다.

절차

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

참고

현재 **InMemoryChannel** 채널 오브젝트만 기본적으로 지원됩니다. **OpenShift Serverless**에 **Knative Kafka**를 설치한 경우 **Kafka** 채널을 사용할 수 있습니다.

4. 생성을 클릭합니다.

5.6.2.2. 개발자 화면을 사용하여 채널 생성

OpenShift Container Platform 웹 콘솔을 사용하면 간소화되고 직관적인 사용자 인터페이스를 사용하여 채널을 생성할 수 있습니다. **Knative Eventing**이 클러스터에 설치되면 웹 콘솔을 사용하여 채널을 생성할 수 있습니다.

사전 요구 사항

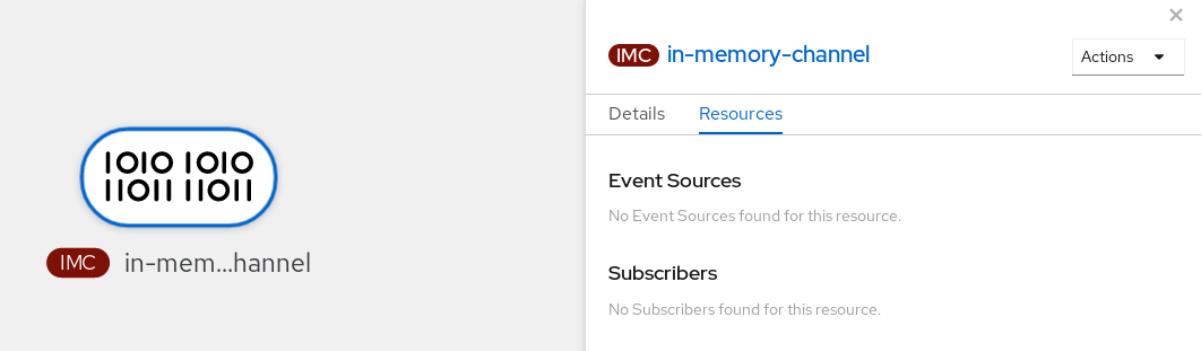
- **OpenShift Container Platform** 웹 콘솔에 로그인했습니다.
- **OpenShift Serverless Operator** 및 **Knative Eventing**이 **OpenShift Container Platform** 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 **OpenShift Container Platform**에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

절차

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

검증

- 토폴로지 페이지로 이동하여 채널이 있는지 확인합니다.



5.6.2.3. Knative CLI를 사용하여 채널 생성

Knative(kn) CLI를 사용하여 채널을 생성하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다. kn channel create 명령을 사용하여 채널을 생성할 수 있습니다.

사전 요구 사항

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

절차

- 채널을 생성합니다.

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

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

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

출력 예

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

-

검증

- 채널이 존재하는지 확인하려면 기존 채널을 나열하고 출력을 검사합니다.

```
$ kn channel list
```

출력 예

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

채널 삭제

- 채널을 삭제합니다.

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

5.6.2.4. YAML을 사용하여 기본 구현 채널 생성

YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 방식으로 채널을 선언적으로 설명할 수 있으며 재현할 수 있는 방식으로 채널을 설명할 수 있습니다. YAML을 사용하여 서버리스 채널을 생성하려면 Channel 오브젝트를 정의하는 YAML 파일을 생성한 다음 oc apply 명령을 사용하여 적용해야 합니다.

사전 요구 사항

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

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

절차

1. Channel 오브젝트를 YAML 파일로 생성합니다.

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

2. YAML 파일을 적용합니다.

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

5.6.2.5. YAML을 사용하여 Kafka 채널 생성

YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 방식으로 채널을 선언적으로 설명할 수 있으며 재현할 수 있는 방식으로 채널을 설명할 수 있습니다. Kafka 채널을 생성하여 Kafka 항목에서 지원하는 Knative Eventing 채널을 생성할 수 있습니다. YAML을 사용하여 Kafka 채널을 생성하려면 KafkaChannel 오브젝트를 정의하는 YAML 파일을 생성한 다음 oc apply 명령을 사용하여 적용해야 합니다.

사전 요구 사항

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

절차

1. KafkaChannel 오브젝트를 YAML 파일로 생성합니다.

-

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

중요

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

2. KafkaChannel YAML 파일을 적용합니다.

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

5.6.2.6. 다음 단계

- 채널을 생성한 후 이벤트 싱크에서 채널을 구독하고 이벤트를 수신할 수 있는 서브스크립션을 생성합니다.
- 이벤트가 이벤트 싱크로 전달되지 않는 경우 적용되는 이벤트 전달 매개변수를 구성합니다. 이벤트 전달 매개변수 구성 예를 참조하십시오.

5.6.3. 기본 채널 구현

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

5.6.3.1. 기본 채널 구현 구성

사전 요구 사항

- OpenShift Container Platform에 대한 관리자 권한이 있습니다.

- OpenShift Serverless Operator 및 Knative Eventing을 클러스터에 설치했습니다.

- Kafka 채널을 기본 채널 구현으로 사용하려면 클러스터에 KnativeKafka CR도 설치해야 합니다.

절차

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

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

(1) spec.config에서 수정된 구성을 추가할 구성 맵을 지정할 수 있습니다.

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

(3)

4

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

중요

네임스페이스별 기본값을 구성하면 클러스터 수준 설정을 덮어씁니다.

5.6.4. Knative Kafka 채널의 보안 구성

5.6.4.1. Kafka 채널의 TLS 인증 구성

TLS(*Transport Layer Security*)는 **Apache Kafka** 클라이언트와 서버에서 **Knative**와 **Kafka** 간의 트래픽 암호화 및 인증에 사용됩니다. **TLS**는 **Knative Kafka**에서 지원되는 유일한 트래픽 암호화 방법입니다.

사전 요구 사항

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

절차

1. 선택한 네임스페이스에서 인증서 파일을 시크릿으로 생성합니다.

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

중요

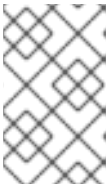
키 이름 **ca.crt,user.crt, user.key**를 사용합니다. 이 값은 변경하지 마십시오.

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

```
$ oc edit knativekafka
```

3. 시크릿 및 시크릿의 네임스페이스를 참조합니다.

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



참고

부트스트랩 서버에서 일치하는 포트를 지정해야 합니다.

예를 들어 다음과 같습니다.

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  namespace: knative-eventing
```

```
  name: knative-kafka
spec:
  channel:
    authSecretName: tls-user
    authSecretNamespace: kafka
    bootstrapServers: eventing-kafka-bootstrap.kafka.svc:9094
    enabled: true
  source:
    enabled: true
```

5.6.4.2. Kafka 채널의 SASL 인증 구성

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

사전 요구 사항

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

절차

1. 선택한 네임스페이스에서 인증서 파일을 시크릿으로 생성합니다.

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

- 키 이름 **ca.crt**,암호, **sasl.mechanism** 을 사용합니다. 이 값은 변경하지 마십시오.

- 공용 **CA** 인증서와 함께 **SASL**을 사용하려면 시크릿을 생성할 때 **ca.crt** 인수 대신 **tls.enabled=true** 플래그를 사용해야 합니다. 예를 들어 다음과 같습니다.

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

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

```
$ oc edit knativekafka
```

3. 시크릿 및 시크릿의 네임스페이스를 참조합니다.

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

참고

부트스트랩 서버에서 일치하는 포트를 지정해야 합니다.

예를 들어 다음과 같습니다.

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

5.7. 서브스크립션

5.7.1. 서브스크립션 생성

채널 및 이벤트 싱크를 생성한 후 이벤트 전달을 활성화하는 서브스크립션을 생성할 수 있습니다. 서브스크립션은 이벤트를 전달할 채널 및 싱크(서브스크립션자라고도 함)를 지정하는 **Subscription** 오브젝트를 구성하여 생성합니다.

5.7.1.1. 관리자 화면을 사용하여 서브스크립션 생성

*채널과 구독자 라고도 하는 이벤트 싱크를 생성한 후에는 서브스크립션을 생성하여 이벤트 전달을 활성화할 수 있습니다. 서브스크립션은 이벤트를 전달할 채널과 구독자를 지정하는 **Subscription** 오브젝트를 구성하여 생성합니다. 또한 오류를 처리하는 방법과 같은 몇 가지 구독자별 옵션을 지정할 수도 있습니다.*

사전 요구 사항

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

- *Knative 채널을 생성했습니다.*

- *구독자로 사용할 Knative 서비스를 생성했습니다.*

절차

1. *OpenShift Container Platform 웹 콘솔의 관리자 화면에서 Serverless → Eventing으로 이동합니다.*

2. *채널 탭에서 서브스크립션을 추가할 채널의 옵션 메뉴*

를 선택합니다.

3. *목록에서 서브스크립션 추가를 클릭합니다.*

4. *서브스크립션 추가 대화 상자에서 서브스크립션에 대한 구독자를 선택합니다. 구독자는 채널에서 이벤트를 수신하는 Knative 서비스입니다.*

5. *추가를 클릭합니다.*

5.7.1.2. 개발자 화면을 사용하여 서브스크립션 생성

채널 및 이벤트 싱크를 생성한 후 이벤트 전달을 활성화하는 서브스크립션을 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔을 사용하면 간결하고 직관적인 사용자 인터페이스를 사용하여 서브스크립션을 생성할 수 있습니다.

사전 요구 사항

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

- *웹 콘솔에 로그인했습니다.*

- 이벤트 싱크(예: **Knative** 서비스) 및 채널을 생성했습니다.

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

절차

1. 개발자 화면에서 토폴로지 페이지로 이동합니다.

2. 다음 방법 중 하나를 사용하여 서브스크립션을 생성합니다.

   a. 서브스크립션을 생성할 채널 위로 마우스 커서를 이동하고 화살표를 끕니다. 서브스크립션 추가 옵션이 표시됩니다.

      i. **Subscriber** 목록에서 싱크를 선택합니다.

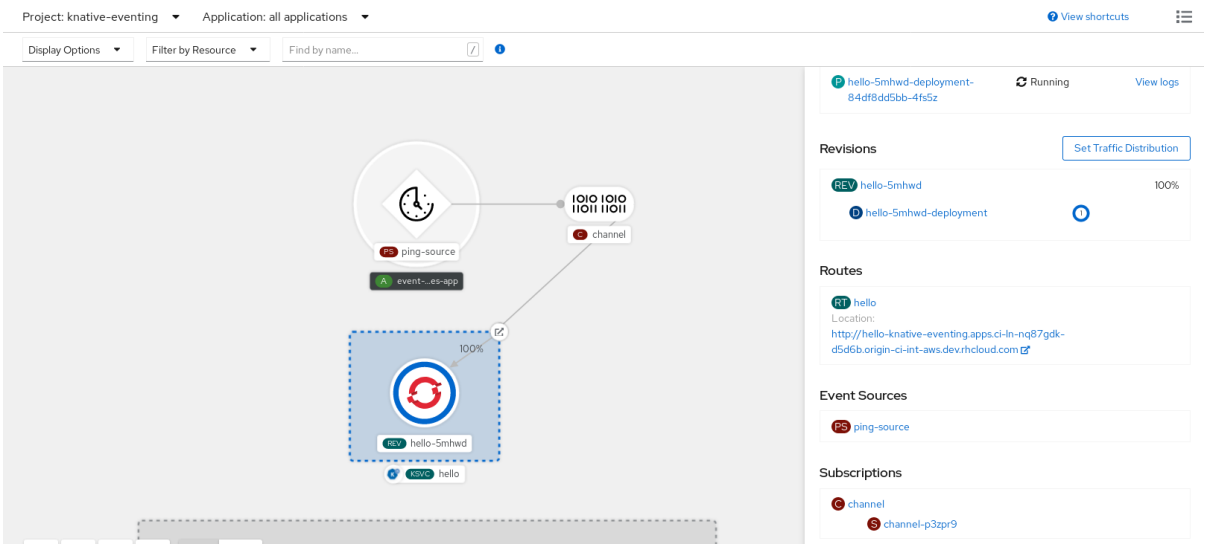
      ii. 추가를 클릭합니다.

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

검증

- 서브스크립션이 생성되면 토폴로지 보기에 채널을 서비스에 연결하는 선이 표시되어 이를

확인할 수 있습니다.



5.7.1.3. YAML을 사용하여 서브스크립션 생성

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

사전 요구 사항

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

절차

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

```
apiVersion: messaging.knative.dev/v1beta1
kind: Subscription
```

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

(1) 서브스크립션 이름입니다.

(2) 서브스크립션이 연결되는 채널의 구성 설정입니다.

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

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

- **YAML** 파일을 적용합니다.

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

5.7.1.4. Knative CLI를 사용하여 서브스크립션 생성

채널 및 이벤트 싱크를 생성한 후 이벤트 전달을 활성화하는 서브스크립션을 생성할 수 있습니다.

Knative(kn) CLI를 사용하여 서브스크립션을 생성하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다. kn subscription create 명령을 적절한 플래그와 함께 사용하여 서브스크립션을 생성할 수 있습니다.

사전 요구 사항

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

절차

- 싱크를 채널에 연결하는 서브스크립션을 생성합니다.

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

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

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

ksvc

Knative 서비스입니다.

channel

대상으로 사용해야 하는 채널입니다. 여기에서는 기본 채널 유형만 참조할 수

있습니다.

broker

Eventing 브로커입니다.

3

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

명령 예

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

출력 예

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

검증

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

```
$ kn subscription list
```

출력 예

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

서브스크립션 삭제

- 서브스크립션을 삭제합니다.

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

5.7.1.5. 다음 단계

- 이벤트가 이벤트 싱크로 전달되지 않는 경우 적용되는 이벤트 전달 매개변수를 구성합니다. 이벤트 전달 매개변수 구성 예를 참조하십시오.

5.7.2. 서브스크립션 관리

5.7.2.1. Knative CLI를 사용하여 서브스크립션 설명

kn subscription describe 명령을 사용하여 **Knative(kn) CLI**를 사용하여 터미널에서 서브스크립션에 대한 정보를 출력할 수 있습니다. **Knative CLI**를 사용하여 서브스크립션을 설명하는 경우 **YAML** 파일을 직접 보는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다.

사전 요구 사항

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

절차

- 서브스크립션을 설명합니다.

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

출력 예

```
Name:            my-subscription
```

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

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

5.7.2.2. Knative CLI를 사용하여 서브스크립션 나열

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

사전 요구 사항

- **Knative(kn) CLI**가 설치되어 있습니다.

절차

- 클러스터의 서브스크립션을 나열합니다.

```
$ kn subscription list
```

출력 예

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

5.7.2.3. Knative CLI를 사용하여 서브스크립션 업데이트

kn subscription update 명령과 적절한 플래그를 사용하여 **Knative(kn) CLI**를 사용하여 터미널에서 서브스크립션을 업데이트할 수 있습니다. **Knative CLI**를 사용하여 서브스크립션을 업데이트하는 경우 **YAML** 파일을 직접 업데이트하는 것보다 더 효율적이고 직관적인 사용자 인터페이스를 제공합니다.

사전 요구 사항

- **Knative(kn) CLI**가 설치되어 있습니다.
- 서브스크립션이 생성되어 있습니다.

절차

- 서브스크립션을 업데이트합니다.

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

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

   ksvc
   : **Knative** 서비스입니다.

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

   broker
   : **Eventing** 브로커입니다.

2.

명령 예

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

5.8. 이벤트 검색

5.8.1. 이벤트 소스 및 이벤트 소스 유형 나열

존재하는 모든 이벤트 소스 또는 이벤트 소스 유형 목록을 볼 수 있으며, OpenShift Container Platform 클러스터에서 사용할 수 있습니다. Knative (kn) CLI 또는 OpenShift Container Platform 웹 콘솔의 개발자 화면을 사용하여 사용 가능한 이벤트 소스 또는 이벤트 소스 유형을 나열할 수 있습니다.

5.8.2. 명령줄에서 이벤트 소스 유형 나열

Knative(kn) CLI를 사용하면 간소화되고 직관적인 사용자 인터페이스를 통해 클러스터에서 사용 가능한 이벤트 소스 유형을 볼 수 있습니다.

5.8.2.1. Knative CLI를 사용하여 사용 가능한 이벤트 소스 유형 나열

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

사전 요구 사항

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

절차

1. 터미널에서 사용 가능한 이벤트 소스 유형을 나열합니다.

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

출력 예

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

2. 선택 사항: 사용 가능한 이벤트 소스 유형을 **YAML** 형식으로 나열할 수도 있습니다.

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

5.8.3. 개발자 화면에서 이벤트 소스 유형 나열

클러스터에서 사용 가능한 모든 이벤트 소스 유형 목록을 볼 수 있습니다. **OpenShift Container Platform** 웹 콘솔을 사용하면 사용 가능한 이벤트 소스 유형을 볼 수 있도록 간소화되고 직관적인 사용자 인터페이스가 제공됩니다.

5.8.3.1. 개발자 화면 내에서 사용 가능한 이벤트 소스 유형 보기

사전 요구 사항

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

절차

1. 개발자 화면에 액세스합니다.

2. *+추가를 클릭합니다.*

3. *이벤트 소스를 클릭합니다.*

4. *사용 가능한 이벤트 소스 유형을 확인합니다.*

5.8.4. 명령줄에서 이벤트 소스 나열

Knative(kn) CLI를 사용하면 간소화되고 직관적인 사용자 인터페이스를 제공하여 클러스터의 기존 이벤트 소스를 볼 수 있습니다.

5.8.4.1. Knative CLI를 사용하여 사용 가능한 이벤트 소스 나열

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

사전 요구 사항

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

절차

1. *터미널에서 기존 이벤트 소스를 나열합니다.*

```
$ kn source list
```

출력 예

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

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

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

명령 예

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

출력 예

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

5.9. 이벤트 구성 튜닝

5.9.1. Knative Eventing 시스템 배포 구성 덮어쓰기

*KnativeEventing CR(사용자 정의 리소스)에서 **deployments** 사양을 수정하여 일부 특정 배포에 대한 기본 구성을 덮어쓸 수 있습니다.*

5.9.1.1. 배포 구성 덮어쓰기

*현재 **eventing-controller,eventing-webhook, imc-controller** 필드에는 기본 구성 설정을 재정의하고 프로브의 **readiness** 및 **liveness** 필드에도 지원됩니다.*

중요

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

다음 예에서 KnativeEventing CR은 eventing-controller 배포를 재정의하여 다음을 수행합니다.

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

KnativeEventing CR 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  deployments:
  - name: eventing-controller
    readinessProbes: 1
      - container: controller
        timeoutSeconds: 10
    resources:
    - container: eventing-controller
      requests:
        cpu: 300m
        memory: 100Mi
```

```
    limits:
      cpu: 1000m
      memory: 250Mi
  replicas: 3
  labels:
    example-label: label
  annotations:
    example-annotation: annotation
  nodeSelector:
    disktype: hdd
```

1

준비 상태 및 활성 상태 프로브 덮어쓰기를 사용하여 프로브에 지정된 대로 **Kubernetes API**에 지정된 대로 배포 컨테이너에 있는 프로브의 모든 필드를 재정의할 수 있습니다. 그러나 프로브 처리기와 관련된 필드인 **exec,grpc,httpGet, tcpSocket.**

참고

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

추가 리소스

- **Kubernetes API** 문서의 프로브 구성 섹션

5.9.2. 고가용성

고가용성(**HA**)은 중단이 발생하는 경우 **API**가 작동하도록 하는 데 도움이 되는 **Kubernetes API**의 표준 기능입니다. **HA** 배포에서 활성 컨트롤러가 충돌하거나 삭제되면 다른 컨트롤러를 쉽게 사용할 수 있습니다. 이 컨트롤러는 현재 사용할 수 없는 컨트롤러에서 서비스하고 있는 **API**를 대신 처리합니다.

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

OpenShift Serverless의 **HA**는 **Knative Serving** 또는 **Eventing** 컨트롤 플레인을 설치하면 기본적으로 활성화되는 리더 선택을 통해 사용할 수 있습니다. 리더 선택 **HA** 패턴을 사용하는 경우에는 요구하기

전에 컨트롤러의 인스턴스가 이미 예약되어 클러스터 내에서 실행됩니다. 이러한 컨트롤러 인스턴스는 리더 선택 잠금이라는 공유 리소스를 사용하기 위해 경쟁합니다. 특정 시점에 리더 선택 잠금 리소스에 액세스할 수 있는 컨트롤러의 인스턴스를 리더라고 합니다.

5.9.2.1. Knative Eventing의 고가용성 복제본 구성

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

참고

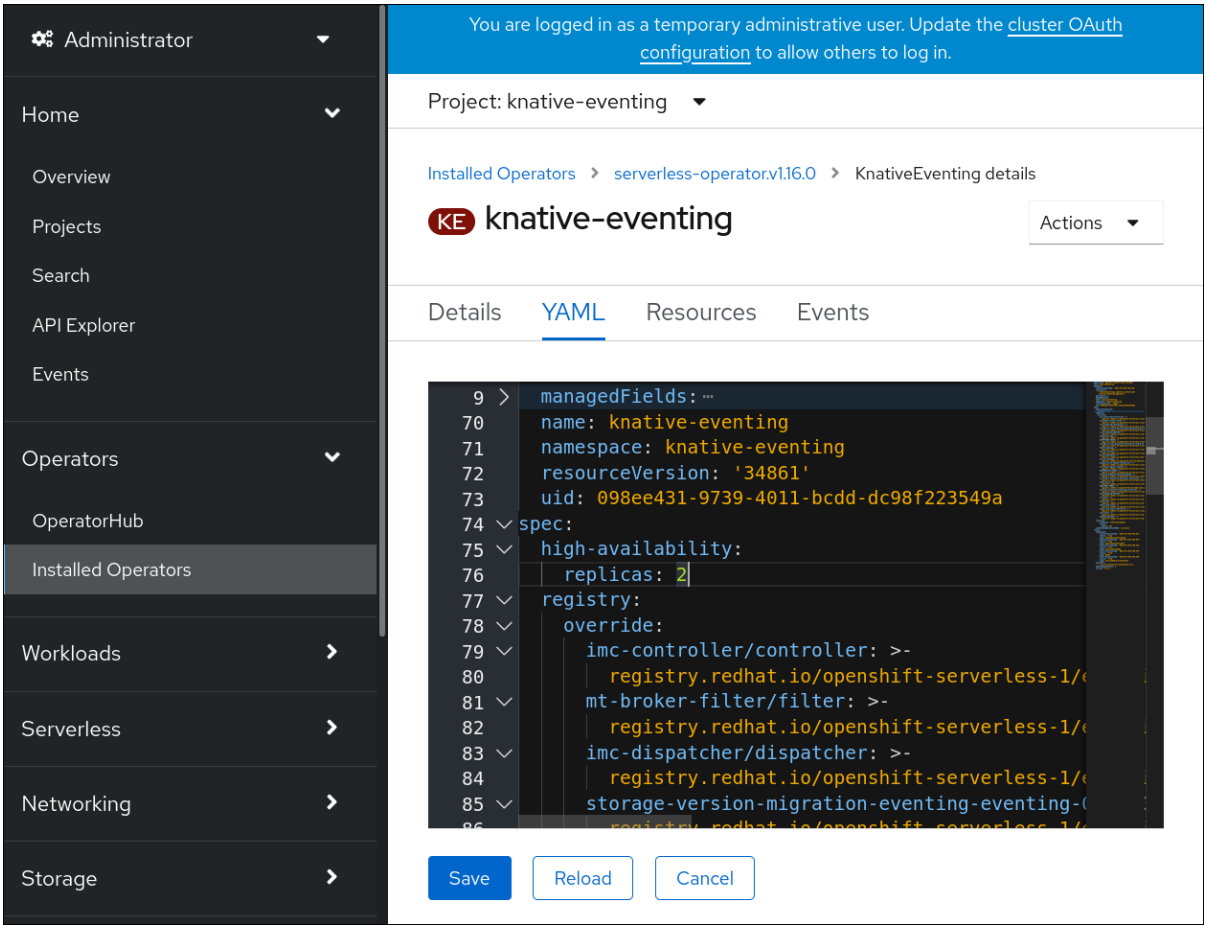
Knative Eventing의 경우 mt-broker-filter 및 mt-broker-ingress 배포는 HA에 의해 확장되지 않습니다. 여러 배포가 필요한 경우 이러한 구성 요소를 수동으로 스케일링합니다.

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.

프로세스

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



5. *KnativeEventing CR의 복제본 수를 수정합니다.*

YAML의 예

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

5.9.2.2. Knative Kafka의 고가용성 복제본 구성

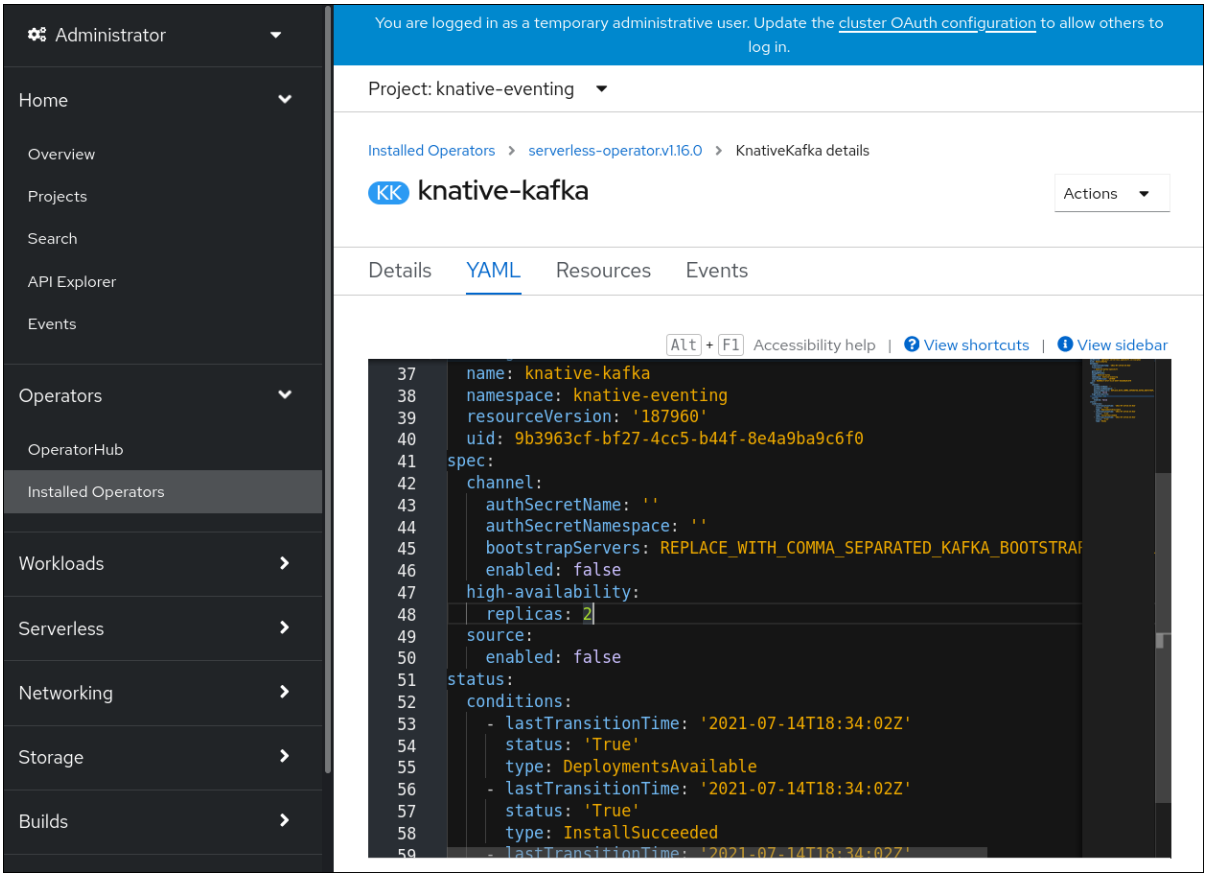
*기본적으로 각 복제본이 두 개로 구성된 **Knative Kafka kafka-controller** 및 **kafka-webhook-eventing** 구성 요소에 **HA(고가용성)**를 사용할 수 있습니다. **KnativeKafka CR(사용자 정의 리소스)**의 **spec.high-availability.replicas** 값을 수정하여 이러한 구성 요소의 복제본 수를 변경할 수 있습니다.*

사전 요구 사항

- *클러스터 관리자 액세스 권한이 있는 **OpenShift Container Platform** 계정에 액세스할 수 있습니다.*
- ***OpenShift Serverless Operator** 및 **Knative Kafka**가 클러스터에 설치되어 있습니다.*

절차

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



5.

5.

KnativeKafka CR의 복제본 수를 수정합니다.

YAML의 예

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

6장. 함수

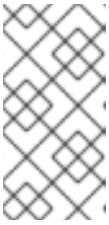
6.1. OPENSHIFT SERVERLESS FUNCTIONS 설정

애플리케이션 코드 배포 프로세스를 개선하기 위해 OpenShift Serverless를 사용하여 이벤트 중심 기능을 OpenShift Container Platform에서 Knative 서비스로 배포할 수 있습니다. 함수를 개발하려면 설정 단계를 완료해야 합니다.

6.1.1. 사전 요구 사항

클러스터에서 OpenShift Serverless Functions을 사용하려면 다음 단계를 완료해야 합니다.

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.



참고

함수는 Knative 서비스로 배포됩니다. 함수에 이벤트 중심 아키텍처를 사용하려면 Knative Eventing도 설치해야 합니다.

- oc CLI 가 설치되어 있어야 합니다.
- Knative(kn) CLI 가 설치되어 있습니다. Knative CLI를 설치하면 함수를 생성하고 관리하는 데 사용할 수 있는 kn func 명령을 사용할 수 있습니다.
- Docker Container Engine 또는 Podman 버전 3.4.7 이상이 설치되어 있습니다.
- OpenShift Container Registry와 같이 사용 가능한 이미지 레지스트리에 액세스할 수 있습니다.
- Quay.io 를 이미지 레지스트리로 사용하는 경우 리포지토리가 비공개가 아닌지 확인하거나 pod가 다른 보안 레지스트리의 이미지를 참조하도록 허용하는 OpenShift Container Platform 설명서를 따라야 합니다.
- OpenShift Container Registry를 사용하는 경우 클러스터 관리자가 레지스트리를 공개해야 합니다.

6.1.2. Podman 설정

고급 컨테이너 관리 기능을 사용하려면 OpenShift Serverless Functions와 함께 Podman을 사용할 수 있습니다. 이를 위해 Podman 서비스를 시작하고 Knative(kn) CLI를 구성하여 연결합니다.

절차

1. ${XDG_RUNTIME_DIR}/podman/podman.sock:의 UNIX 소켓에서 Docker API를 제공하는 Podman 서비스를 시작합니다.

```
$ systemctl start --user podman.socket
```

참고

대부분의 시스템에서 이 소켓은 /run/user/$(id -u)/podman/podman.sock에 있습니다.

2. 기능을 구축하는 데 사용되는 환경 변수를 설정합니다.

```
$ export DOCKER_HOST="unix://${XDG_RUNTIME_DIR}/podman/podman.sock"
```

3. 자세한 출력을 보려면 -v 플래그를 사용하여 함수 프로젝트 디렉터리 내에서 build 명령을 실행합니다. 로컬 UNIX 소켓에 대한 연결이 표시됩니다.

```
$ kn func build -v
```

6.1.3. macOS에서 Podman 설정

고급 컨테이너 관리 기능을 사용하려면 OpenShift Serverless Functions와 함께 Podman을 사용할 수 있습니다. macOS에서 이 작업을 수행하려면 Podman 머신을 시작하고 Knative(kn) CLI를 구성하여 연결합니다.

절차

1. Podman 시스템을 생성합니다.

```
$ podman machine init --memory=8192 --cpus=2 --disk-size=20
```

2. UNIX 소켓에서 Docker API를 제공하는 Podman 시스템을 시작합니다.

```
$ podman machine start
Starting machine "podman-machine-default"
Waiting for VM ...
Mounting volume... /Users/myuser:/Users/user

[...truncated output...]

You can still connect Docker API clients by setting DOCKER_HOST using the
following command in your terminal session:

 export
DOCKER_HOST='unix:///Users/myuser/.local/share/containers/podman/machine/podm
an-machine-default/podman.sock'

Machine "podman-machine-default" started successfully
```

참고

대부분의 macOS 시스템에서 이 소켓은 /Users/myuser/.local/share/containers/podman/machine/podman-default/podman.sock 에 있습니다.

3. 기능을 구축하는 데 사용되는 환경 변수를 설정합니다.

```
$ export
DOCKER_HOST='unix:///Users/myuser/.local/share/containers/podman/machine/podm
an-machine-default/podman.sock'
```

4. 자세한 출력을 보려면 -v 플래그를 사용하여 함수 프로젝트 디렉터리 내에서 build 명령을 실행합니다. 로컬 UNIX 소켓에 대한 연결이 표시됩니다.

```
$ kn func build -v
```

6.1.4. 다음 단계

- Docker Container Engine 또는 Podman에 대한 자세한 내용은 컨테이너 빌드 툴 옵션을 참조하십시오.

- 함수 시작하기를 참조하십시오.

6.2. 함수 시작하기

함수 라이프사이클 관리에는 함수 생성, 빌드 및 배포가 포함됩니다. 선택적으로 배포된 함수를 호출하여 테스트할 수도 있습니다. kn func 툴을 사용하여 OpenShift Serverless에서 이러한 모든 작업을 수행할 수 있습니다.

6.2.1. 사전 요구 사항

다음 절차를 완료하려면 먼저 OpenShift Serverless Functions 설정에서 모든 사전 요구 사항 작업을 완료해야 합니다.

6.2.2. 함수 생성

함수를 빌드하고 배포하려면 Knative(kn) CLI를 사용하여 생성해야 합니다. 경로, 런타임, 템플릿 및 이미지 레지스트리를 명령줄에서 플래그로 지정하거나 -c 플래그를 사용하여 터미널에서 대화형 환경을 시작할 수 있습니다.

사전 요구 사항

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

절차

- 함수 프로젝트를 생성합니다.

```
$ kn func create -r <repository> -l <runtime> -t <template> <path>
```

  - 허용되는 런타임 값에는 quarkus,node,typescript,go,python, Springboot, 및 rust가 포함됩니다.
  - 허용되는 템플릿 값에는 http 및 cloudevents 가 포함됩니다.

명령 예

```
$ kn func create -l typescript -t cloudevents examplefunc
```

출력 예

```
Created typescript function in /home/user/demo/examplefunc
```

- *또는 사용자 지정 템플릿이 포함된 리포지토리를 지정할 수도 있습니다.*

명령 예

```
$ kn func create -r https://github.com/boson-project/templates/ -l node -t hello-world examplefunc
```

출력 예

```
Created node function in /home/user/demo/examplefunc
```

6.2.3. 로컬로 함수 실행

***kn func run** 명령을 사용하여 현재 디렉터리 또는 **--path** 플래그에서 지정한 디렉터리에서 로컬로 함수를 실행할 수 있습니다. 실행 중인 함수가 이전에 빌드되지 않았거나 마지막으로 작성된 프로젝트 파일*

*이 변경된 경우 **kn func run** 명령은 기본적으로 함수를 실행하기 전에 함수를 빌드합니다.*

현재 디렉터리에서 함수를 실행하는 명령의 예

```
$ kn func run
```

경로로 지정된 디렉터리에서 함수를 실행하는 명령의 예

```
$ kn func run --path=<directory_path>
```

--build 플래그를 사용하여 프로젝트 파일이 변경되지 않은 경우에도 함수를 실행하기 전에 기존 이미지를 강제로 다시 빌드할 수도 있습니다.

*빌드 플래그를 사용하는 **run** 명령의 예*

```
$ kn func run --build
```

*빌드 플래그를 **false**로 설정하면 이미지 빌드가 비활성화되고 이전에 빌드한 이미지를 사용하여 함수가 실행됩니다.*

*빌드 플래그를 사용하는 **run** 명령의 예*

```
$ kn func run --build=false
```

help 명령을 사용하여 kn func run 명령 옵션에 대해 자세히 알아볼 수 있습니다.

빌드 도움말 명령

```
$ kn func help run
```

6.2.4. 함수 빌드

함수를 실행하려면 함수 프로젝트를 빌드해야 합니다. kn func run 명령을 사용하는 경우 함수가 자동으로 빌드됩니다. 그러나 kn func build 명령을 사용하여 함수를 실행하지 않고 빌드할 수 있습니다. 이 기능은 고급 사용자 또는 디버깅 시나리오에 유용할 수 있습니다.

kn func build 명령은 컴퓨터 또는 OpenShift Container Platform 클러스터에서 로컬로 실행할 수 있는 OCI 컨테이너 이미지를 생성합니다. 이 명령은 함수 프로젝트 이름과 이미지 레지스트리 이름을 사용하여 함수에 대해 정규화된 이미지 이름을 구성합니다.

6.2.4.1. 이미지 컨테이너 유형

기본적으로 kn func 빌드 는 Red Hat S2I(Source-to-Image) 기술을 사용하여 컨테이너 이미지를 생성합니다.

Red Hat S2I(Source-to-Image)를 사용하는 빌드 명령의 예

```
$ kn func build
```

6.2.4.2. 이미지 레지스트리 유형

OpenShift Container Registry는 기본적으로 함수 이미지를 저장하기 위한 이미지 레지스트리로 사용됩니다.

OpenShift Container Registry를 사용하는 빌드 명령 예

```
$ kn func build
```

출력 예

```
Building function image
Function image has been built, image: registry.redhat.io/example/example-function:latest
```

--registry 플래그를 사용하여 기본 이미지 레지스트리로 OpenShift Container Registry를 재정의할 수 있습니다.

quay.io를 사용하도록 OpenShift Container Registry를 재정의하는 빌드 명령의 예

```
$ kn func build --registry quay.io/username
```

출력 예

```
Building function image
Function image has been built, image: quay.io/username/example-function:latest
```

6.2.4.3. push 플래그

kn func build 명령에 --push 플래그를 추가하여 성공적으로 빌드한 후 함수 이미지를 자동으로 푸시할 수 있습니다.

OpenShift Container Registry를 사용하는 빌드 명령 예

```
$ kn func build --push
```

6.2.4.4. 도움말 명령

help 명령을 사용하여 kn func build 명령 옵션에 대해 자세히 알아볼 수 있습니다.

빌드 도움말 명령

```
$ kn func help build
```

6.2.5. 함수 배포

kn func deploy 명령을 사용하여 Knative 서비스로 클러스터에 함수를 배포할 수 있습니다. 대상 함수가 이미 배포된 경우 컨테이너 이미지 레지스트리로 푸시된 새 컨테이너 이미지로 업데이트되고 Knative 서비스가 업데이트됩니다.

사전 요구 사항

- *OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.*
- *Knative(kn) CLI가 설치되어 있습니다.*
- *프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.*
- *배포하려는 함수를 이미 생성하고 초기화해야 합니다.*

절차

- 함수를 배포합니다.

```
$ kn func deploy [-n <namespace> -p <path> -i <image>]
```

출력 예

```
Function deployed at: http://func.example.com
```

  - namespace를 지정하지 않으면 함수가 현재 네임스페이스에 배포됩니다.
  - 이 함수는 path를 지정하지 않는 한 현재 디렉터리에서 배포됩니다.
  - Knative 서비스 이름은 프로젝트 이름에서 파생되며 이 명령을 사용하여 변경할 수 없습니다.

6.2.6. 테스트 이벤트를 사용하여 배포된 함수 호출

kn func invoke CLI 명령을 사용하여 로컬에서 또는 OpenShift Container Platform 클러스터에서 함수를 호출하기 위해 테스트 요청을 보낼 수 있습니다. 이 명령을 사용하여 함수가 작동하고 이벤트를 올바르게 수신할 수 있는지 테스트할 수 있습니다. 함수를 로컬로 호출하면 함수 개발 중에 빠른 테스트에 유용합니다. 클러스터에서 함수를 호출하면 프로덕션 환경에 더 가까운 테스트에 유용합니다.

사전 요구 사항

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

- 호출하려는 함수를 이미 배포해야 합니다.

절차

- 함수를 호출합니다.

```
$ kn func invoke
```

  - **kn func invoke** 명령은 현재 실행 중인 로컬 컨테이너 이미지가 있거나 클러스터에 배포된 함수가 있는 경우에만 작동합니다.

  - **kn func invoke** 명령은 기본적으로 로컬 디렉터리에서 실행되며 이 디렉터리는 함수 프로젝트라고 가정합니다.

6.2.7. 함수 삭제

kn func delete 명령을 사용하여 함수를 삭제할 수 있습니다. 이 기능은 함수가 더 이상 필요하지 않은 경우 유용하며 클러스터에 리소스를 저장하는 데 도움이 될 수 있습니다.

절차

- 함수를 삭제합니다.

```
$ kn func delete [<function_name> -n <namespace> -p <path>]
```

  - 삭제할 함수의 이름 또는 경로가 지정되지 않은 경우 현재 디렉터리에서 **func.yaml** 파일을 검색하고 삭제할 함수를 결정합니다.

  - 네임스페이스를 지정하지 않으면 기본값은 **func.yaml** 파일의 **namespace** 값으로 설정됩니다.

6.2.8. 추가 리소스

- 기본 레지스트리를 수동으로 공개

- *Intellij Knative 플러그인의 마켓플레이스 페이지*

- *Visual Studio Code Knative 플러그인에 대한 마켓플레이스 페이지*

6.2.9. 다음 단계

- *Knative Eventing에서 함수 사용을*참조하십시오.

6.3. 클러스터상의 기능 빌드 및 배포

함수를 로컬로 빌드하는 대신 클러스터에서 직접 함수를 빌드할 수 있습니다. 로컬 개발 머신에서 이 워크플로를 사용하는 경우 함수 소스 코드에서만 작업하면 됩니다. 예를 들어 **docker** 또는 **podman**과 같은 클러스터 함수 빌드 툴을 설치할 수 없는 경우 유용합니다.

6.3.1. 클러스터에 함수 빌드 및 배포

Knative(kn) CLI를 사용하여 함수 프로젝트 빌드를 시작한 다음 클러스터에 직접 함수를 배포할 수 있습니다. 이러한 방식으로 함수 프로젝트를 빌드하려면 함수 프로젝트의 소스 코드가 클러스터에 액세스할 수 있는 **Git** 리포지토리 분기에 있어야 합니다.

사전 요구 사항

- **Red Hat OpenShift Pipelines**가 클러스터에 설치되어 있어야 합니다.

- **OpenShift CLI(oc)**가 설치되어 있습니다.

- **Knative(kn) CLI**가 설치되어 있습니다.

절차

1. **Pipeline**을 실행하고 함수를 배포하려는 각 네임스페이스에서 다음 리소스를 생성해야 합니다.

    a. **s2i Tekton** 작업을 생성하여 파이프라인에서 **Source-to-Image**를 사용할 수 있습니다.

```
$ oc apply -f https://raw.githubusercontent.com/openshift-knative/kn-plugin-func/serverless-1.28.0/pipelines/resources/tekton/task/func-s2i/0.1/func-s2i.yaml
```

b. 파이프라인에 함수를 배포할 수 있도록 **kn func deploy Tekton** 작업을 생성합니다.

```
$ oc apply -f https://raw.githubusercontent.com/openshift-knative/kn-plugin-func/serverless-1.28.0/pipelines/resources/tekton/task/func-deploy/0.1/func-deploy.yaml
```

2. 함수를 생성합니다.

```
$ kn func create <function_name> -l <runtime>
```

3. 새 함수 프로젝트를 생성한 후 **Git** 리포지토리에 프로젝트를 추가하고 클러스터에서 리포지토리를 사용할 수 있는지 확인해야 합니다. 이 **Git** 리포지토리에 대한 정보는 다음 단계에서 **func.yaml** 파일을 업데이트하는 데 사용됩니다.

4. 함수 프로젝트의 **func.yaml** 파일에서 **Git** 리포지토리에 대한 클러스터의 구성을 업데이트합니다.

```
...
git:
  url: <git_repository_url> 1
  revision: main 2
  contextDir: <directory_path> 3
...
```

1. 필수 항목입니다. 함수의 소스 코드가 포함된 **Git** 리포지토리를 지정합니다.

2. 선택 사항: 사용할 **Git** 리포지토리 버전을 지정합니다. 분기, 태그 또는 커밋일 수 있습니다.

3. 선택 사항: 함수가 **Git** 리포지토리 루트 폴더에 없는 경우 함수의 디렉터리 경로를 지정합니다.

5.

함수의 비즈니스 로직을 구현합니다. 그런 다음 **Git**을 사용하여 변경 사항을 커밋하고 내보냅니다.

6. 함수를 배포합니다.

```
$ kn func deploy --remote
```

함수 구성에서 참조된 컨테이너 레지스트리에 로그인하지 않은 경우 함수 이미지를 호스팅하는 원격 컨테이너 레지스트리에 대한 인증 정보를 제공하라는 메시지가 표시됩니다.

출력 및 프롬프트의 예

```
  Creating Pipeline resources
Please provide credentials for image registry used by Pipeline.
? Server: https://index.docker.io/v1/
? Username: my-repo
? Password: ********
  Function deployed at URL: http://test-function.default.svc.cluster.local
```

7. 함수를 업데이트하려면 **Git**을 사용하여 새 변경 사항을 커밋하고 푸시한 다음 **kn func deploy --remote** 명령을 다시 실행합니다.

6.3.2. 함수 버전 지정

클러스터에서 함수를 빌드하고 배포하는 경우 리포지토리 내에서 **Git** 리포지토리, 분기 및 하위 디렉터리를 지정하여 함수 코드의 위치를 지정해야 합니다. 기본 분기를 사용하는 경우 분기를 지정할 필요가 없습니다. 마찬가지로 함수가 리포지토리의 루트에 있는 경우 하위 디렉터리를 지정할 필요가 없습니다. **func.yaml** 구성 파일에서 이러한 매개변수를 지정하거나 **kn func deploy** 명령과 함께 플래그를 사용하여 지정할 수 있습니다.

사전 요구 사항

- **Red Hat OpenShift Pipelines**가 클러스터에 설치되어 있어야 합니다.
- **OpenShift(oc) CLI**를 설치했습니다.

- Knative(kn) CLI가 설치되어 있습니다.

절차

- 함수를 배포합니다.

```
$ kn func deploy --remote \ 1
        --git-url <repo-url> \ 2
        [--git-branch <branch>] \ 3
        [--git-dir <function-dir>] 4
```

1 --remote 플래그를 사용하면 빌드가 원격으로 실행됩니다.

2 & lt;repo-url& gt;을 Git 리포지토리의 URL로 바꿉니다.

3 & lt;branch& gt;를 Git 분기, 태그 또는 커밋으로 바꿉니다. 기본 분기에서 최신 커밋을 사용하는 경우 이 플래그를 건너뛸 수 있습니다.

4 리포지토리 루트 디렉터리와 다른 경우 함수가 포함된 디렉터리로 <function-dir >을 바꿉니다.

예를 들어 다음과 같습니다.

```
$ kn func deploy --remote \
        --git-url https://example.com/alice/myfunc.git \
        --git-branch my-feature \
        --git-dir functions/example-func/
```

6.4. QUARKUS 함수 개발

Quarkus 함수 프로젝트를 생성한 후에는 제공된 템플릿 파일을 수정하여 비즈니스 로직을 함수에 추가할 수 있습니다. 여기에는 함수 호출 구성 및 반환된 헤더 및 상태 코드가 포함됩니다.

6.4.1. 사전 요구 사항

- 함수를 개발하려면 먼저 OpenShift Serverless Functions 설정에서 설정 단계를 완료해야 합니다.

6.4.2. Quarkus 함수 템플릿 구조

Knative(kn) CLI를 사용하여 Quarkus 함수를 생성할 때 프로젝트 디렉터리는 일반적인 Maven 프로젝트와 유사합니다. 또한 프로젝트에는 함수 구성에 사용되는 func.yaml 파일이 포함되어 있습니다.

http 및 event 트리거 함수 모두 동일한 템플릿 구조를 갖습니다.

템플릿 구조

```
.
├── func.yaml 1
├── mvnw
├── mvnw.cmd
├── pom.xml 2
├── README.md
└── src
    ├── main
    │   ├── java
    │   │   └── functions
    │   │       ├── Function.java 3
    │   │       ├── Input.java
    │   │       └── Output.java
    │   └── resources
    │       └── application.properties
    └── test
        └── java
            └── functions 4
                ├── FunctionTest.java
                └── NativeFunctionIT.java
```

1

이미지 이름과 레지스트리를 결정하는 데 사용됩니다.

2

추가 종속 항목 예

```
...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.assertj</groupId>
      <artifactId>assertj-core</artifactId>
      <version>3.8.0</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
...
```

종속성은 첫 번째 컴파일 중에 다운로드됩니다.

3

함수 프로젝트에는 @Funq 주석이 추가된 Java 메서드가 포함되어야 합니다. 이 메서드를 Function.java 클래스에 배치할 수 있습니다.

4

함수의 로컬 테스트에 사용할 수 있는 간단한 테스트 케이스가 포함되어 있습니다.

6.4.3. Quarkus 함수 호출 정보

클라우드 이벤트에 응답하는 Quarkus 프로젝트 또는 간단한 HTTP 요청에 응답하는 Quarkus 프로젝트를 생성할 수 있습니다. Knative의 클라우드 이벤트는 HTTP를 통해 POST 요청으로 전송되므로 두 기능 유형 모두 들어오는 HTTP 요청을 수신하고 응답할 수 있습니다.

들어오는 요청이 수신되면 Quarkus 함수가 허용된 유형의 인스턴스와 함께 호출됩니다.

표 6.1. 함수 호출 옵션

호출 메소드	인스턴스에 포함된 데이터 유형	데이터 예
HTTP POST 요청	요청 본문에 있는 JSON 오브젝트	**{ "customerId": "0123456", "productId": "6543210" }**
HTTP GET 요청	쿼리 문자열의 데이터	**?customerId=0123456&productId=6543210**
CloudEvent	**data** 속성의 JSON 개체	**{ "customerId": "0123456", "productId": "6543210" }**

*다음 예제에서는 이전 표에 나열된 **customerId** 및 **productId** 구매 데이터를 수신하고 처리하는 함수를 보여줍니다.*

***Quarkus** 함수의 예*

```
public class Functions {
    @Funq
    public void processPurchase(Purchase purchase) {
        // process the purchase
    }
}
```

*구매 데이터를 포함하는 **Purchase JavaBean** 클래스는 다음과 같습니다.*

클래스 예

```
public class Purchase {
    private long customerId;
    private long productId;
    // getters and setters
}
```

6.4.3.1. 호출 예

다음 예제 코드는 withBeans, withCloudEvent, withBinary라는 세 가지 함수를 정의합니다.

예제

```
import io.quarkus.funqy.Funq;
import io.quarkus.funqy.knative.events.CloudEvent;

public class Input {
    private String message;

    // getters and setters
}

public class Output {
    private String message;

    // getters and setters
}

public class Functions {
    @Funq
    public Output withBeans(Input in) {
        // function body
    }

    @Funq
    public CloudEvent<Output> withCloudEvent(CloudEvent<Input> in) {
        // function body
    }

    @Funq
    public void withBinary(byte[] in) {
        // function body
    }
}
```

Functions 클래스의 withBeans 함수는 다음을 통해 호출할 수 있습니다.

- JSON 본문이 있는 HTTP POST 요청:

```
$ curl "http://localhost:8080/withBeans" -X POST \
    -H "Content-Type: application/json" \
    -d '{"message": "Hello there."}'
```

- *쿼리 매개변수가 있는 **HTTP GET** 요청:*

```
$ curl "http://localhost:8080/withBeans?message=Hello%20there." -X GET
```

- *바이너리 인코딩의 **CloudEvent** 오브젝트:*

```
$ curl "http://localhost:8080/" -X POST \
  -H "Content-Type: application/json" \
  -H "Ce-SpecVersion: 1.0" \
  -H "Ce-Type: withBeans" \
  -H "Ce-Source: cURL" \
  -H "Ce-Id: 42" \
  -d '{"message": "Hello there."}'
```

- *구조화된 인코딩의 **CloudEvent** 오브젝트:*

```
$ curl http://localhost:8080/ \
    -H "Content-Type: application/cloudevents+json" \
    -d '{ "data": {"message":"Hello there."},
          "datacontenttype": "application/json",
          "id": "42",
          "source": "curl",
          "type": "withBeans",
          "specversion": "1.0"}'
```

***Functions** 클래스의 **withCloudEvent** 함수는 **withBeans** 함수와 유사하게 **CloudEvent** 오브젝트를 사용하여 호출할 수 있습니다. 그러나 **withBeans**와 달리**withCloudEvent**는 일반 **HTTP** 요청으로 호출할 수 없습니다.*

***Functions** 클래스의 **withBinary** 함수는 다음을 통해 호출할 수 있습니다.*

- *바이너리 인코딩의 **CloudEvent** 오브젝트:*

```
$ curl "http://localhost:8080/" -X POST \
  -H "Content-Type: application/octet-stream" \
  -H "Ce-SpecVersion: 1.0"\
  -H "Ce-Type: withBinary" \
  -H "Ce-Source: cURL" \
  -H "Ce-Id: 42" \
  --data-binary '@img.jpg'
```

- 구조화된 인코딩의 **CloudEvent** 오브젝트:

```
$ curl http://localhost:8080/ \
  -H "Content-Type: application/cloudevents+json" \
  -d "{ \"data_base64\": \"$(base64 --wrap=0 img.jpg)\",
       \"datacontenttype\": \"application/octet-stream\",
       \"id\": \"42\",
       \"source\": \"curl\",
       \"type\": \"withBinary\",
       \"specversion\": \"1.0\"}"
```

6.4.4. CloudEvent 속성

type 또는 **subject**와 같은 **CloudEvent**의 속성을 읽거나 작성해야 하는 경우 **CloudEvent<T>** 일반 인터페이스와 **CloudEventBuilder** 빌더를 사용할 수 있습니다. **<T>** 유형 매개변수는 허용된 유형 중 하나여야 합니다.

다음 예에서 **CloudEventBuilder**는 구매 처리 성공 또는 실패를 반환하는 데 사용됩니다.

```
public class Functions {

    private boolean _processPurchase(Purchase purchase) {
        // do stuff
    }

    public CloudEvent<Void> processPurchase(CloudEvent<Purchase> purchaseEvent) {
        System.out.println("subject is: " + purchaseEvent.subject());

        if (!_processPurchase(purchaseEvent.data())) {
            return CloudEventBuilder.create()
                    .type("purchase.error")
                    .build();
        }
        return CloudEventBuilder.create()
                .type("purchase.success")
                .build();
    }
}
```

6.4.5. Quarkus 함수 반환 값

함수는 허용된 유형 목록에서 모든 유형의 인스턴스를 반환할 수 있습니다. 또는 **< T>** 유형 매개변수가 허용된 유형의 모든 유형일 수 있는 **Uni < T >** 유형을 반환할 수 있습니다.

Uni<T> 유형은 반환된 오브젝트가 수신된 오브젝트와 동일한 형식으로 직렬화되기 때문에 함수가 비

동기 API를 호출할 때 유용합니다. 예를 들어 다음과 같습니다.

- 함수가 HTTP 요청을 수신하면 반환된 오브젝트가 HTTP 응답 본문에 전송됩니다.
- 함수가 바이너리 인코딩으로 CloudEvent 오브젝트를 수신하는 경우 반환된 오브젝트는 바이너리 인코딩 CloudEvent 오브젝트의 데이터 속성으로 전송됩니다.

다음 예제에서는 구매 목록을 가져오는 함수를 보여줍니다.

명령 예

```
public class Functions {
  @Funq
  public List<Purchase> getPurchasesByName(String name) {
    // logic to retrieve purchases
  }
}
```

- HTTP 요청을 통해 이 함수를 호출하면 응답 본문에서 구매한 목록을 포함하는 HTTP 응답이 생성됩니다.
- 들어오는 CloudEvent 오브젝트를 통해 이 함수를 호출하면 data 속성에서 구매 목록이 포함된 CloudEvent 응답이 생성됩니다.

6.4.5.1. 허용된 유형

함수의 입력 및 출력은 void,String 또는 byte[] 유형 중 하나일 수 있습니다. 또한 기본 유형 및 해당 래퍼(예: int 및 Integer)일 수 있습니다. Javabeans, maps, lists, arrays, special CloudEvents<T > 유형의 복잡한 오브젝트일 수도 있습니다.

맵, 목록, 배열, CloudEvents< T > 유형의 <T > 유형 매개변수 및 Javabeans의 속성은 여기에 나열된 유형만 사용할 수 있습니다.

예제

```
public class Functions {
    public List<Integer> getIds();
    public Purchase[] getPurchasesByName(String name);
    public String getNameById(int id);
    public Map<String,Integer> getNameIdMapping();
    public void processImage(byte[] img);
}
```

6.4.6. Quarkus 함수 테스트

Quarkus 함수는 컴퓨터에서 로컬로 테스트할 수 있습니다. kn func create 를 사용하여 함수를 생성할 때 생성되는 기본 프로젝트에는 basic Maven 테스트가 포함된labs /test/ 디렉터리가 있습니다. 이러한 테스트는 필요에 따라 확장할 수 있습니다.

사전 요구 사항

- Quarkus 함수를 생성했습니다.
- Knative(kn) CLI가 설치되어 있습니다.

절차

1. 함수의 프로젝트 폴더로 이동합니다.Navigate to the project folder for your function.
2. Maven 테스트를 실행합니다.

```
$ ./mvnw test
```

6.4.7. 다음 단계

- 함수 를 빌드하고 배포합니다.

6.5. NODE.JS 함수 개발

Node.js 함수 프로젝트를 생성한 후에는 제공된 템플릿 파일을 수정하여 비즈니스 로직을 함수에 추가할 수 있습니다. 여기에는 함수 호출 구성 및 반환된 헤더 및 상태 코드가 포함됩니다.

6.5.1. 사전 요구 사항

- 함수를 개발하려면 먼저 OpenShift Serverless Functions 설정 단계를 완료해야 합니다.

6.5.2. Node.js 함수 템플릿 구조

Knative(kn) CLI를 사용하여 Node.js 함수를 생성하면 프로젝트 디렉터리는 일반적인 Node.js 프로젝트와 같습니다. 유일한 예외는 함수를 구성하는 데 사용되는 추가 func.yaml 파일입니다.

http 및 event 트리거 함수 모두 동일한 템플릿 구조를 갖습니다.

템플릿 구조

```
.
├── func.yaml 1
├── index.js 2
├── package.json 3
├── README.md
└── test 4
    ├── integration.js
    └── unit.js
```

1 func.yaml 구성 파일은 이미지 이름과 레지스트리를 결정하는 데 사용됩니다.

2

3 템플릿 package.json 파일에 제공된 종속성으로 제한되지 않습니다. 다른 Node.js 프로젝트에서와 마찬가지로 추가 종속 항목을 추가할 수 있습니다.

npm 종속성 추가 예

```
npm install --save opossum
```

프로젝트가 배포용으로 빌드되면 이러한 종속 항목은 생성된 런타임 컨테이너 이미지에 포함됩니다.

4

통합 및 테스트 스크립트는 함수 템플릿의 일부로 제공됩니다.

6.5.3. Node.js 함수 호출 정보

Knative(kn) CLI를 사용하여 함수 프로젝트를 생성할 때 CloudEvents에 응답하는 프로젝트 또는 간단한 HTTP 요청에 응답하는 프로젝트를 생성할 수 있습니다. Knative의 CloudEvents는 HTTP를 통해 POST 요청으로 전송되므로 함수 유형 모두 수신되는 HTTP 이벤트를 수신하고 응답합니다.

Node.js 함수는 간단한 HTTP 요청을 사용하여 호출할 수 있습니다. 들어오는 요청이 수신되면 context 오브젝트를 첫 번째 매개 변수로 사용하여 함수가 호출됩니다.

6.5.3.1. Node.js 컨텍스트 오브젝트

함수는 context 오브젝트를 첫 번째 매개 변수로 제공하여 호출됩니다. 이 오브젝트는 들어오는 HTTP 요청 정보에 대한 액세스를 제공합니다.

컨텍스트 오브젝트의 예

```
function handle(context, data)
```

이 정보에는 HTTP 요청 메서드, 요청 문자열 또는 요청과 함께 전송된 쿼리 문자열 또는 헤더, 요청 본문이 포함됩니다. CloudEvent 가 포함된 들어오는 요청은 context.cloudevent 를 사용하여 액세스할 수

있도록 CloudEvent의 들어오는 인스턴스를 컨텍스트 오브젝트에 연결합니다.

6.5.3.1.1. 컨텍스트 오브젝트 메서드

context 오브젝트에는 데이터 값을 수락하고 CloudEvent를 반환하는 단일 메서드 cloudEventResponse()가 있습니다.

Knative 시스템에서 서비스로 배포된 함수가 CloudEvent를 보내는 이벤트 브로커에 의해 호출되는 경우 브로커는 응답을 확인합니다. 응답이 CloudEvent인 경우 브로커가 이 이벤트를 처리합니다.

컨텍스트 오브젝트 메서드 예

```
// Expects to receive a CloudEvent with customer data
function handle(context, customer) {
  // process the customer
  const processed = handle(customer);
  return context.cloudEventResponse(customer)
    .source('/handle')
    .type('fn.process.customer')
    .response();
}
```

6.5.3.1.2. CloudEvent 데이터

들어오는 요청이 CloudEvent인 경우 CloudEvent와 관련된 모든 데이터가 이벤트에서 추출되며 두 번째 매개변수로 제공됩니다. 예를 들어 데이터 속성에 다음과 유사한 JSON 문자열이 포함된 CloudEvent가 수신되는 경우 다음과 같이 됩니다.

```
{
  "customerId": "0123456",
  "productId": "6543210"
}
```

호출될 때 context 오브젝트 다음에 함수에 대한 두 번째 매개 변수는 customerId 및 productId 속성이 있는 JavaScript 오브젝트가 됩니다.

서명 예

```
function handle(context, data)
```

이 예제의 data 매개변수는 customerId 및 productId 속성을 포함하는 JavaScript 오브젝트입니다.

6.5.4. Node.js 함수 반환 값

함수는 유효한 JavaScript 유형을 반환하거나 반환 값이 없을 수 있습니다. 함수에 반환 값이 지정되지 않고 실패가 표시되지 않으면 호출자는 204 No Content 응답을 받습니다.

또한 함수는 이벤트를 Knative Eventing 시스템으로 푸시하기 위해 CloudEvent 또는 Message 오브젝트를 반환할 수 있습니다. 이 경우 개발자는 CloudEvent 메시징 사양을 이해하고 구현할 필요가 없습니다. 반환된 값의 헤더 및 기타 관련 정보는 추출된 응답으로 전송됩니다.

예

```
function handle(context, customer) {
  // process customer and return a new CloudEvent
  return new CloudEvent({
    source: 'customer.processor',
    type: 'customer.processed'
  })
}
```

6.5.4.1. 헤더 반환

return 오브젝트에 headers 속성을 추가하여 응답 헤더를 설정할 수 있습니다. 이러한 헤더는 추출된 호출에 대한 응답으로 전송됩니다.

응답 헤더의 예

```
function handle(context, customer) {
  // process customer and return custom headers
  // the response will be '204 No content'
```

```
  return { headers: { customerid: customer.id } };
}
```

6.5.4.2. 상태 코드 반환

statusCode 속성을 **return** 오브젝트에 추가하여 호출자에게 반환된 상태 코드를 설정할 수 있습니다.

상태 코드 예

```
function handle(context, customer) {
  // process customer
  if (customer.restricted) {
    return { statusCode: 451 }
  }
}
```

상태 코드는 함수에서 생성되어 발생하는 오류에 대해 설정할 수 있습니다.

오류 상태 코드의 예

```
function handle(context, customer) {
  // process customer
  if (customer.restricted) {
    const err = new Error('Unavailable for legal reasons');
    err.statusCode = 451;
    throw err;
  }
}
```

6.5.5. Node.js 함수 테스트

Node.js 함수는 컴퓨터에서 로컬로 테스트할 수 있습니다. **kn func create** 를 사용하여 함수를 생성할

때 생성되는 기본 프로젝트에는 몇 가지 간단한 단위 및 통합 테스트 가 포함된 테스트 폴더가 있습니다.

사전 요구 사항

- ***OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다.*
- ***Knative(kn) CLI**가 설치되어 있습니다.*
- ***kn func create** 를 사용하여 함수를 생성했습니다.*

절차

1. *함수의 테스트 폴더로 이동합니다.**Navigate to the test folder for your function.***
2. *테스트를 실행합니다.*

```
$ npm test
```

6.5.6. 다음 단계

- ***Node.js** 컨텍스트 오브젝트 참조 설명서를 참조하십시오.*
- *함수 를 빌드하고 배포합니다.*

6.6. TYPESCRIPT 함수 개발

***TypeScript** 함수 프로젝트를 생성한 후에는 제공된 템플릿 파일을 수정하여 비즈니스 로직을 기능에 추가할 수 있습니다. 여기에는 함수 호출 구성 및 반환된 헤더 및 상태 코드가 포함됩니다.*

6.6.1. 사전 요구 사항

- *함수를 개발하려면 먼저 **OpenShift Serverless Functions** 설정 단계를 완료해야 합니다.*

6.6.2. TypeScript 함수 템플릿 구조

Knative(kn) CLI를 사용하여 TypeScript 함수를 생성할 때 프로젝트 디렉터리는 일반적인 TypeScript 프로젝트처럼 보입니다. 유일한 예외는 함수 구성에 사용되는 추가 func.yaml 파일입니다.

http 및 event 트리거 함수 모두 동일한 템플릿 구조를 갖습니다.

템플릿 구조

```
.
├── func.yaml 1
├── package.json 2
├── package-lock.json
├── README.md
├── src
│   └── index.ts 3
├── test 4
│   ├── integration.ts
│   └── unit.ts
└── tsconfig.json
```

1

func.yaml 구성 파일은 이미지 이름과 레지스트리를 결정하는 데 사용됩니다.

2

템플릿 package.json 파일에 제공된 종속성으로 제한되지 않습니다. 다른 TypeScript 프로젝트에서와 마찬가지로 종속 항목을 추가할 수 있습니다.

npm 종속성 추가 예

```
npm install --save opossum
```

프로젝트가 배포용으로 빌드되면 이러한 종속 항목은 생성된 런타임 컨테이너 이미지에 포함됩니다.

3

프로젝트에는 **handle**라는 함수를 내보내는 **src/index.js** 파일이 포함되어야 합니다.

4

통합 및 테스트 스크립트는 함수 템플릿의 일부로 제공됩니다.

6.6.3. TypeScript 함수 호출 정보

Knative(kn) CLI를 사용하여 함수 프로젝트를 생성할 때 **CloudEvents** 또는 간단한 **HTTP** 요청에 응답하는 프로젝트를 생성할 수 있습니다. **Knative**의 **CloudEvents**는 **HTTP**를 통해 **POST** 요청으로 전송되므로 함수 유형 모두 수신되는 **HTTP** 이벤트를 수신하고 응답합니다.

간단한 **HTTP** 요청을 사용하여 **TypeScript** 함수를 호출할 수 있습니다. 들어오는 요청이 수신되면 **context** 오브젝트를 첫 번째 매개 변수로 사용하여 함수가 호출됩니다.

6.6.3.1. TypeScript 컨텍스트 오브젝트

함수를 호출하려면 **context** 오브젝트를 첫 번째 매개 변수로 제공합니다. 컨텍스트 오브젝트의 속성에 액세스하면 들어오는 **HTTP** 요청에 대한 정보를 제공할 수 있습니다.

컨텍스트 오브젝트의 예

```
function handle(context:Context): string
```

이 정보에는 **HTTP** 요청 메서드, 요청 문자열 또는 요청과 함께 전송된 쿼리 문자열 또는 헤더, 요청 본문이 포함됩니다. **CloudEvent** 가 포함된 들어오는 요청은 **context.cloudevent** 를 사용하여 액세스할 수 있도록 **CloudEvent**의 들어오는 인스턴스를 컨텍스트 오브젝트에 연결합니다.

6.6.3.1.1. 컨텍스트 오브젝트 메서드

context 오브젝트에는 데이터 값을 수락하고 **CloudEvent**를 반환하는 단일 메서드 **cloudEventResponse()**가 있습니다.

Knative 시스템에서 서비스로 배포된 함수가 CloudEvent를 보내는 이벤트 브로커에 의해 호출되는 경우 브로커는 응답을 확인합니다. 응답이 CloudEvent인 경우 브로커가 이 이벤트를 처리합니다.

컨텍스트 오브젝트 메서드 예

```
// Expects to receive a CloudEvent with customer data
export function handle(context: Context, cloudevent?: CloudEvent): CloudEvent {
  // process the customer
  const customer = cloudevent.data;
  const processed = processCustomer(customer);
  return context.cloudEventResponse(customer)
    .source('/customer/process')
    .type('customer.processed')
    .response();
}
```

6.6.3.1.2. 컨텍스트 유형

TypeScript 유형 정의 파일은 함수에 사용하기 위해 다음 유형을 내보냅니다.

내보낸 유형 정의

```
// Invokable is the expeted Function signature for user functions
export interface Invokable {
    (context: Context, cloudevent?: CloudEvent): any
}

// Logger can be used for structural logging to the console
export interface Logger {
  debug: (msg: any) => void,
  info:  (msg: any) => void,
  warn:  (msg: any) => void,
  error: (msg: any) => void,
  fatal: (msg: any) => void,
  trace: (msg: any) => void,
}

// Context represents the function invocation context, and provides
// access to the event itself as well as raw HTTP objects.
export interface Context {
    log: Logger;
    req: IncomingMessage;
```

```
    query?: Record<string, any>;
    body?: Record<string, any>|string;
    method: string;
    headers: IncomingHttpHeaders;
    httpVersion: string;
    httpVersionMajor: number;
    httpVersionMinor: number;
    cloudevent: CloudEvent;
    cloudEventResponse(data: string|object): CloudEventResponse;
}

// CloudEventResponse is a convenience class used to create
// CloudEvents on function returns
export interface CloudEventResponse {
    id(id: string): CloudEventResponse;
    source(source: string): CloudEventResponse;
    type(type: string): CloudEventResponse;
    version(version: string): CloudEventResponse;
    response(): CloudEvent;
}
```

6.6.3.1.3. CloudEvent 데이터

들어오는 요청이 **CloudEvent**인 경우 **CloudEvent**와 관련된 모든 데이터가 이벤트에서 추출되며 두 번째 매개변수로 제공됩니다. 예를 들어 데이터 속성에 다음과 유사한 **JSON** 문자열이 포함된 **CloudEvent**가 수신되는 경우 다음과 같이 됩니다.

```
{
  "customerId": "0123456",
  "productId": "6543210"
}
```

호출될 때 **context** 오브젝트 다음에 함수에 대한 두 번째 매개 변수는 **customerId** 및 **productId** 속성이 있는 **JavaScript** 오브젝트가 됩니다.

서명 예

```
function handle(context: Context, cloudevent?: CloudEvent): CloudEvent
```

이 예제의 cloudevent 매개 변수는 customerId 및 productId 속성이 포함된 JavaScript 오브젝트입니다.

6.6.4. TypeScript 함수 반환 값

함수는 유효한 JavaScript 유형을 반환하거나 반환 값이 없을 수 있습니다. 함수에 반환 값이 지정되지 않고 실패가 표시되지 않으면 호출자는 204 No Content 응답을 받습니다.

또한 함수는 이벤트를 Knative Eventing 시스템으로 푸시하기 위해 CloudEvent 또는 Message 오브젝트를 반환할 수 있습니다. 이 경우 개발자는 CloudEvent 메시징 사양을 이해하고 구현할 필요가 없습니다. 반환된 값의 헤더 및 기타 관련 정보는 추출된 응답으로 전송됩니다.

예

```
export const handle: Invokable = function (
  context: Context,
  cloudevent?: CloudEvent
): Message {
  // process customer and return a new CloudEvent
  const customer = cloudevent.data;
  return HTTP.binary(
    new CloudEvent({
      source: 'customer.processor',
      type: 'customer.processed'
    })
  );
};
```

6.6.4.1. 헤더 반환

return 오브젝트에 headers 속성을 추가하여 응답 헤더를 설정할 수 있습니다. 이러한 헤더는 추출된 호출에 대한 응답으로 전송됩니다.

응답 헤더의 예

```
export function handle(context: Context, cloudevent?: CloudEvent): Record<string, any> {
  // process customer and return custom headers
  const customer = cloudevent.data as Record<string, any>;
```

```
  return { headers: { 'customer-id': customer.id } };
}
```

6.6.4.2. 상태 코드 반환

statusCode 속성을 return 오브젝트에 추가하여 호출자에게 반환된 상태 코드를 설정할 수 있습니다.

상태 코드 예

```
export function handle(context: Context, cloudevent?: CloudEvent): Record<string, any> {
  // process customer
  const customer = cloudevent.data as Record<string, any>;
  if (customer.restricted) {
    return {
      statusCode: 451
    }
  }
  // business logic, then
  return {
    statusCode: 240
  }
}
```

상태 코드는 함수에서 생성되어 발생하는 오류에 대해 설정할 수 있습니다.

오류 상태 코드의 예

```
export function handle(context: Context, cloudevent?: CloudEvent): Record<string, string> {
  // process customer
  const customer = cloudevent.data as Record<string, any>;
  if (customer.restricted) {
    const err = new Error('Unavailable for legal reasons');
    err.statusCode = 451;
    throw err;
  }
}
```

6.6.5. TypeScript 함수 테스트

TypeScript 함수는 컴퓨터에서 로컬에서 테스트할 수 있습니다. kn func create를 사용하여 함수를 만들 때 생성되는 기본 프로젝트에는 몇 가지 간단한 단위 및 통합 테스트가 포함된 테스트 폴더가 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- Knative(kn) CLI가 설치되어 있습니다.
- kn func create 를 사용하여 함수를 생성했습니다.

절차

1. 이전에 테스트를 실행하지 않은 경우 먼저 종속성을 설치합니다.

```
$ npm install
```

2. 함수의 테스트 폴더로 이동합니다.Navigate to the test folder for your function.

3. 테스트를 실행합니다.

```
$ npm test
```

6.6.6. 다음 단계

- TypeScript 컨텍스트 오브젝트 참조 설명서를 참조하십시오.
- 함수 를 빌드하고 배포합니다.

- 함수 로깅에 대한 자세한 내용은 Pino API 설명서를 참조하십시오.

6.7. PYTHON 함수 개발

중요

OpenShift Serverless Functions with Python은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

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

Python 함수 프로젝트를 생성한 후에는 제공된 템플릿 파일을 수정하여 비즈니스 로직을 함수에 추가할 수 있습니다. 여기에는 함수 호출 구성 및 반환된 헤더 및 상태 코드가 포함됩니다.

6.7.1. 사전 요구 사항

- 함수를 개발하려면 먼저 OpenShift Serverless Functions 설정 단계를 완료해야 합니다.

6.7.2. Python 함수 템플릿 구조

Knative(kn) CLI를 사용하여 Python 함수를 생성할 때 프로젝트 디렉터리는 일반적인 Python 프로젝트와 유사합니다. Python 함수는 약간의 제한 사항이 있습니다. 유일한 요구 사항은 프로젝트에 main() 함수와 func.yaml 구성 파일이 포함된 func.py 파일이 포함되어 있다는 것입니다.

개발자는 템플릿 requirements.txt 파일에 제공된 종속성으로 제한되지 않습니다. 추가 종속 항목은 다른 Python 프로젝트에서 추가될 수 있습니다. 프로젝트가 배포용으로 빌드되면 이러한 종속성이 생성된 런타임 컨테이너 이미지에 포함됩니다.

http 및 event 트리거 함수 모두 동일한 템플릿 구조를 갖습니다.

템플릿 구조

```
fn
├── func.py 1
├── func.yaml 2
├── requirements.txt 3
└── test_func.py 4
```

1

main() 함수를 포함합니다.

2

이미지 이름과 레지스트리를 결정하는 데 사용됩니다.

3

다른 Python 프로젝트에 있는 것처럼 requirements.txt 파일에 기타 종속 항목을 추가할 수 있습니다.

4

함수의 로컬 테스트에 사용할 수 있는 간단한 단위 테스트가 포함되어 있습니다.

6.7.3. Python 함수 호출 정보

Python 함수는 간단한 HTTP 요청으로 호출할 수 있습니다. 들어오는 요청이 수신되면 context 오브젝트를 첫 번째 매개 변수로 사용하여 함수가 호출됩니다.

context 오브젝트는 두 개의 속성이 있는 Python 클래스입니다.

- *request 속성은 항상 존재하며 Flask request 오브젝트를 포함합니다.*
- *들어오는 요청이 CloudEvent 오브젝트인 경우 두 번째 속성 cloud_event가 채워집니다.*

개발자는 컨텍스트 오브젝트에서 모든 CloudEvent 데이터에 액세스할 수 있습니다.

컨텍스트 오브젝트의 예

```
def main(context: Context):
    """
    The context parameter contains the Flask request object and any
    CloudEvent received with the request.
    """
    print(f"Method: {context.request.method}")
    print(f"Event data {context.cloud_event.data}")
    # ... business logic here
```

6.7.4. Python 함수 반환 값

함수는 Flask 에서 지원하는 모든 값을 반환할 수 있습니다. 호출 프레임워크가 이러한 값을 Flask 서버에 직접 프록시하기 때문입니다.

예

```
def main(context: Context):
    body = { "message": "Howdy!" }
    headers = { "content-type": "application/json" }
    return body, 200, headers
```

함수는 함수 호출에서 헤더와 응답 코드를 모두 2차 및 3차 응답 값으로 설정할 수 있습니다.

6.7.4.1. CloudEvents 반환

개발자는 @event 데코레이터를 사용하여 응답을 보내기 전에 함수 반환 값을 CloudEvent로 변환해야 함을 호출자에게 알릴 수 있습니다.

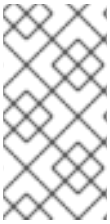
예

```
@event("event_source"="/my/function", "event_type"="my.type")
def main(context):
    # business logic here
    data = do_something()
    # more data processing
    return data
```

이 예제에서는 "my.type" 유형과 "/my/function" 소스를 사용하여 CloudEvent를 응답 값으로 보냅니다. CloudEvent data 속성은 반환된 data 변수로 설정됩니다. event_source 및 event_type 데코레이터 속성은 선택 사항입니다.

6.7.5. Python 함수 테스트

컴퓨터에서 Python 함수를 로컬로 테스트할 수 있습니다. 기본 프로젝트에는 기능에 대한 간단한 단위 테스트를 제공하는 test_proxyc.py 파일이 포함되어 있습니다.



참고

Python 함수의 기본 테스트 프레임워크는 unittest입니다. 필요에 따라 다른 테스트 프레임워크를 사용할 수 있습니다.

사전 요구 사항

- Python 함수 테스트를 로컬에서 실행하려면 필요한 종속 항목을 설치해야 합니다.

```
$ pip install -r requirements.txt
```

절차

1. test_func.py 파일이 포함된 함수의 폴더로 이동합니다.
2. 테스트를 실행합니다.

```
$ python3 test_func.py
```

6.7.6. 다음 단계

- 함수 를 빌드하고 배포합니다.

6.8. KNATIVE EVENTING에서 함수 사용

함수는 OpenShift Container Platform 클러스터에 Knative 서비스로 배포됩니다. 들어오는 이벤트를 수신할 수 있도록 함수를 Knative Eventing 구성 요소에 연결할 수 있습니다.

6.8.1. 개발자 화면을 사용하여 이벤트 소스를 함수에 연결

함수는 OpenShift Container Platform 클러스터에 Knative 서비스로 배포됩니다. OpenShift Container Platform 웹 콘솔을 사용하여 이벤트 소스를 생성할 때 해당 소스에서 이벤트가 전송되는 배포된 함수를 지정할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Serving, Knative Eventing이 OpenShift Container Platform 클러스터에 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 개발자 화면으로 갑니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 함수를 생성하고 배포했습니다.

절차

1. +추가 → 이벤트 소스로 이동하여 모든 유형의 이벤트 소스를 생성하고 생성할 이벤트 소스 유형을 선택합니다.
2. 이벤트 소스 생성 양식 보기의 싱크 섹션에서 리소스 목록에서 함수를 선택합니다.
3. 생성을 클릭합니다.

검증

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

1. 개발자 화면에서 토폴로지로 이동합니다.

2. 이벤트 소스를 보고 연결된 함수를 클릭하여 오른쪽 패널에서 기능 세부 정보를 확인합니다.

6.9. FUNC.YAML의 함수 프로젝트 구성

func.yaml 파일에는 함수 프로젝트의 구성이 포함되어 있습니다. **func.yaml** 에 지정된 값은 **kn func** 명령을 실행할 때 사용됩니다. 예를 들어 **kn func build** 명령을 실행하면 빌드 필드의 값이 사용됩니다. 경우에 따라 명령줄 플래그 또는 환경 변수로 이러한 값을 덮어쓸 수 있습니다.

6.9.1. func.yaml의 구성 가능한 필드

func.yaml의 대부분의 필드는 함수를 생성, 빌드 및 배포할 때 자동으로 생성됩니다. 그러나 함수 이름 또는 이미지 이름과 같은 사항을 변경하기 위해 수동으로 변경하는 필드도 있습니다.

6.9.1.1. buildEnvs

buildEnvs 필드를 사용하면 함수를 빌드하는 환경에서 사용할 수 있도록 환경 변수를 설정할 수 있습니다. **envs** 를 사용하여 설정된 변수와 달리 **buildEnv** 를 사용한 변수는 함수 런타임 중에 사용할 수 없습니다.

값에서 **buildEnv** 변수를 직접 설정할 수 있습니다. 다음 예에서 **EXAMPLE1** 이라는 **buildEnv** 변수에는 하나의 값이 직접 할당됩니다.

```
buildEnvs:
- name: EXAMPLE1
  value: one
```

로컬 환경 변수에서 **buildEnv** 변수를 설정할 수도 있습니다. 다음 예에서 **EXAMPLE2** 라는 **buildEnv** 변수에는 **LOCAL_ENV_VAR** 로컬 환경 변수의 값이 할당됩니다.

```
buildEnvs:
- name: EXAMPLE1
  value: '{{ env:LOCAL_ENV_VAR }}'
```

6.9.1.2. envs

envs 필드를 사용하면 런타임 시 사용할 수 있는 환경 변수를 설정할 수 있습니다. 환경 변수를 여러 가지 방법으로 설정할 수 있습니다.

1. 값을 직접 설정할 수 있습니다.
2. 로컬 환경 변수에 할당 된 값에서 설정합니다. 자세한 내용은 **func.yaml** 필드에서 로컬 환경 변수 참조 섹션을 참조하십시오.
3. 시크릿 또는 구성 맵에 저장된 키-값 쌍에서 설정합니다.
4. 생성된 환경 변수의 이름으로 사용되는 키를 사용하여 시크릿 또는 구성 맵에 저장된 모든 키-값 쌍을 가져올 수도 있습니다.

이 예제에서는 환경 변수를 설정하는 다양한 방법을 보여줍니다.

```
name: test
namespace: ""
runtime: go
...
envs:
- name: EXAMPLE1 (1)
  value: value
- name: EXAMPLE2 (2)
  value: '{{ env:LOCAL_ENV_VALUE }}'
- name: EXAMPLE3 (3)
  value: '{{ secret:mysecret:key }}'
- name: EXAMPLE4 (4)
  value: '{{ configMap:myconfigmap:key }}'
- value: '{{ secret:mysecret2 }}' (5)
- value: '{{ configMap:myconfigmap2 }}' (6)
```

(1)

(2)

로컬 환경 변수에 할당된 값에서 설정된 환경 변수.

(3)

4

구성 맵에 저장된 키-값 쌍에서 할당된 환경 변수.

5

시크릿의 키-값 쌍에서 가져온 환경 변수 세트.

6

구성 맵의 키-값 쌍에서 가져온 환경 변수 세트.

6.9.1.3. builder

builder 필드는 함수에서 이미지를 빌드하는 데 사용하는 전략을 지정합니다. 이는 **pack** 또는 **s2i** 의 값을 허용합니다.

6.9.1.4. Build

빌드 필드는 함수를 빌드하는 방법을 나타냅니다. 값 **local** 은 함수가 시스템에서 로컬로 빌드되었음을 나타냅니다. 값 **git** 은 **git** 필드에 지정된 값을 사용하여 함수가 클러스터에 빌드되었음을 나타냅니다.

6.9.1.5. volumes

volumes 필드를 사용하면 다음 예와 같이 지정된 경로에서 기능에 액세스할 수 있는 볼륨으로 시크릿 및 구성 맵을 마운트할 수 있습니다.

```
name: test
namespace: ""
runtime: go
...
volumes:
- secret: mysecret (1)
  path: /workspace/secret
- configMap: myconfigmap (2)
  path: /workspace/configmap
```

1

mysecret 시크릿은 **/workspace/secret**에 있는 볼륨으로 마운트됩니다.

2

6.9.1.6. options

options 필드를 사용하면 자동 스케일링과 같이 배포된 함수에 대한 **Knative Service** 속성을 수정할 수 있습니다. 이러한 옵션이 설정되어 있지 않으면 기본 옵션이 사용됩니다.

다음 옵션을 사용할 수 있습니다.

- **scale**
  - **min:** 최소 복제 수입니다. 음수가 아닌 정수여야 합니다. 기본값은 **0**입니다.
  - **max:** 최대 복제본 수입니다. 음수가 아닌 정수여야 합니다. 기본값은 **0**이며 이는 제한이 없음을 의미합니다.
  - **metric:** **Autoscaler**에서 감시하는 메트릭 유형을 정의합니다. 기본값인 **concurrency** 또는 **rps**로 설정할 수 있습니다.
  - **target:** 동시에 수신되는 요청 수에 따라 버전을 확장할 시기에 대한 권장 사항을 제공합니다. **target** 옵션은 **0.01**보다 큰 부동 소수점 값을 지정할 수 있습니다. **options.resources.limits.concurrency**가 설정되지 않는 한 기본값은 **100**입니다. 이 경우 **target**은 기본값으로 설정됩니다.
  - **utilization:** 스케일 업하기 전에 허용된 동시 요청 사용률(백분율)입니다. **1**에서 **100**까지의 부동 소수점 값을 지정할 수 있습니다. 기본값은 **70**입니다.
- **resources**
  - **requests**
    - **cpu:** 배포된 함수가 있는 컨테이너에 대한 **CPU** 리소스 요청입니다.

- **memory:** 배포된 함수가 있는 컨테이너에 대한 메모리 리소스 요청입니다.

- 제한
  - **cpu:** 배포된 함수가 있는 컨테이너의 **CPU** 리소스 제한입니다.
  - **memory:** 배포된 함수가 있는 컨테이너의 메모리 리소스 제한입니다.
  - **concurrency:** 단일 복제본에 의해 처리되는 동시 요청 수에 대한 하드 제한입니다. **0**보다 크거나 같은 정수 값이 될 수 있습니다. 기본값은 **0**이며 이는 제한이 없음을 의미합니다.

다음은 **scale** 옵션 구성의 예입니다.

```
name: test
namespace: ""
runtime: go
...
options:
  scale:
    min: 0
    max: 10
    metric: concurrency
    target: 75
    utilization: 75
  resources:
    requests:
      cpu: 100m
      memory: 128Mi
    limits:
      cpu: 1000m
      memory: 256Mi
      concurrency: 100
```

6.9.1.7. image

image 필드는 해당 함수의 이미지 이름을 빌드한 후 설정합니다. 이 필드는 필요에 따라 수정할 수 있습니다. 이 경우 다음에 **kn func build** 또는 **kn func deploy**를 실행하면 함수 이미지가 새 이름으로 생성됩니다.

6.9.1.8. imageDigest

imageDigest 필드에는 함수가 배포될 때 이미지 매니페스트의 SHA256 해시가 포함됩니다. 이 값은 변경하지 마십시오.

6.9.1.9. labels

labels 필드를 사용하면 배포된 기능에 라벨을 설정할 수 있습니다.

값에서 직접 레이블을 설정할 수 있습니다. 다음 예에서는 role 키가 있는 라벨에 backend 값이 직접 할당됩니다.

```
labels:
- key: role
  value: backend
```

로컬 환경 변수에서 레이블을 설정할 수도 있습니다. 다음 예에서는 author 키가 있는 라벨에 USER 로컬 환경 변수의 값이 할당됩니다.

```
labels:
- key: author
  value: '{{ env:USER }}'
```

6.9.1.10. name

name 필드는 함수의 이름을 정의합니다. 이 값은 배포 시 Knative 서비스의 이름으로 사용됩니다. 이 필드를 변경하여 후속 배포의 함수 이름을 변경할 수 있습니다.

6.9.1.11. namespace

namespace 필드는 함수가 배포되는 네임스페이스를 지정합니다.

6.9.1.12. runtime

runtime 필드는 기능에 대한 언어 런타임(예: python)을 지정합니다.

6.9.2. func.yaml 필드의 로컬 환경 변수 참조

함수 구성에 API 키와 같은 중요한 정보를 저장하지 않으려면 로컬 환경에서 사용 가능한 환경 변수에 대한 참조를 추가할 수 있습니다. func.yaml 파일의 envs 필드를 수정하여 이 작업을 수행할 수 있습니다.

사전 요구 사항

- *함수 프로젝트를 만들어야 합니다.*
- *로컬 환경에는 참조하려는 변수가 포함되어야 합니다.*

절차

- *로컬 환경 변수를 참조하려면 다음 구문을 사용합니다.*

```
{{ env:ENV_VAR }}
```

  ***ENV_VAR**을 사용하려는 로컬 환경의 변수 이름으로 바꿉니다.*

  *예를 들어 로컬 환경에서 사용할 수 있는 **API_KEY** 변수가 있을 수 있습니다. **MY_API_KEY** 변수에 해당 값을 할당하면 함수 내에서 직접 사용할 수 있습니다.*

  함수 예

```
name: test
namespace: ""
runtime: go
...
envs:
- name: MY_API_KEY
  value: '{{ env:API_KEY }}'
...
```

6.9.3. 추가 리소스

- *함수 시작하기*

- *Serverless 함수에서 시크릿 및 구성 맵에 액세스*
- *자동 확장에 대한 Knative 문서*
- *컨테이너 리소스 관리에 대한 Kubernetes 설명서*
- *동시성 구성에 대한 Knative 설명서*

6.10. 함수에서 시크릿 및 구성 맵에 액세스

함수가 클러스터에 배포된 후 시크릿 및 구성 맵에 저장된 데이터에 액세스할 수 있습니다. 이 데이터는 볼륨으로 마운트하거나 환경 변수에 할당할 수 있습니다. Knative CLI를 사용하거나 함수 구성 YAML 파일을 수동으로 편집하여 이 액세스를 대화식으로 구성할 수 있습니다.

중요

시크릿 및 구성 맵에 액세스하려면 함수를 클러스터에 배포해야 합니다. 이 기능은 로컬에서 실행되는 함수에 사용할 수 없습니다.

시크릿 또는 구성 맵 값에 액세스할 수 없는 경우 액세스할 수 없는 값을 지정하는 오류 메시지와 함께 배포가 실패합니다.

6.10.1. 시크릿 및 구성 맵에 대한 함수 액세스의 상호 작용 변경

kn func config 대화형 유틸리티를 사용하여 함수에서 액세스하는 시크릿 및 구성 맵을 관리할 수 있습니다. 사용 가능한 작업에는 구성 맵 및 시크릿에 저장된 값 목록, 추가, 제거, 볼륨 나열, 추가, 제거 등이 포함됩니다. 이 기능을 사용하면 함수에서 액세스할 수 있는 클러스터에 저장된 데이터를 관리할 수 있습니다.

사전 요구 사항

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

- 함수를 생성했습니다.

절차

1. 함수 프로젝트에서 다음 명령을 실행합니다.

```
$ kn func config
```

또는 **--path** 또는 **-p** 옵션을 사용하여 함수 프로젝트 디렉터리를 지정할 수 있습니다.

2. 대화형 인터페이스를 사용하여 필요한 작업을 수행합니다. 예를 들어 유틸리티를 사용하여 구성된 볼륨을 나열하면 다음과 유사한 출력이 생성됩니다.

```
$ kn func config
? What do you want to configure? Volumes
? What operation do you want to perform? List
Configured Volumes mounts:
- Secret "mysecret" mounted at path: "/workspace/secret"
- Secret "mysecret2" mounted at path: "/workspace/secret2"
```

이 스키마는 대화형 유틸리티에서 사용할 수 있는 모든 작업과 해당 유틸리티로 이동하는 방법을 보여줍니다.

```
kn func config
  ├─> Environment variables
  │             ├─> Add
  │             │    ├─> ConfigMap: Add all key-value pairs from a config map
  │             │    ├─> ConfigMap: Add value from a key in a config map
  │             │    ├─> Secret: Add all key-value pairs from a secret
  │             │    └─> Secret: Add value from a key in a secret
  │             ├─> List: List all configured environment variables
  │             └─> Remove: Remove a configured environment variable
  └─> Volumes
        ├─> Add
        │    ├─> ConfigMap: Mount a config map as a volume
        │    └─> Secret: Mount a secret as a volume
        ├─> List: List all configured volumes
        └─> Remove: Remove a configured volume
```

3. 선택 사항: 함수를 배포하여 변경 사항을 적용합니다.

```
$ kn func deploy -p test
```

6.10.2. 특수 명령을 사용하여 시크릿 및 구성 맵에 대한 함수 액세스 수정

kn func config 유틸리티를 실행할 때마다 전체 대화 상자를 탐색하여 이전 섹션에서와 같이 필요한 작업을 선택해야 합니다. 단계를 저장하려면 **kn func config** 명령 보다 구체적인 양식을 실행하여 특정 작업을 직접 수행합니다.

- 구성된 환경 변수를 나열하려면 다음을 수행합니다.

```
$ kn func config envs [-p <function-project-path>]
```

- 함수 구성에 환경 변수를 추가하려면 다음을 수행합니다.

```
$ kn func config envs add [-p <function-project-path>]
```

- 함수 구성에서 환경 변수를 제거하려면 다음을 수행합니다.

```
$ kn func config envs remove [-p <function-project-path>]
```

- 구성된 볼륨을 나열하려면 다음을 수행합니다.

```
$ kn func config volumes [-p <function-project-path>]
```

- 함수 구성에 볼륨을 추가하려면 다음을 수행합니다.

```
$ kn func config volumes add [-p <function-project-path>]
```

- 함수 구성에서 볼륨을 제거하려면 다음을 수행합니다.

```
$ kn func config volumes remove [-p <function-project-path>]
```

6.10.3. 시크릿 및 구성 맵에 수동으로 함수 액세스 추가

시크릿 및 구성 맵에 액세스하는 구성을 함수에 수동으로 추가할 수 있습니다. **kn func config** 대화형 유틸리티 및 명령을 사용하는 것이 좋습니다(예: 기존 구성 스니펫이 있는 경우).

6.10.3.1. 시크릿을 볼륨으로 마운트

시크릿을 볼륨으로 마운트할 수 있습니다. 시크릿이 마운트되면 함수에서 일반 파일로 액세스할 수 있습니다. 이를 통해 함수에 필요한 클러스터 데이터(예: 함수에서 액세스해야 하는 URI 목록)에 저장할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- Knative(kn) CLI가 설치되어 있습니다.
- 함수를 생성했습니다.

절차

1. 함수에 사용할 func.yaml 파일을 엽니다.
2. 볼륨으로 마운트하려는 각 시크릿에 대해 volumes 섹션에 다음 YAML을 추가합니다.

```
name: test
namespace: ""
runtime: go
...
volumes:
- secret: mysecret
  path: /workspace/secret
```

- mysecret을 대상 시크릿의 이름으로 대체합니다.
- 시크릿을 마운트하려는 경로로 /workspace/secret을 대체합니다.

예를 들어 주소 시크릿을 마운트하려면 다음 YAML을 사용합니다.

```
name: test
namespace: ""
runtime: go
...
```

```
volumes:
- configMap: addresses
  path: /workspace/secret-addresses
```

3. 설정을 저장합니다.

6.10.3.2. 구성 맵을 볼륨으로 마운트

구성 맵을 볼륨으로 마운트할 수 있습니다. 구성 맵이 마운트되면 함수에서 일반 파일로 액세스할 수 있습니다. 이를 통해 함수에 필요한 클러스터 데이터(예: 함수에서 액세스해야 하는 **URI** 목록)에 저장할 수 있습니다.

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다.
- **Knative(kn) CLI**가 설치되어 있습니다.
- 함수를 생성했습니다.

절차

1. 함수에 사용할 **func.yaml** 파일을 엽니다.
2. 볼륨으로 마운트하려는 각 구성 맵에 대해 **volumes** 섹션에 다음 **YAML**을 추가합니다.

```
name: test
namespace: ""
runtime: go
...
volumes:
- configMap: myconfigmap
  path: /workspace/configmap
```

- **myconfigmap**을 대상 구성 맵의 이름으로 대체합니다.
- **/workspace/configmap**을 구성 맵을 마운트하려는 경로로 바꿉니다.

예를 들어 주소 구성 맵을 마운트하려면 다음 YAML을 사용합니다.

```
name: test
namespace: ""
runtime: go
...
volumes:
- configMap: addresses
  path: /workspace/configmap-addresses
```

3. 설정을 저장합니다.

6.10.3.3. 시크릿에 정의된 키 값에서 환경 변수 설정

보안으로 정의된 키 값에서 환경 변수를 설정할 수 있습니다. 그런 다음 이전에 시크릿에 저장된 값에 런타임 시 함수를 통해 환경 변수로 액세스할 수 있습니다. 이는 사용자의 ID와 같이 시크릿에 저장된 값에 대한 액세스 권한을 얻는 데 유용할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- Knative(kn) CLI가 설치되어 있습니다.
- 함수를 생성했습니다.

절차

1. 함수에 사용할 func.yaml 파일을 엽니다.
2. 환경 변수에 할당할 시크릿 키-값 쌍의 각 값에 대해 envs 섹션에 다음 YAML을 추가합니다.

```
name: test
namespace: ""
runtime: go
...
envs:
- name: EXAMPLE
  value: '{{ secret:mysecret:key }}'
```

- *EXAMPLE*을 환경 변수 이름으로 대체합니다.
- *mysecret*을 대상 시크릿의 이름으로 대체합니다.
- *key*를 대상 값에 매핑된 키로 대체합니다.

예를 들어 사용자 *detailssecret*에 저장된 사용자 *ID*에 액세스하려면 다음 *YAML*을 사용합니다.

```
name: test
namespace: ""
runtime: go
...
envs:
- value: '{{ configMap:userdetailssecret:userid }}'
```

3. 설정을 저장합니다.

6.10.3.4. 구성 맵에 정의된 키 값에서 환경 변수 설정

구성 맵으로 정의된 키 값에서 환경 변수를 설정할 수 있습니다. 그런 다음 이전에 구성 맵에 저장된 값에 런타임 시 함수를 통해 환경 변수로 액세스할 수 있습니다. 이는 사용자의 *ID*와 같이 구성 맵에 저장된 값에 대한 액세스 권한을 얻는 데 유용할 수 있습니다.

사전 요구 사항

- *OpenShift Serverless Operator* 및 *Knative Serving*이 클러스터에 설치되어 있습니다.
- *Knative(kn) CLI*가 설치되어 있습니다.
- 함수를 생성했습니다.

절차

1. 함수에 사용할 *func.yaml* 파일을 엽니다.

2. 환경 변수에 할당할 구성 맵 키-값 쌍의 각 값에 대해 **envs** 섹션에 다음 **YAML**을 추가합니다.

```
name: test
namespace: ""
runtime: go
...
envs:
- name: EXAMPLE
  value: '{{ configMap:myconfigmap:key }}'
```

- **EXAMPLE**을 환경 변수 이름으로 대체합니다.
- **myconfigmap**을 대상 구성 맵의 이름으로 대체합니다.
- **key**를 대상 값에 매핑된 키로 대체합니다.

예를 들어 **userdetailsmap** 에 저장된 사용자 **ID**에 액세스하려면 다음 **YAML**을 사용합니다.

```
name: test
namespace: ""
runtime: go
...
envs:
- value: '{{ configMap:userdetailsmap:userid }}'
```

3. 설정을 저장합니다.

6.10.3.5. 시크릿에 정의된 모든 값에서 환경 변수 설정

시크릿에 정의된 모든 값에서 환경 변수를 설정할 수 있습니다. 그런 다음 이전에 시크릿에 저장된 값에 런타임 시 함수를 통해 환경 변수로 액세스할 수 있습니다. 이는 비밀에 저장된 값 컬렉션에 동시에 액세스 권한을 얻는 데 유용할 수 있습니다(예: 사용자와 관련된 데이터 집합).

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다.

- Knative(kn) CLI가 설치되어 있습니다.
- 함수를 생성했습니다.

절차

1. 함수에 사용할 func.yaml 파일을 엽니다.
2. 모든 키-값 쌍을 환경 변수로 가져오려는 모든 시크릿에 다음 YAML을 envs 섹션에 추가합니다.

```
name: test
namespace: ""
runtime: go
...
envs:
- value: '{{ secret:mysecret }}' (1)
```

(1) mysecret을 대상 시크릿의 이름으로 대체합니다.

예를 들어 사용자 detailssecret에 저장된 모든 사용자 데이터에 액세스하려면 다음 YAML을 사용합니다.

```
name: test
namespace: ""
runtime: go
...
envs:
- value: '{{ configMap:userdetailssecret }}'
```

3. 설정을 저장합니다.

6.10.3.6. 구성 맵에 정의된 모든 값에서 환경 변수 설정

구성 맵에 정의된 모든 값에서 환경 변수를 설정할 수 있습니다. 그런 다음 이전에 구성 맵에 저장된 값에 런타임 시 함수를 통해 환경 변수로 액세스할 수 있습니다. 이는 구성 맵에 저장된 값 컬렉션에 동시에 액세스 권한을 얻는 데 유용할 수 있습니다(예: 사용자와 관련된 데이터 집합).

사전 요구 사항

- *OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.*
- *Knative(kn) CLI가 설치되어 있습니다.*
- *함수를 생성했습니다.*

절차

1. *함수에 사용할 func.yaml 파일을 엽니다.*
2. *모든 키-값 쌍을 환경 변수로 가져오려는 모든 구성 맵에 대해 envs 섹션에 다음 YAML을 추가합니다.*

```
name: test
namespace: ""
runtime: go
...
envs:
- value: '{{ configMap:myconfigmap }}' (1)
```

(1) *myconfigmap을 대상 구성 맵의 이름으로 대체합니다.*

예를 들어 userdetailsmap 에 저장된 모든 사용자 데이터에 액세스하려면 다음 YAML을 사용합니다.

```
name: test
namespace: ""
runtime: go
...
envs:
- value: '{{ configMap:userdetailsmap }}'
```

3. *파일을 저장합니다.*

6.11. 함수에 주석 추가

배포된 **Serverless** 함수에 **Kubernetes** 주석을 추가할 수 있습니다. 주석을 사용하면 임의의 메타데이터를 함수에 연결할 수 있습니다(예: 함수의 목적에 대한 참고). 주석은 **func.yaml** 구성 파일의 **annotations** 섹션에 추가됩니다.

함수 주석 기능에는 다음 두 가지 제한 사항이 있습니다.

- 함수 주석이 클러스터의 해당 **Knative** 서비스로 전파되면 **func.yaml** 파일에서 삭제하여 서비스에서 제거할 수 없습니다. 서비스의 **YAML** 파일을 직접 수정하거나 **OpenShift Container Platform** 웹 콘솔을 사용하여 **Knative** 서비스에서 주석을 제거해야 합니다.
- **Knative**에서 설정한 주석(예: **autoscaling** 주석)을 설정할 수 없습니다.

6.11.1. 함수에 주석 추가

함수에 주석을 추가할 수 있습니다. 레이블과 유사하게 주석은 키-값 맵으로 정의됩니다. 예를 들어 주석은 함수 작성자와 같은 함수에 대한 메타데이터를 제공하는 데 유용합니다.

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다.
- **Knative(kn) CLI**가 설치되어 있습니다.
- 함수를 생성했습니다.

절차

1. 함수에 사용할 **func.yaml** 파일을 엽니다.
2. 추가할 모든 주석에 대해 **annotations** 섹션에 다음 **YAML**을 추가합니다.

```
name: test
namespace: ""
runtime: go
```

```
...
annotations:
  <annotation_name>: "<annotation_value>" (1)
```

(1) <annotation_name>: "<annotation_value>"를 주석으로 바꿉니다.

예를 들어 Alice에서 함수가 생성되었음을 나타내기 위해 다음 주석을 포함할 수 있습니다.

```
name: test
namespace: ""
runtime: go
...
annotations:
  author: "alice@example.com"
```

3. 설정을 저장합니다.

다음에 함수를 클러스터에 배포할 때 해당 Knative 서비스에 주석이 추가됩니다.

6.12. 함수 개발 참조 가이드

OpenShift Serverless Functions는 기본 기능을 생성하는 데 사용할 수 있는 템플릿을 제공합니다. 템플릿은 기능 프로젝트 상용구를 시작하고 kn func 툴과 함께 사용하도록 준비합니다. 각 함수 템플릿은 특정 런타임에 맞게 조정되며 해당 규칙을 따릅니다. 템플릿을 사용하면 함수 프로젝트를 자동으로 시작할 수 있습니다.

다음 런타임에 대한 템플릿을 사용할 수 있습니다.

- Node.js
- Quarkus
- TypeScript

6.12.1. Node.js 컨텍스트 오브젝트 참조

context 오브젝트에는 함수 개발자가 액세스할 수 있는 여러 속성이 있습니다. 이러한 속성에 액세스하면 HTTP 요청에 대한 정보를 제공하고 클러스터 로그에 출력을 쓸 수 있습니다.

6.12.1.1. log

클러스터 로그에 출력을 작성하는 데 사용할 수 있는 로깅 오브젝트를 제공합니다. 로그는 Pino 로깅 API를 따릅니다.

로그 예

```
function handle(context) {
  context.log.info("Processing customer");
}
```

kn func invoke 명령을 사용하여 함수에 액세스할 수 있습니다.

명령 예

```
$ kn func invoke --target 'http://example.function.com'
```

출력 예

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":
1,"msg":"Processing customer"}
```

로그 수준을 fatal,error,warn,info,debug,trace 또는 silent 중 하나로 변경할 수 있습니다. 이렇게 하려면 config 명령을 사용하여 해당 값 중 하나를 환경 변수 FujiNC _LOG_LEVEL에 할당하여 logLevel

값을 변경합니다.

6.12.1.2. query

요청에 대한 쿼리 문자열을 키-값 쌍으로 반환합니다. 이러한 속성은 컨텍스트 오브젝트 자체에서도 확인할 수 있습니다.

예제 쿼리

```
function handle(context) {
  // Log the 'name' query parameter
  context.log.info(context.query.name);
  // Query parameters are also attached to the context
  context.log.info(context.name);
}
```

kn func invoke 명령을 사용하여 함수에 액세스할 수 있습니다.

명령 예

```
$ kn func invoke --target 'http://example.com?name=tiger'
```

출력 예

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":
1,"msg":"tiger"}
```

6.12.1.3. body

필요한 경우 요청 본문을 반환합니다. 요청 본문에 JSON 코드가 포함된 경우 속성을 직접 사용할 수 있도록 구문 분석됩니다.

본문의 예

```
function handle(context) {
  // log the incoming request body's 'hello' parameter
  context.log.info(context.body.hello);
}
```

curl 명령을 사용하여 이를 호출하여 함수에 액세스할 수 있습니다.

명령 예

```
$ kn func invoke -d '{"Hello": "world"}'
```

출력 예

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":
1,"msg":"world"}
```

6.12.1.4. headers

HTTP 요청 헤더를 오브젝트로 반환합니다.

헤더 예

```
function handle(context) {
  context.log.info(context.headers["custom-header"]);
}
```

kn func invoke 명령을 사용하여 함수에 액세스할 수 있습니다.

명령 예

```
$ kn func invoke --target 'http://example.function.com'
```

출력 예

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":
1,"msg":"some-value"}
```

6.12.1.5. HTTP 요청

method

HTTP 요청 메서드를 문자열로 반환합니다.

httpVersion

HTTP 버전을 문자열로 반환합니다.

httpVersionMajor

HTTP 주요 버전 번호를 문자열로 반환합니다.

httpVersionMinor

HTTP 마이너 버전 번호를 문자열로 반환합니다.

6.12.2. TypeScript 컨텍스트 오브젝트 참조

context 오브젝트에는 함수 개발자가 액세스할 수 있는 여러 속성이 있습니다. 이러한 속성에 액세스하면 들어오는 **HTTP** 요청에 대한 정보를 제공하고 클러스터 로그에 출력을 쓸 수 있습니다.

6.12.2.1. log

클러스터 로그에 출력을 작성하는 데 사용할 수 있는 로깅 오브젝트를 제공합니다. 로그는 **Pino** 로깅 **API**를 따릅니다.

로그 예

```
export function handle(context: Context): string {
  // log the incoming request body's 'hello' parameter
  if (context.body) {
    context.log.info((context.body as Record<string, string>).hello);
  } else {
    context.log.info('No data received');
  }
  return 'OK';
}
```

kn func invoke 명령을 사용하여 함수에 액세스할 수 있습니다.

명령 예

```
$ kn func invoke --target 'http://example.function.com'
```

출력 예

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":
1,"msg":"Processing customer"}
```

로그 수준을 fatal,error,warn,info,debug,trace 또는 silent 중 하나로 변경할 수 있습니다. 이렇게 하려면 config 명령을 사용하여 해당 값 중 하나를 환경 변수 FujiNC _LOG_LEVEL에 할당하여 logLevel 값을 변경합니다.

6.12.2.2. query

요청에 대한 쿼리 문자열을 키-값 쌍으로 반환합니다. 이러한 속성은 컨텍스트 오브젝트 자체에서도 확인할 수 있습니다.

예제 쿼리

```
export function handle(context: Context): string {
    // log the 'name' query parameter
  if (context.query) {
    context.log.info((context.query as Record<string, string>).name);
  } else {
    context.log.info('No data received');
  }
  return 'OK';
}
```

kn func invoke 명령을 사용하여 함수에 액세스할 수 있습니다.

명령 예

```
$ kn func invoke --target 'http://example.function.com' --data '{"name": "tiger"}'
```

출력 예

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":
1,"msg":"tiger"}
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":
1,"msg":"tiger"}
```

6.12.2.3. body

요청 본문(있는 경우)을 반환합니다. 요청 본문에 JSON 코드가 포함된 경우 속성을 직접 사용할 수 있도록 구문 분석됩니다.

본문의 예

```
export function handle(context: Context): string {
  // log the incoming request body's 'hello' parameter
  if (context.body) {
    context.log.info((context.body as Record<string, string>).hello);
  } else {
    context.log.info('No data received');
  }
  return 'OK';
}
```

kn func invoke 명령을 사용하여 함수에 액세스할 수 있습니다.

명령 예

```
$ kn func invoke --target 'http://example.function.com' --data '{"hello": "world"}'
```

출력 예

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":
1,"msg":"world"}
```

6.12.2.4. headers

HTTP 요청 헤더를 오브젝트로 반환합니다.

헤더 예

```
export function handle(context: Context): string {
  // log the incoming request body's 'hello' parameter
  if (context.body) {
    context.log.info((context.headers as Record<string, string>)['custom-header']);
  } else {
    context.log.info('No data received');
  }
  return 'OK';
}
```

curl 명령을 사용하여 이를 호출하여 함수에 액세스할 수 있습니다.

명령 예

```
$ curl -H'x-custom-header: some-value'' http://example.function.com
```

출력 예

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":
1,"msg":"some-value"}
```

6.12.2.5. HTTP 요청

method

HTTP 요청 메서드를 문자열로 반환합니다.

httpVersion

HTTP 버전을 문자열로 반환합니다.

httpVersionMajor

HTTP 주요 버전 번호를 문자열로 반환합니다.

httpVersionMinor

HTTP 마이너 버전 번호를 문자열로 반환합니다.

7장. KNATIVE CLI

7.1. KNATIVE SERVING CLI 명령

7.1.1. kn service 명령

다음 명령을 사용하여 Knative 서비스를 생성하고 관리할 수 있습니다.

7.1.1.1. Knative CLI를 사용하여 서버리스 애플리케이션 생성

Knative(kn) CLI를 사용하여 서버리스 애플리케이션을 생성하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다. kn service create 명령을 사용하여 기본 서버리스 애플리케이션을 생성할 수 있습니다.

사전 요구 사항

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

절차

- Knative 서비스를 생성합니다.

```
$ kn service create <service-name> --image <image> --tag <tag-value>
```

  다음과 같습니다.

  - --image 는 애플리케이션의 이미지 URI입니다.
  - --tag 는 서비스로 생성된 초기 버전에 태그를 추가하는 데 사용할 수 있는 선택적 플래그입니다.

명령 예

```
$ kn service create event-display \
   --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

출력 예

```
Creating service 'event-display' in namespace 'default':

 0.271s The Route is still working to reflect the latest desired specification.
 0.580s Configuration "event-display" is waiting for a Revision to become ready.
 3.857s ...
 3.861s Ingress has not yet been reconciled.
 4.270s Ready to serve.

Service 'event-display' created with latest revision 'event-display-bxshg-1' and
URL:
http://event-display-default.apps-crc.testing
```

7.1.1.2. Knative CLI를 사용하여 서버리스 애플리케이션 업데이트

서비스를 단계적으로 구축할 때 명령줄에서 대화형 세션에 **kn service update** 명령을 사용할 수 있습니다. **kn service apply** 명령과 달리 **kn service update** 명령을 사용하는 경우 **Knative** 서비스의 전체 구성이 아닌 업데이트하려는 변경 사항만 지정해야 합니다.

명령 예

- 새 환경 변수를 추가하여 서비스를 업데이트합니다.

```
$ kn service update <service_name> --env <key>=<value>
```

- 새 포트를 추가하여 서비스를 업데이트합니다.

```
$ kn service update <service_name> --port 80
```

- 새 요청 및 제한 매개변수를 추가하여 서비스를 업데이트합니다.

```
$ kn service update <service_name> --request cpu=500m --limit memory=1024Mi --
limit cpu=1000m
```

- latest 태그를 개정 버전에 할당합니다.

```
$ kn service update <service_name> --tag <revision_name>=latest
```

- 서비스의 최신 **READY** 버전에 대한 태그를 **testing**에서 **staging**으로 업데이트합니다.

```
$ kn service update <service_name> --untag testing --tag @latest=staging
```

- 트래픽의 **10%**를 수신하는 버전에 **test** 태그를 추가하고 나머지 트래픽을 서비스의 최신 **READY** 버전으로 전송합니다.

```
$ kn service update <service_name> --tag <revision_name>=test --traffic
test=10,@latest=90
```

7.1.1.3. 서비스 선언 적용

kn service apply 명령을 사용하여 **Knative** 서비스를 선언적으로 구성할 수 있습니다. 서비스가 존재하지 않으면 기존 서비스가 변경된 옵션으로 업데이트됩니다.

kn service apply 명령은 사용자가 일반적으로 단일 명령으로 서비스 상태를 완전히 지정하여 대상 상태를 선언하려는 셸 스크립트 또는 지속적 통합 파이프라인에 특히 유용합니다.

kn service apply를 사용하는 경우 **Knative** 서비스에 대한 전체 구성을 제공해야 합니다. 이 동작은 업데이트하려는 옵션을 명령에서 지정하기만 하면 되는 **kn service update** 명령과 다릅니다.

명령 예

- 서비스를 생성합니다.

```
$ kn service apply <service_name> --image <image>
```

- 서비스에 환경 변수를 추가합니다.

```
$ kn service apply <service_name> --image <image> --env <key>=<value>
```

- JSON 또는 YAML 파일에서 서비스 선언을 읽습니다.

```
$ kn service apply <service_name> -f <filename>
```

7.1.1.4. Knative CLI를 사용하여 서버리스 애플리케이션 설명

kn service describe 명령을 사용하여 Knative 서비스를 설명할 수 있습니다.

명령 예

- 서비스를 설명합니다.

```
$ kn service describe --verbose <service_name>
```

--verbose 플래그는 선택 사항이지만 자세한 설명을 제공하기 위해 포함할 수 있습니다. 일반 출력과 자세한 출력의 차이점은 다음 예에 표시됩니다.

--verbose 플래그를 사용하지 않는 출력 예

```
Name:       hello
Namespace:  default
Age:        2m
URL:        http://hello-default.apps.ocp.example.com

Revisions:
  100%  @latest (hello-00001) [1] (2m)
        Image:  docker.io/openshift/hello-openshift (pinned to aaea76)

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                 1m
  ++ ConfigurationsReady     1m
  ++ RoutesReady           1m
```

--verbose 플래그를 사용하는 출력 예

```
Name:       hello
Namespace:  default
Annotations:  serving.knative.dev/creator=system:admin
              serving.knative.dev/lastModifier=system:admin
Age:        3m
URL:        http://hello-default.apps.ocp.example.com
Cluster:    http://hello.default.svc.cluster.local

Revisions:
  100%  @latest (hello-00001) [1] (3m)
        Image:  docker.io/openshift/hello-openshift (pinned to aaea76)
        Env:    RESPONSE=Hello Serverless!

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                3m
  ++ ConfigurationsReady  3m
  ++ RoutesReady          3m
```

- *YAML 형식으로 서비스를 설명합니다.*

```
$ kn service describe <service_name> -o yaml
```

- *JSON 형식으로 서비스를 설명합니다.*

```
$ kn service describe <service_name> -o json
```

- *서비스 URL만 인쇄합니다.*

```
$ kn service describe <service_name> -o url
```

7.1.2. kn service 명령 오프라인 모드

7.1.2.1. Knative CLI 오프라인 모드 정보

kn service 명령을 실행하면 변경 사항이 즉시 클러스터로 전파됩니다. 그러나 대안으로 오프라인 모드에서 kn service 명령을 실행할 수 있습니다. 오프라인 모드에서 서비스를 생성하면 클러스터에서 변경

사항이 발생하지 않으며, 대신 로컬 시스템에 서비스 설명자 파일이 생성됩니다.

중요

Knative CLI의 오프라인 모드는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

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

설명자 파일이 생성되면 수동으로 수정하고 버전 제어 시스템에서 추적할 수 있습니다. 설명자 파일에서 **kn service create -f**,**kn service apply -f** 또는 **oc apply -f** 명령을 사용하여 변경 사항을 클러스터에 전파할 수도 있습니다.

오프라인 모드에는 여러 방식이 있습니다.

- 클러스터 변경에 사용하기 전에 설명자 파일을 수동으로 수정할 수 있습니다.
- 버전 관리 시스템에서 서비스의 설명자 파일을 로컬로 추적할 수 있습니다. 이를 통해 **CI**(지속적 통합) 파이프라인, 개발 환경 또는 데모와 같이 대상 클러스터 이외의 위치에서 설명자 파일을 재사용할 수 있습니다.
- 생성된 설명자 파일을 검사하여 **Knative** 서비스에 대해 확인할 수 있습니다. 특히 결과 서비스가 **kn** 명령에 전달된 다양한 인수에 영향을 받는 방법을 확인할 수 있습니다.

오프라인 모드는 속도가 빠르고 클러스터에 연결할 필요가 없다는 장점이 있습니다. 그러나 오프라인 모드에서는 서버 측 유효성 검사가 없습니다. 따라서 서비스 이름이 고유하거나 지정된 이미지를 가져올 수 있는지 등을 확인할 수 없습니다.

7.1.2.2. 오프라인 모드를 사용하여 서비스 생성

오프라인 모드에서 **kn service** 명령을 실행할 수 있으므로 클러스터에서 변경 사항이 발생하지 않고 로컬 시스템에서 서비스 설명자 파일이 생성됩니다. 설명자 파일이 생성되면 클러스터에 대한 변경 사항

을 전파하기 전에 파일을 수정할 수 있습니다.

중요

***Knative CLI**의 오프라인 모드는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.*

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

사전 요구 사항

- ***OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다.*
- ***Knative(kn) CLI**가 설치되어 있습니다.*

절차

1. *오프라인 모드에서 로컬 **Knative** 서비스 설명자 파일을 생성합니다.*

```
$ kn service create event-display \
  --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest \
  --target ./ \
  --namespace test
```

출력 예

```
Service 'event-display' created in namespace 'test'.
```

- ***--target ./** 플래그는 오프라인 모드를 활성화하고 새 디렉터리 트리를 저장하는 디렉토리로./를 지정합니다.*

*기존 디렉터리를 지정하지 않고 **--target my-service.yaml**과 같은 파일 이름을 사용하면 디렉터리 트리가 생성되지 않습니다. 대신 서비스 설명자 파일 **my-service.yaml**만 현재 디렉터리에 생성됩니다.*

*파일 이름에는. **.yaml, .yml,** 또는 **.json** 확장을 사용할 수 있습니다. **.json**을 선택하면 **JSON** 형식으로 서비스 설명자 파일을 생성합니다.*

- *--namespace test 옵션은 새 서비스를 **test** 네임스페이스에 배치합니다.*

*--namespace를 사용하지 않고 **OpenShift Container Platform** 클러스터에 로그인한 경우 현재 네임스페이스에 설명자 파일이 생성됩니다. 그렇지 않으면 설명자 파일이 **default** 네임스페이스에 생성됩니다.*

2. *생성된 디렉터리 구조를 확인합니다.*

```
$ tree ./
```

출력 예

```
./
└── test
    └── ksvc
        └── event-display.yaml

2 directories, 1 file
```

- *--target에서 지정된 현재 ./ 디렉터리에는 지정된 네임스페이스를 바탕으로 이름이 지정된 **test/** 디렉터리가 포함되어 있습니다.*
- *test/ 디렉터리에는 리소스 유형의 이름에 따라 이름이 지정된 **ksvc** 디렉터리가 포함되어 있습니다.*
- *ksvc 디렉터리에는 지정된 서비스 이름에 따라 이름이 지정된 기술자 파일 **event-display.yaml**이 포함되어 있습니다.*

3. 생성된 서비스 기술자 파일을 확인합니다.

```
$ cat test/ksvc/event-display.yaml
```

출력 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  creationTimestamp: null
  name: event-display
  namespace: test
spec:
  template:
    metadata:
      annotations:
        client.knative.dev/user-image: quay.io/openshift-knative/knative-eventing-
sources-event-display:latest
      creationTimestamp: null
    spec:
      containers:
      - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
        name: ""
        resources: {}
status: {}
```

4. 새 서비스에 대한 정보를 나열합니다.

```
$ kn service describe event-display --target ./ --namespace test
```

출력 예

```
Name:       event-display
Namespace:  test
Age:
URL:

Revisions:

Conditions:
  OK TYPE    AGE REASON
```

- *target ./ 옵션은 네임스페이스 하위 디렉터리를 포함하는 디렉터리 구조의 루트 디렉터리를 지정합니다.*

  또는 --target 옵션을 사용하여 YAML 또는 JSON 파일 이름을 직접 지정할 수 있습니다. 허용된 파일 확장자는 .yaml, .yml, .json입니다.

- *--namespace 옵션은 필요한 서비스 기술자 파일을 포함하는 하위 디렉터리를 kn와 통신하는 네임스페이스를 지정합니다.*

  --namespace를 사용하지 않고 OpenShift Container Platform 클러스터에 로그인한 경우 kn은 현재 네임스페이스를 바탕으로 이름이 지정된 하위 디렉터리에서 서비스를 검색합니다. 그렇지 않으면 kn은 default/ 하위 디렉터리에서 검색합니다.

5. *서비스 설명자 파일을 사용하여 클러스터에 서비스를 생성합니다.*

```
$ kn service create -f test/ksvc/event-display.yaml
```

출력 예

```
Creating service 'event-display' in namespace 'test':

 0.058s The Route is still working to reflect the latest desired specification.
 0.098s ...
 0.168s Configuration "event-display" is waiting for a Revision to become ready.
23.377s ...
23.419s Ingress has not yet been reconciled.
23.534s Waiting for load balancer to be ready
23.723s Ready to serve.

Service 'event-display' created to latest revision 'event-display-00001' is available at
URL:
http://event-display-test.apps.example.com
```

7.1.3. kn container 명령

다음 명령을 사용하여 Knative 서비스 사양에서 여러 컨테이너를 생성하고 관리할 수 있습니다.

7.1.3.1. Knative 클라이언트 다중 컨테이너 지원

kn container add 명령을 사용하여 YAML 컨테이너 사양을 표준 출력에 인쇄할 수 있습니다. 이 명령은 다른 표준 kn 플래그와 함께 사용하여 정의를 생성할 수 있으므로 멀티 컨테이너 사용 사례에 유용합니다.

kn container add 명령은 kn service create 명령과 함께 사용할 수 있도록 지원되는 모든 컨테이너 관련 플래그를 허용합니다. kn container add 명령은 한 번에 여러 컨테이너 정의를 생성하기 위해 UNIX 파이프(|)를 사용하여 연결할 수도 있습니다.

명령 예

- 이미지에서 컨테이너를 추가하고 표준 출력에 출력합니다.

```
$ kn container add <container_name> --image <image_uri>
```

명령 예

```
$ kn container add sidecar --image docker.io/example/sidecar
```

출력 예

```
containers:
- image: docker.io/example/sidecar
  name: sidecar
  resources: {}
```

- 두 개의 kn container add 명령을 함께 연결한 다음 kn service create 명령에 전달하여 두 개의 컨테이너로 Knative 서비스를 생성합니다.

```
$ kn container add <first_container_name> --image <image_uri> | \
kn container add <second_container_name> --image <image_uri> | \
kn service create <service_name> --image <image_uri> --extra-containers -
```

--extra-containers - kn 이 YAML 파일 대신 파이프 입력을 읽는 특수 사례를 지정합니다.

명령 예

```
$ kn container add sidecar --image docker.io/example/sidecar:first | \
kn container add second --image docker.io/example/sidecar:second | \
kn service create my-service --image docker.io/example/my-app:latest --extra-
containers -
```

--extra-containers 플래그는 YAML 파일의 경로를 허용할 수도 있습니다.

```
$ kn service create <service_name> --image <image_uri> --extra-containers
<filename>
```

명령 예

```
$ kn service create my-service --image docker.io/example/my-app:latest --extra-
containers my-extra-containers.yaml
```

7.1.4. kn domain 명령

다음 명령을 사용하여 도메인 매핑을 생성하고 관리할 수 있습니다.

7.1.4.1. Knative CLI를 사용하여 사용자 정의 도메인 매핑 생성

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.

- Knative 서비스 또는 경로를 생성했으며 CR에 매핑할 사용자 정의 도메인을 제어할 수 있습니다.

참고

사용자 정의 도메인에서 OpenShift Container Platform 클러스터의 DNS를 가리켜야 합니다.

- Knative(kn) CLI가 설치되어 있습니다.

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

절차

- 현재 네임스페이스의 CR에 도메인을 매핑합니다.

```
$ kn domain create <domain_mapping_name> --ref <target_name>
```

명령 예

```
$ kn domain create example-domain-map --ref example-service
```

--ref 플래그는 도메인 매핑을 위해 주소 지정 가능한 대상 CR을 지정합니다.

--ref 플래그를 사용할 때 접두사가 지정되어 있지 않은 경우 대상이 현재 네임스페이스의 Knative 서비스라고 가정합니다.

- 지정된 네임스페이스의 Knative 서비스에 도메인을 매핑합니다.

```
$ kn domain create <domain_mapping_name> --ref
<ksvc:service_name:service_namespace>
```

-

명령 예

```
$ kn domain create example-domain-map --ref ksvc:example-service:example-namespace
```

- 도메인을 Knative 경로에 매핑합니다.

```
$ kn domain create <domain_mapping_name> --ref <kroute:route_name>
```

명령 예

```
$ kn domain create example-domain-map --ref kroute:example-route
```

7.1.4.2. Knative CLI를 사용하여 사용자 정의 도메인 매핑 관리

DomainMapping CR(사용자 정의 리소스)을 생성한 후에는 기존 CR을 나열하고, 기존 CR에 대한 정보를 보거나, CR을 업데이트하거나, Knative(kn) CLI를 사용하여 CR을 삭제할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- 하나 이상의 DomainMapping CR을 생성했습니다.
- Knative(kn) CLI 툴을 설치했습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

절차

- 기존 DomainMapping CR을 나열합니다.

```
$ kn domain list -n <domain_mapping_namespace>
```

- 기존 DomainMapping CR 정보를 표시합니다.

```
$ kn domain describe <domain_mapping_name>
```

- 새 대상을 참조하도록 DomainMapping CR을 업데이트합니다.

```
$ kn domain update --ref <target>
```

- DomainMapping CR을 삭제합니다.

```
$ kn domain delete <domain_mapping_name>
```

7.2. KNATIVE CLI 구성

config.yaml 구성 파일을 생성하여 Knative(kn) CLI 설정을 사용자 지정할 수 있습니다. --config 플래그를 사용하여 이 구성을 지정할 수 있습니다. 지정하지 않으면 기본 위치에서 구성을 가져옵니다. 기본 구성 위치는 XDG Base Directory Specification 을 준수하며 UNIX 시스템 및 Windows 시스템에 따라 다릅니다.

UNIX 시스템의 경우:

- XDG_CONFIG_HOME 환경 변수가 설정된 경우 Knative(kn) CLI에서 찾는 기본 구성 위치는 $XDG_CONFIG_HOME/kn 입니다.
- XDG_CONFIG_HOME 환경 변수가 설정되지 않은 경우 Knative(kn) CLI는 $HOME/.config/kn/config.yaml 에서 사용자의 홈 디렉터리에서 구성을 찾습니다.

Windows 시스템의 경우 기본 Knative(kn) CLI 구성 위치는 %APPDATA%\kn 입니다.

설정 파일 예

```
plugins:
  path-lookup: true (1)
  directory: ~/.config/kn/plugins (2)
eventing:
  sink-mappings: (3)
  - prefix: svc (4)
    group: core (5)
    version: v1 (6)
    resource: services (7)
```

(1) Knative(kn) CLI에서 PATH 환경 변수의 플러그인을 찾을지 여부를 지정합니다. 이는 부울 구성 옵션입니다. 기본값은 false입니다.

(2) Knative(kn) CLI에서 플러그인을 찾는 디렉터리를 지정합니다. 기본 경로는 이전에 설명한 대로 운영 체제에 따라 다릅니다. 이는 사용자에게 표시되는 모든 디렉터리일 수 있습니다.

(3) sink-mappings 사양은 --sink 플래그를 Knative(kn) CLI 명령과 함께 사용할 때 사용되는 Kubernetes 주소 지정 가능 리소스를 정의합니다.

(4) 싱크를 설명하는 데 사용할 접두사입니다. 서비스 svc, channel, broker 는 Knative(kn) CLI의 사전 정의된 접두사입니다.

(5) Kubernetes 리소스의 API 그룹입니다.

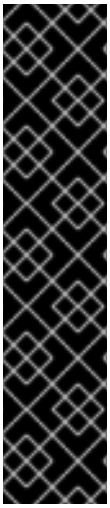
(6) Kubernetes 리소스의 버전입니다.

(7) Kubernetes 리소스 유형의 복수형 이름입니다. 예: services 또는 brokers

7.3. KNATIVE CLI 플러그인

Knative(kn) CLI는 플러그인 사용을 지원하므로 사용자 지정 명령 및 코어 배포에 포함되지 않은 기타 공유 명령을 추가하여 kn 설치 기능을 확장할 수 있습니다. Knative(kn) CLI 플러그인은 기본 kn 기능과 동일한 방식으로 사용됩니다.

현재 Red Hat은 kn-source-kafka 플러그인 및 kn-event 플러그인을 지원합니다.



중요

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

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

7.3.1. kn-event 플러그인을 사용하여 이벤트 빌드

kn event build 명령의 builder-like 인터페이스를 사용하여 이벤트를 빌드할 수 있습니다. 그런 다음 나중에 해당 이벤트를 보내거나 다른 컨텍스트에서 사용할 수 있습니다.

사전 요구 사항

- Knative(kn) CLI가 설치되어 있습니다.

절차

- 이벤트를 빌드합니다.

```
$ kn event build --field <field-name>=<value> --type <type-name> --id <id> --output <format>
```

다음과 같습니다.

  -

--field 플래그는 이벤트에 데이터를 필드-값 쌍으로 추가합니다. 여러 번 사용할 수 있습니다.

- *--type 플래그를 사용하면 이벤트 유형을 지정하는 문자열을 지정할 수 있습니다.*
- *--id 플래그는 이벤트의 ID를 지정합니다.*
- *--output 플래그와 함께 json 또는 yaml 인수를 사용하여 이벤트의 출력 형식을 변경할 수 있습니다.*

이러한 모든 플래그는 선택 사항입니다.

간단한 이벤트 만들기

```
$ kn event build -o yaml
```

YAML 형식의 결과 이벤트

```
data: {}
datacontenttype: application/json
id: 81a402a2-9c29-4c27-b8ed-246a253c9e58
source: kn-event/v0.4.0
specversion: "1.0"
time: "2021-10-15T10:42:57.713226203Z"
type: dev.knative.cli.plugin.event.generic
```

샘플 트랜잭션 이벤트 빌드

```
$ kn event build \
  --field operation.type=local-wire-transfer \
  --field operation.amount=2345.40 \
```

```
    --field operation.from=87656231 \
    --field operation.to=2344121 \
    --field automated=true \
    --field signature='FGzCPLvYWdEgsdpb3qXkaVp7Da0=' \
    --type org.example.bank.bar \
    --id $(head -c 10 < /dev/urandom | base64 -w 0) \
    --output json
```

JSON 형식의 결과 이벤트

```
{
  "specversion": "1.0",
  "id": "RjtL8UH66X+UJg==",
  "source": "kn-event/v0.4.0",
  "type": "org.example.bank.bar",
  "datacontenttype": "application/json",
  "time": "2021-10-15T10:43:23.113187943Z",
  "data": {
    "automated": true,
    "operation": {
      "amount": "2345.40",
      "from": 87656231,
      "to": 2344121,
      "type": "local-wire-transfer"
    },
    "signature": "FGzCPLvYWdEgsdpb3qXkaVp7Da0="
  }
}
```

7.3.2. kn-event 플러그인을 사용하여 이벤트 전송

kn event send 명령을 사용하여 이벤트를 보낼 수 있습니다. 이벤트는 공개적으로 사용 가능한 주소 또는 **Kubernetes** 서비스, **Knative** 서비스, 브로커 및 채널과 같은 클러스터 내에서 주소 지정 가능한 리소스로 보낼 수 있습니다. 이 명령은 **kn event build** 명령과 동일한 **builder-like** 인터페이스를 사용합니다.

사전 요구 사항

- **Knative(kn) CLI**가 설치되어 있습니다.

설자

- 이벤트 보내기:

```
$ kn event send --field <field-name>=<value> --type <type-name> --id <id> --to-url
<url> --to <cluster-resource> --namespace <namespace>
```

다음과 같습니다.

  - **--field** 플래그는 이벤트에 데이터를 필드-값 쌍으로 추가합니다. 여러 번 사용할 수 있습니다.
  - **--type** 플래그를 사용하면 이벤트 유형을 지정하는 문자열을 지정할 수 있습니다.
  - **--id** 플래그는 이벤트의 **ID**를 지정합니다.
  - 공개적으로 액세스 가능한 대상으로 이벤트를 보내는 경우 **--to-url** 플래그를 사용하여 **URL**을 지정합니다.
  - 클러스터 내 **Kubernetes** 리소스로 이벤트를 보내는 경우 **--to** 플래그를 사용하여 대상을 지정합니다.
    - **< Kind>:<ApiVersion>:<name >** 형식을 사용하여 **Kubernetes** 리소스를 지정합니다.
  - **--namespace** 플래그는 네임스페이스를 지정합니다. 생략하면 네임스페이스가 현재 컨텍스트에서 가져옵니다.

--to-url 또는 **--to -to**를 사용해야 하는 대상 사양을 제외하고 이러한 모든 플래그는 선택 사항입니다.

다음 예제에서는 **URL**로 이벤트를 보내는 방법을 보여줍니다.

명령 예

```
$ kn event send \
    --field player.id=6354aa60-ddb1-452e-8c13-24893667de20 \
    --field player.game=2345 \
    --field points=456 \
    --type org.example.gaming.foo \
    --to-url http://ce-api.foo.example.com/
```

다음 예제에서는 클러스터 내 리소스로 이벤트를 전송하는 방법을 보여줍니다.

명령 예

```
$ kn event send \
    --type org.example.kn.ping \
    --id $(uuidgen) \
    --field event.type=test \
    --field event.data=98765 \
    --to Service:serving.knative.dev/v1:event-display
```

7.4. KNATIVE EVENTING CLI 명령

7.4.1. kn source 명령

다음 명령을 사용하여 Knative 이벤트 소스를 나열, 생성 및 관리할 수 있습니다.

7.4.1.1. Knative CLI를 사용하여 사용 가능한 이벤트 소스 유형 나열

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

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.

- Knative(kn) CLI가 설치되어 있습니다.

절차

1. 터미널에서 사용 가능한 이벤트 소스 유형을 나열합니다.

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

출력 예

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

2. 선택 사항: 사용 가능한 이벤트 소스 유형을 YAML 형식으로 나열할 수도 있습니다.

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

7.4.1.2. Knative CLI 싱크 플래그

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

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

싱크 플래그를 사용하는 명령의 예

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

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

1

http://event-display.svc.cluster.local 의 svc 는 싱크가 Knative 서비스인지 확인합니다. 기타 기본 싱크 접두사에는 channel, 및 broker가 포함됩니다.

7.4.1.3. Knative CLI를 사용하여 컨테이너 소스 생성 및 관리

kn 소스 컨테이너 명령을 사용하여 Knative(kn) CLI를 사용하여 컨테이너 소스를 생성하고 관리할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다.

컨테이너 소스 생성

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

컨테이너 소스 삭제

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

컨테이너 소스 설명

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

기존 컨테이너 소스 나열

```
$ kn source container list
```

YAML 형식으로 기존 컨테이너 소스 나열

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

컨테이너 소스 업데이트

이 명령은 기존 컨테이너 소스의 이미지 URI를 업데이트합니다.

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

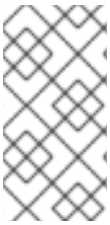
7.4.1.4. Knative CLI를 사용하여 API 서버 소스 생성

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

사전 요구 사항

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

- *Knative(kn) CLI가 설치되어 있습니다.*



절차

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

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

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

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

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

(1) (2) (3) (4)

이 네임스페이스를 이벤트 소스 설치를 위해 선택한 네임스페이스로 변경합니다.

2. *YAML 파일을 적용합니다.*

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

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

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

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

```
$ kn service create <service_name> --image quay.io/openshift-knative/knative-
eventing-sources-event-display:latest
```

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

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

6. *기본 네임스페이스에서 Pod를 시작하여 이벤트를 생성합니다.*

```
$ oc create deployment hello-node --image quay.io/openshift-knative/knative-eventing-
sources-event-display:latest
```

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

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

출력 예

```
Name:              mysource
Namespace:         default
Annotations:       sources.knative.dev/creator=developer,
```

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

검증

*메시지 덤퍼 기능 로그를 확인하면 **Kubernetes** 이벤트가 **Knative**로 전송되었는지 확인할 수 있습니다.*

1. ***Pod**를 가져옵니다.*

```
$ oc get pods
```

2. ***Pod**의 메시지 덤퍼 기능 로그를 확인합니다.*

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

출력 예

```
☁    cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.apiserver.resource.update
  datacontenttype: application/json
  ...
Data,
  {
```

```
  "apiVersion": "v1",
  "involvedObject": {
    "apiVersion": "v1",
    "fieldPath": "spec.containers{hello-node}",
    "kind": "Pod",
    "name": "hello-node",
    "namespace": "default",
     .....
  },
  "kind": "Event",
  "message": "Started container",
  "metadata": {
    "name": "hello-node.159d7608e3a3572c",
    "namespace": "default",
    ....
  },
  "reason": "Started",
  ...
}
```

API 서버 소스 삭제

1. 트리거를 삭제합니다.

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

2. 이벤트 소스를 삭제합니다.

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

3. 서비스 계정, 클러스터 역할, 클러스터 바인딩을 삭제합니다.

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

7.4.1.5. Knative CLI를 사용하여 ping 소스 생성

kn 소스 **ping create** 명령을 사용하여 **Knative(kn) CLI**를 사용하여 **ping** 소스를 생성할 수 있습니다. **Knative CLI**를 사용하여 이벤트 소스를 생성하면 **YAML** 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다.

사전 요구 사항

- *OpenShift Serverless Operator, Knative Serving 및 Knative Eventing이 클러스터에 설치되어 있습니다.*

- *Knative(kn) CLI가 설치되어 있습니다.*

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

- *선택 사항: 이 프로세스에 대한 확인 단계를 사용하려면 OpenShift CLI(oc)를 설치합니다.*

절차

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

```
$ kn service create event-display \
  --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

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

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

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

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

출력 예

```
Name:         test-ping-source
Namespace:    default
Annotations:  sources.knative.dev/creator=developer,
sources.knative.dev/lastModifier=developer
Age:          15s
Schedule:     */2 * * * *
```

```
Data:       {"message": "Hello world!"}

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

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

검증

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

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

1. *새 Pod가 생성되었는지 확인합니다.*

```
$ watch oc get pods
```

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

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

출력 예

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.sources.ping
  source: /apis/v1/namespaces/default/pingsources/test-ping-source
```

```
  id: 99e4f4f6-08ff-4bff-acf1-47f61ded68c9
  time: 2020-04-07T16:16:00.000601161Z
  datacontenttype: application/json
Data,
  {
    "message": "Hello world!"
  }
```

ping 소스 삭제

- *ping 소스를 삭제합니다.*

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

7.4.1.6. Knative CLI를 사용하여 Kafka 이벤트 소스 생성

kn source kafka create 명령을 사용하여 Knative(kn) CLI를 사용하여 Kafka 소스를 생성할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 간소화되고 직관적인 사용자 인터페이스를 제공합니다.

사전 요구 사항

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

절차

1.

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

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

2.

KafkaSource CR을 생성합니다.

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

참고

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

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

3.

선택 사항: 생성한 KafkaSource CR에 대한 세부 정보를 확인합니다.

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

출력 예

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

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

Conditions:
  OK TYPE           AGE REASON
```

```
++ Ready            1h
++ Deployed         1h
++ SinkProvided     1h
```

검증 단계

1. *Kafka 인스턴스를 트리거하여 메시지를 항목에 보냅니다.*

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

프롬프트에 메시지를 입력합니다. 이 명령은 다음을 가정합니다.

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

2. *로그를 보고 메시지가 도착했는지 확인합니다.*

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

출력 예

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

```
traceparent: 00-161ff3815727d8755848ec01c866d1cd-7ff3916c44334678-00
Data,
Hello!
```

7.5. KNATIVE FUNCTIONS CLI 명령

7.5.1. kn 함수 명령

7.5.1.1. 함수 생성

함수를 빌드하고 배포하려면 **Knative(kn) CLI**를 사용하여 생성해야 합니다. 경로, 런타임, 템플릿 및 이미지 레지스트리를 명령줄에서 플래그로 지정하거나 **-c** 플래그를 사용하여 터미널에서 대화형 환경을 시작할 수 있습니다.

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다.
- **Knative(kn) CLI**가 설치되어 있습니다.

절차

- 함수 프로젝트를 생성합니다.

```
$ kn func create -r <repository> -l <runtime> -t <template> <path>
```

  - 허용되는 런타임 값에는 **quarkus,node,typescript,go,python, Springboot,** 및 **rust**가 포함됩니다.
  - 허용되는 템플릿 값에는 **http** 및 **cloudevents** 가 포함됩니다.

명령 예

```
$ kn func create -l typescript -t cloudevents examplefunc
```

출력 예

```
Created typescript function in /home/user/demo/examplefunc
```

- *또는 사용자 지정 템플릿이 포함된 리포지토리를 지정할 수도 있습니다.*

명령 예

```
$ kn func create -r https://github.com/boson-project/templates/ -l node -t hello-world examplefunc
```

출력 예

```
Created node function in /home/user/demo/examplefunc
```

7.5.1.2. 로컬로 함수 실행

kn func run 명령을 사용하여 현재 디렉터리 또는 --path 플래그에서 지정한 디렉터리에서 로컬로 함수를 실행할 수 있습니다. 실행 중인 함수가 이전에 빌드되지 않았거나 마지막으로 작성된 프로젝트 파일이 변경된 경우 kn func run 명령은 기본적으로 함수를 실행하기 전에 함수를 빌드합니다.

현재 디렉터리에서 함수를 실행하는 명령의 예

```
$ kn func run
```

-

경로로 지정된 디렉터리에서 함수를 실행하는 명령의 예

```
$ kn func run --path=<directory_path>
```

--build *플래그를 사용하여 프로젝트 파일이 변경되지 않은 경우에도 함수를 실행하기 전에 기존 이미지를 강제로 다시 빌드할 수도 있습니다.*

빌드 플래그를 사용하는 **run** *명령의 예*

```
$ kn func run --build
```

빌드 플래그를 **false***로 설정하면 이미지 빌드가 비활성화되고 이전에 빌드한 이미지를 사용하여 함수가 실행됩니다.*

빌드 플래그를 사용하는 **run** *명령의 예*

```
$ kn func run --build=false
```

help *명령을 사용하여* **kn func run** *명령 옵션에 대해 자세히 알아볼 수 있습니다.*

빌드 도움말 명령

-

```
$ kn func help run
```

7.5.1.3. 함수 빌드

함수를 실행하려면 함수 프로젝트를 빌드해야 합니다. **kn func run** 명령을 사용하는 경우 함수가 자동으로 빌드됩니다. 그러나 **kn func build** 명령을 사용하여 함수를 실행하지 않고 빌드할 수 있습니다. 이 기능은 고급 사용자 또는 디버깅 시나리오에 유용할 수 있습니다.

kn func build 명령은 컴퓨터 또는 **OpenShift Container Platform** 클러스터에서 로컬로 실행할 수 있는 **OCI** 컨테이너 이미지를 생성합니다. 이 명령은 함수 프로젝트 이름과 이미지 레지스트리 이름을 사용하여 함수에 대해 정규화된 이미지 이름을 구성합니다.

7.5.1.3.1. 이미지 컨테이너 유형

기본적으로 **kn func** 빌드 는 **Red Hat S2I(Source-to-Image)** 기술을 사용하여 컨테이너 이미지를 생성합니다.

Red Hat S2I(Source-to-Image)를 사용하는 빌드 명령의 예

```
$ kn func build
```

7.5.1.3.2. 이미지 레지스트리 유형

OpenShift Container Registry는 기본적으로 함수 이미지를 저장하기 위한 이미지 레지스트리로 사용됩니다.

OpenShift Container Registry를 사용하는 빌드 명령 예

```
$ kn func build
```

출력 예

```
Building function image
Function image has been built, image: registry.redhat.io/example/example-function:latest
```

--registry 플래그를 사용하여 기본 이미지 레지스트리로 OpenShift Container Registry를 재정의할 수 있습니다.

quay.io를 사용하도록 OpenShift Container Registry를 재정의하는 빌드 명령의 예

```
$ kn func build --registry quay.io/username
```

출력 예

```
Building function image
Function image has been built, image: quay.io/username/example-function:latest
```

7.5.1.3.3. push 플래그

kn func build 명령에 --push 플래그를 추가하여 성공적으로 빌드한 후 함수 이미지를 자동으로 푸시할 수 있습니다.

OpenShift Container Registry를 사용하는 빌드 명령 예

```
$ kn func build --push
```

7.5.1.3.4. 도움말 명령

help 명령을 사용하여 kn func build 명령 옵션에 대해 자세히 알아볼 수 있습니다.

빌드 도움말 명령

```
$ kn func help build
```

7.5.1.4. 함수 배포

kn func deploy 명령을 사용하여 Knative 서비스로 클러스터에 함수를 배포할 수 있습니다. 대상 함수가 이미 배포된 경우 컨테이너 이미지 레지스트리로 푸시된 새 컨테이너 이미지로 업데이트되고 Knative 서비스가 업데이트됩니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- Knative(kn) CLI가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 배포하려는 함수를 이미 생성하고 초기화해야 합니다.

절차

- 함수를 배포합니다.

```
$ kn func deploy [-n <namespace> -p <path> -i <image>]
```

출력 예

```
Function deployed at: http://func.example.com
```

- namespace를 지정하지 않으면 함수가 현재 네임스페이스에 배포됩니다.
- 이 함수는 path를 지정하지 않는 한 현재 디렉터리에서 배포됩니다.
- Knative 서비스 이름은 프로젝트 이름에서 파생되며 이 명령을 사용하여 변경할 수 없습니다.

7.5.1.5. 기존 함수 나열

kn func list를 사용하여 기존 함수를 나열할 수 있습니다. Knative 서비스로 배포된 함수를 나열하려면 kn service list를 사용할 수도 있습니다.

절차

- 기존 함수를 나열합니다.

```
$ kn func list [-n <namespace> -p <path>]
```

출력 예

```
NAME            NAMESPACE  RUNTIME  URL
READY
example-function default    node     http://example-function.default.apps.ci-ln-g9f36hb-d5d6b.origin-ci-int-aws.dev.rhcloud.com  True
```

- Knative 서비스로 배포된 함수를 나열합니다.

```
$ kn service list -n <namespace>
```

출력 예

```
NAME           URL                                                                LATEST
AGE   CONDITIONS   READY   REASON
example-function   http://example-function.default.apps.ci-ln-g9f36hb-d5d6b.origin-ci-
int-aws.dev.rhcloud.com   example-function-gzl4c   16m   3 OK / 3     True
```

7.5.1.6. 함수 설명

kn func info 명령은 함수 이름, 이미지, 네임스페이스, Knative 서비스 정보, 경로 정보 및 이벤트 서브스크립션과 같은 배포된 기능에 대한 정보를 출력합니다.

절차

- 함수를 설명합니다.

```
$ kn func info [-f <format> -n <namespace> -p <path>]
```

명령 예

```
$ kn func info -p function/example-function
```

출력 예

```
Function name:
  example-function
Function is built in image:
  docker.io/user/example-function:latest
```

```
Function is deployed as Knative Service:
  example-function
Function is deployed in namespace:
  default
Routes:
  http://example-function.default.apps.ci-ln-g9f36hb-d5d6b.origin-ci-int-
aws.dev.rhcloud.com
```

7.5.1.7. 테스트 이벤트를 사용하여 배포된 함수 호출

kn func invoke CLI 명령을 사용하여 로컬에서 또는 **OpenShift Container Platform** 클러스터에서 함수를 호출하기 위해 테스트 요청을 보낼 수 있습니다. 이 명령을 사용하여 함수가 작동하고 이벤트를 올바르게 수신할 수 있는지 테스트할 수 있습니다. 함수를 로컬로 호출하면 함수 개발 중에 빠른 테스트에 유용합니다. 클러스터에서 함수를 호출하면 프로덕션 환경에 더 가까운 테스트에 유용합니다.

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다.
- **Knative(kn) CLI**가 설치되어 있습니다.
- 프로젝트를 생성했거나 **OpenShift Container Platform**에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 호출하려는 함수를 이미 배포해야 합니다.

절차

- 함수를 호출합니다.

```
$ kn func invoke
```

  - **kn func invoke** 명령은 현재 실행 중인 로컬 컨테이너 이미지가 있거나 클러스터에 배포된 함수가 있는 경우에만 작동합니다.
  - **kn func invoke** 명령은 기본적으로 로컬 디렉터리에서 실행되며 이 디렉터리는 함수

프로젝트라고 가정합니다.

7.5.1.7.1. kn func invoke 선택적 매개 변수

***kn func invoke CLI** 명령 플래그를 사용하여 요청에 대한 선택적 매개변수를 지정할 수 있습니다.*

플래그	설명
-t,--target	호출된 함수의 대상 인스턴스를 지정합니다(예: **로컬** 또는 **원격** 또는 **https://staging.example.com/**). 기본 대상은 **로컬** 입니다.
-f, --format	메시지 형식을 지정합니다(예: **cloudevent** 또는 **http**).
--id	요청에 대한 고유 문자열 식별자를 지정합니다.
-n,--namespace	클러스터의 네임스페이스를 지정합니다.
--source	요청의 발신자 이름을 지정합니다. 이는 CloudEvent **소스** 속성에 해당합니다.
--type	요청 유형을 지정합니다(예: **boson.fn**). 이는 CloudEvent **유형** 속성에 해당합니다.
--data	요청에 대한 콘텐츠를 지정합니다. CloudEvent 요청의 경우 CloudEvent **data** 속성입니다.
--file	보낼 데이터를 포함하는 로컬 파일의 경로를 지정합니다.
--content-type	요청에 대한 MIME 콘텐츠 형식을 지정합니다.
-p, --path	프로젝트 디렉터리의 경로를 지정합니다.
-c, --confirm	모든 옵션을 대화형으로 확인하도록 프롬프트를 활성화합니다.
-v,--verbose	자세한 출력을 인쇄할 수 있습니다.
-h, --help	**kn func invoke** 사용에 대한 정보를 출력합니다.

7.5.1.7.1.1. 기본 매개변수

*다음 매개 변수는 **kn func invoke** 명령의 기본 속성을 정의합니다.*

이벤트 대상(-t,--target)

*호출된 함수의 대상 인스턴스입니다. 로컬 로 배포된 함수의 로컬 값, 원격 으로 배포된 함수의 원격 값 또는 임의의 엔드포인트에 배포된 함수의 **URL**을 수락합니다. 대상을 지정하지 않으면 기본값*

은 **local** 입니다.

*이벤트 메시지 형식(**-f,--format**)*

이벤트의 메시지 형식(예: **http** 또는 **cloudevent**). 이 기본값은 함수를 만들 때 사용된 템플릿 형식입니다.

*이벤트 유형 (**--type**)*

전송된 이벤트 유형입니다. 각 이벤트 프로듀서에 대한 문서에 설정된 **type** 매개변수에 대한 정보를 찾을 수 있습니다. 예를 들어 **API** 서버 소스는 생성된 이벤트의 **type** 매개변수를 **dev.knative.apiserver.resource.update** 로 설정할 수 있습니다.

*이벤트 소스(**--source**)*

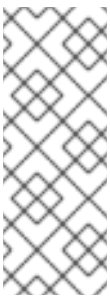
이벤트를 생성한 고유한 이벤트 소스입니다. 이벤트 소스의 **URI**(예: https://10.96.0.1/) 또는 이벤트 소스의 이름일 수 있습니다.

*이벤트 **ID**(**--id**)*

이벤트 프로듀서에 의해 생성되는 임의의 고유 **ID**입니다.

*이벤트 데이터(**--data**)*

kn func invoke 명령에서 보낸 이벤트에 대한 데이터 값을 지정할 수 있습니다. 예를 들어 이벤트에 이 데이터 문자열이 포함되도록 **"Hello World"** 와 같은 **--data** 값을 지정할 수 있습니다. 기본적으로 **kn func invoke** 에서 생성한 이벤트에는 데이터가 포함되지 않습니다.



참고

클러스터에 배포된 함수는 **source** 및 **type**과 같은 속성 값을 제공하는 기존 이벤트 소스의 이벤트에 응답할 수 있습니다. 이러한 이벤트에는 종종 이벤트의 도메인별 컨텍스트를 캡처하는 **JSON** 형식의 **data** 값이 있습니다. 개발자는 이 문서에 명시된 **CLI** 플래그를 사용하여 로컬 테스트를 위해 해당 이벤트를 시뮬레이션할 수 있습니다.

이벤트 데이터를 포함하는 로컬 파일을 제공하기 위해 **--file** 플래그를 사용하여 이벤트 데이터를 보낼 수도 있습니다. 이 경우 **--content-type** 을 사용하여 콘텐츠 유형을 지정합니다.

*데이터 콘텐츠 유형 (**--content-type**)*

--data 플래그를 사용하여 이벤트에 대한 데이터를 추가하는 경우 **--content-type** 플래그를 사용하여 이벤트에서 전달하는 데이터 유형을 지정할 수 있습니다. 이전 예에서 데이터는 일반 텍스트이므로 **kn func invoke --data "Hello world!" --content-type "text/plain"** 을 지정할 수 있습니다.

7.5.1.7.1.2. 명령 예

이는 **kn func invoke** 명령을 일반적인 호출입니다.

```
$ kn func invoke --type <event_type> --source <event_source> --data <event_data> --content-type <content_type> --id <event_ID> --format <format> --namespace <namespace>
```

예를 들어 **"Hello world!"** 이벤트를 보내려면 다음을 실행할 수 있습니다.

```
$ kn func invoke --type ping --source example-ping --data "Hello world!" --content-type "text/plain" --id example-ID --format http --namespace my-ns
```

7.5.1.7.1.2.1. 데이터로 파일 지정

이벤트 데이터가 포함된 디스크에서 파일을 지정하려면 **--file** 및 **--content-type** 플래그를 사용합니다.

```
$ kn func invoke --file <path> --content-type <content-type>
```

예를 들어 **test.json** 파일에 저장된 **JSON** 데이터를 보내려면 다음 명령을 사용하십시오.

```
$ kn func invoke --file ./test.json --content-type application/json
```

7.5.1.7.1.2.2. 함수 프로젝트 지정

--path 플래그를 사용하여 함수 프로젝트의 경로를 지정할 수 있습니다.

```
$ kn func invoke --path <path_to_function>
```

예를 들어 **./example/example-function** 디렉터리에 있는 **function** 프로젝트를 사용하려면 다음 명령을 사용하십시오.

```
$ kn func invoke --path ./example/example-function
```

7.5.1.7.1.2.3. 대상 함수의 배포 위치 지정

기본적으로 **kn func invoke** 는 함수의 로컬 배포를 대상으로 합니다.

```
$ kn func invoke
```

다른 배포를 사용하려면 **--target** 플래그를 사용합니다.

```
$ kn func invoke --target <target>
```

예를 들어 클러스터에 배포된 함수를 사용하려면 **--target** 원격 플래그를 사용합니다.

```
$ kn func invoke --target remote
```

임의의 **URL**에 배포된 함수를 사용하려면 **--target <URL>** 플래그를 사용합니다.

```
$ kn func invoke --target "https://my-event-broker.example.com"
```

로컬 배포를 명시적으로 대상으로 할 수 있습니다. 이 경우 함수가 로컬로 실행되지 않으면 명령이 실패합니다.

```
$ kn func invoke --target local
```

7.5.1.8. 함수 삭제

kn func delete 명령을 사용하여 함수를 삭제할 수 있습니다. 이 기능은 함수가 더 이상 필요하지 않은 경우 유용하며 클러스터에 리소스를 저장하는 데 도움이 될 수 있습니다.

절차

- 함수를 삭제합니다.

  ```
  $ kn func delete [<function_name> -n <namespace> -p <path>]
  ```

  - 삭제할 함수의 이름 또는 경로가 지정되지 않은 경우 현재 디렉터리에서 **func.yaml** 파일을 검색하고 삭제할 함수를 결정합니다.
  - 네임스페이스를 지정하지 않으면 기본값은 **func.yaml** 파일의 **namespace** 값으로 설정됩니다.

8장. 가시성

8.1. 관리자 메트릭

8.1.1. 서버리스 관리자 메트릭

클러스터 관리자는 메트릭을 사용하여 OpenShift Serverless 클러스터 구성 요소 및 워크로드를 수행하는 방법을 모니터링할 수 있습니다.

OpenShift Container Platform 웹 콘솔 관리자 화면에서 대시보드로 이동하여 OpenShift Serverless의 다양한 메트릭을 볼 수 있습니다.

8.1.1.1. 사전 요구 사항

- 클러스터 메트릭 활성화에 대한 정보는 메트릭 관리에 대한 OpenShift Container Platform 설명서를 참조하십시오.
- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- OpenShift Container Platform 웹 콘솔의 관리자 화면에 액세스할 수 있습니다.

주의

Service Mesh가 mTLS를 사용하여 사용하도록 설정된 경우 Service Mesh가 Prometheus의 메트릭 스크랩을 허용하지 않기 때문에 기본적으로 Knative Serviceing에 대한 메트릭이 사용되지 않도록 설정됩니다.

이 문제를 해결하는 방법에 대한 자세한 내용은 mTLS를 사용하여 서비스 메시를 사용할 때 Knative Serving 메트릭 활성화를 참조하십시오.

메트릭을 스크랩하는 작업은 스크랩 요청이 활성화를 통과하지 않기 때문에 Knative 서비스의 자동 확장에 영향을 미치지 않습니다. 결과적으로 실행 중인 Pod가 없는 경우 스크랩이 수행되지 않습니다.

8.1.2. 서버리스 컨트롤러 메트릭

다음 메트릭은 컨트롤러 로직을 구현하는 구성 요소에서 출력됩니다. 이러한 메트릭은 조정 작업 및 조정 요청이 작업 대기열에 추가되는 작업 대기열 동작에 대한 세부 정보를 표시합니다.

메트릭 이름	설명	유형	태그	단위
work_queue_depth	작업 대기열의 깊이	게이지	**reconciler**	정수(단위 없음)
reconcile_count	조정 작업 수	카운터	**reconciler**, **success**	정수(단위 없음)
reconcile_latency	조정 작업 대기 시간	히스토그램	**reconciler**, **success**	밀리초
workqueue_adds_total	작업 대기열에서 처리하는 총 추가 작업 수	카운터	**name**	정수(단위 없음)
workqueue_queue_latency_seconds	항목이 요청되기 전에 작업 대기열에 남아 있는 시간	히스토그램	**name**	초
workqueue_retries_total	작업 대기열에서 처리한 총 재시도 횟수	카운터	**name**	정수(단위 없음)

메트릭 이름	설명	유형	태그	단위
workqueue_work_duration_seconds	작업 대기열에서 처리하는 데 걸리는 시간 및 항목	히스토그램	**name**	초
workqueue_unfinished_work_seconds	처리되지 않은 작업 대기열 항목이 진행 중인 시간	히스토그램	**name**	초
workqueue_longest_running_processor_seconds	가장 긴 미결 작업 대기열 항목이 진행 중인 시간	히스토그램	**name**	초

8.1.3. Webhook 메트릭

Webhook 메트릭은 작업에 대한 유용한 정보를 표시합니다. 예를 들어, 많은 작업이 실패하면 사용자가 생성한 리소스에 문제가 있음을 나타내는 것일 수 있습니다.

메트릭 이름	설명	유형	태그	단위
request_count	webhook로 라우팅되는 요청 수입니다.	카운터	**admission_allowed**, **kind_group**, **kind_kind**, **kind_version**, **request_operation**, **resource_group**, **resource_namespace**, **resource_resource**, **resource_version**	정수(단위 없음)

메트릭 이름	설명	유형	태그	단위
request_latencies	webhook 요청에 대한 응답 시간입니다.	히스토그램	**admission_allowed**, **kind_group**, **kind_kind**, **kind_version**, **request_operation**, **resource_group**, **resource_namespace**, **resource_resource**, **resource_version**	밀리초

8.1.4. Knative Eventing 메트릭

*클러스터 관리자는 **Knative Eventing** 구성 요소에 대한 다음 메트릭을 볼 수 있습니다.*

***HTTP** 코드에서 메트릭을 집계하여 정상적인 이벤트 **(2xx)** 및 실패한 이벤트**(5xx)**의 두 가지 범주로 구분할 수 있습니다.*

8.1.4.1. 브로커 Ingress 메트릭

*다음 메트릭을 사용하여 브로커 **Ingress**를 디버그하고, 수행 방법을 확인하고, **Ingress** 구성 요소에서 전달되는 이벤트를 확인할 수 있습니다.*

메트릭 이름	설명	유형	태그	단위
event_count	브로커가 수신한 이벤트 수.	카운터	**broker_name**, **event_type**, **namespace_name**, **response_code**, **response_code_class**, **unique_name**	정수(단위 없음)

메트릭 이름	설명	유형	태그	단위
event_dispatch_latencies	이벤트를 채널로 전달하는 데 걸린 시간입니다.	히스토그램	**broker_name**, **event_type**, **namespace_name**, **response_code**, **response_code_class**, **unique_name**	밀리초

8.1.4.2. 브로커 필터 메트릭

다음 메트릭을 사용하여 브로커 필터를 디버그하고, 수행 방법을 확인하고, 필터에서 전달되는 이벤트를 확인할 수 있습니다. 이벤트에서 필터링 작업의 대기 시간을 측정할 수도 있습니다.

메트릭 이름	설명	유형	태그	단위
event_count	브로커가 수신한 이벤트 수.	카운터	**broker_name**, **container_name**, **filter_type**, **namespace_name**, **response_code**, **response_code_class**, **trigger_name**, **unique_name**	정수(단위 없음)
event_dispatch_latencies	이벤트를 채널로 전달하는 데 걸린 시간입니다.	히스토그램	**broker_name**, **container_name**, **filter_type**, **namespace_name**, **response_code**, **response_code_class**, **trigger_name**, **unique_name**	밀리초
event_processing_latencies	트리거 구독자에게 전달되기 전에 이벤트를 처리하는 데 걸리는 시간입니다.	히스토그램	**broker_name**, **container_name**, **filter_type**, **namespace_name**, **trigger_name**, **unique_name**	밀리초

8.1.4.3. InMemoryChannel 디스패처 메트릭

*다음 메트릭을 사용하여 **InMemoryChannel** 채널을 디버그하고, 어떻게 수행하는지, 채널에서 디스패치 중인 이벤트를 확인할 수 있습니다.*

메트릭 이름	설명	유형	태그	단위
event_count	**InMemoryChannel** 채널에서 디스패치하는 이벤트 수.	카운터	**broker_name**, **container_name**, **filter_type**, **namespace_name**, **response_code**, **response_code_class**, **trigger_name**, **unique_name**	정수(단위 없음)
event_dispatch_latencies	**InMemoryChannel** 채널에서 이벤트를 디스패치하는 데 걸린 시간.	히스토그램	**broker_name**, **container_name**, **filter_type**, **namespace_name**, **response_code**, **response_code_class**, **trigger_name**, **unique_name**	밀리초

8.1.4.4. 이벤트 소스 메트릭

다음 메트릭을 사용하여 이벤트 소스에서 연결된 이벤트 싱크로 이벤트가 전달되었는지 확인할 수 있습니다.

메트릭 이름	설명	유형	태그	단위
event_count	이벤트 소스에서 보낸 이벤트 수입니다.	카운터	**broker_name**, **container_name**, **filter_type**, **namespace_name**, **response_code**, **response_code_class**, **trigger_name**, **unique_name**	정수(단위 없음)

메트릭 이름	설명	유형	태그	단위
retry_event_count	처음에 이벤트 소스가 전송되지 못한 후 이벤트 소스에 의해 전송되는 재시도 이벤트 수입니다.	카운터	**event_source**, **event_type**, **name**, **namespace_name**, **resource_group**, **response_code**, **response_code_class**, **response_error**, **response_timeout**	정수(단위 없음)

8.1.5. Knative Serving 메트릭

*클러스터 관리자는 **Knative Serving** 구성 요소에 대한 다음 메트릭을 볼 수 있습니다.*

8.1.5.1. 활성화기 메트릭

다음 메트릭을 사용하여 트래픽이 활성화기를 통해 전달될 때 애플리케이션이 응답하는 방식을 파악할 수 있습니다.

메트릭 이름	설명	유형	태그	단위
request_concurrency	활성화기 또는 보고 기간에 평균 동시성으로 라우팅되는 동시 요청 수입니다.	게이지	**configuration_name**, **container_name**, **namespace_name**, **pod_name**, **revision_name**, **service_name**	정수(단위 없음)

메트릭 이름	설명	유형	태그	단위
request_count	활성화기로 라우팅되는 요청 수입니다. 이러한 요청은 활성화기 처리기에서 실행된 요청입니다.	카운터	**configuration_name**, **container_name**, **namespace_name**, **pod_name**, **response_code**, **response_code_class**, **revision_name**, **service_name**,	정수(단위 없음)
request_latencies	실행된 라우팅된 요청에 대한 응답 시간(밀리초)입니다.	히스토그램	**configuration_name**, **container_name**, **namespace_name**, **pod_name**, **response_code**, **response_code_class**, **revision_name**, **service_name**	밀리초

8.1.5.2. 자동 스케일러 메트릭

*자동 스케일러 구성 요소는 각 버전의 자동 스케일러 동작과 관련된 여러 메트릭을 표시합니다. 예를 들어, 자동 스케일러가 서비스에 할당하려는 대상 **Pod** 수, 안정적인 기간 동안 초당 평균 요청 수 또는 **Knative Pod** 자동 스케일러(KPA)를 사용하는 경우 자동 스케일러가 패닉 모드인지를 모니터링할 수 있습니다.*

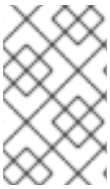
메트릭 이름	설명	유형	태그	단위
desired_pods	자동 스케일러가 서비스에 할당하려는 Pod 수입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)
excess_burst_capacity	stable 창에서 제공되는 추가 버스트 용량입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)

메트릭 이름	설명	유형	태그	단위
stable_request_concurrency	stable 창에서 모니터링되는 각 pod의 평균 요청 수입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)
panic_request_concurrency	panic 창에서 모니터링되는 각 Pod의 평균 요청 수입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)
target_concurrency_per_pod	자동 스케일러가 각 pod에 보내려는 동시 요청 수입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)
stable_requests_per_second	stable 창에서 모니터링되는 각 pod의 평균 요청 수(초당)입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)
panic_requests_per_second	panic 창에서 모니터링되는 각 Pod의 평균 요청 수입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)
target_requests_per_second	각 pod에 대해 자동 스케일러가 대상으로 하는 초당 요청 수입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)
panic_mode	자동 스케일러가 패닉 모드이면 이 값은 **1**이며 자동 스케일러가 패닉 모드가 아닌 경우 **0**입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)

메트릭 이름	설명	유형	태그	단위
requested_pods	Kubernetes 클러스터에서 자동 스케일러가 요청한 Pod 수입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)
actual_pods	할당되어 있고 현재 준비 상태인 Pod 수입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)
not_ready_pods	준비되지 않은 상태의 Pod 수입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)
pending_pods	현재 보류 중인 pod 수입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)
terminating_pods	현재 종료 중인 Pod 수입니다.	게이지	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	정수(단위 없음)

8.1.5.3. Go 런타임 메트릭

*각 **Knative Serving** 컨트롤 플레인 프로세스는 수많은 **Go** 런타임 메모리 통계(**MemStats**)를 출력합니다.*



참고

*각 메트릭의 **name** 태그는 빈 태그입니다.*

메트릭 이름	설명	유형	태그	단위
go_alloc	할당된 힙 오브젝트의 바이트 수입니다. 이 메트릭은 **heap_alloc**과 동일합니다.	게이지	**name**	정수(단위 없음)
go_total_alloc	힙 오브젝트에 할당된 누적 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_sys	운영 체제에서 얻은 총 메모리 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_lookups	런타임에서 수행하는 포인터 검색 수입니다.	게이지	**name**	정수(단위 없음)
go_mallocs	할당된 힙 오브젝트의 누적 개수입니다.	게이지	**name**	정수(단위 없음)
go_frees	해제된 힙 오브젝트의 누적 개수입니다.	게이지	**name**	정수(단위 없음)
go_heap_alloc	할당된 힙 오브젝트의 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_heap_sys	운영 체제에서 얻은 힙 메모리의 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_heap_idle	유휴 상태의 사용되지 않는 범위의 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_heap_in_use	현재 사용 중인 범위의 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_heap_released	운영 체제로 반환되는 실제 메모리의 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_heap_objects	할당된 힙 오브젝트 수입니다.	게이지	**name**	정수(단위 없음)

메트릭 이름	설명	유형	태그	단위
go_stack_in_use	현재 사용 중인 스택 범위의 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_stack_sys	운영 체제에서 얻은 스택 메모리의 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_mspan_in_use	할당된 **mspan** 구조의 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_mspan_sys	**mspan** 구조의 운영 체제에서 얻은 메모리 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_mcache_in_use	할당된 **mcache** 구조의 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_mcache_sys	**mcache** 구조의 운영 체제에서 얻은 메모리 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_bucket_hash_sys	버킷 해시 테이블 프로파일에서의 메모리 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_gc_sys	가비지 컬렉션 메타데이터의 메모리 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_other_sys	기타, 오프-힙 런타임 할당의 메모리 바이트 수입니다.	게이지	**name**	정수(단위 없음)
go_next_gc	다음 가비지 컬렉션 사이클의 대상 힙 크기입니다.	게이지	**name**	정수(단위 없음)
go_last_gc	마지막 가비지 컬렉션이*Epoch* 또는 Unix 시간으로 완료된 시간입니다.	게이지	**name**	나노초

메트릭 이름	설명	유형	태그	단위
go_total_gc_pause_ns	프로그램이 시작된 이후 가비지 컬렉션 *stop-the-world* 일시 중지의 누적 시간입니다	게이지	**name**	나노초
go_num_gc	완료된 가비지 컬렉션 사이클 수입니다.	게이지	**name**	정수(단위 없음)
go_num_forced_gc	가비지 컬렉션 기능을 호출하는 애플리케이션으로 인해 강제된 가비지 컬렉션 사이클의 수입니다.	게이지	**name**	정수(단위 없음)
go_gc_cpu_fraction	프로그램이 시작된 이후 가비지 컬렉터에서 사용한 프로그램의 사용 가능한 일부 CPU 시간입니다.	게이지	**name**	정수(단위 없음)

8.2. 개발자 지표

8.2.1. 서버리스 개발자 메트릭 개요

*지표를 사용하면 개발자가 **Knative** 서비스 성능을 모니터링할 수 있습니다. **OpenShift Container Platform** 모니터링 스택을 사용하여 **Knative** 서비스에 대한 상태 점검 및 메트릭을 기록하고 확인할 수 있습니다.*

***OpenShift Container Platform** 웹 콘솔 개발자 화면에서 대시보드로 이동하여 **OpenShift Serverless**의 다양한 메트릭을 볼 수 있습니다.*

주의

Service Mesh가 mTLS를 사용하여 사용하도록 설정된 경우 Service Mesh가 Prometheus의 메트릭 스크랩을 허용하지 않기 때문에 기본적으로 Knative Serviceing에 대한 메트릭이 사용되지 않도록 설정됩니다.

이 문제를 해결하는 방법에 대한 자세한 내용은 mTLS를 사용하여 서비스 메시를 사용할 때 Knative Serving 메트릭 활성화를 참조하십시오.

메트릭을 스크랩하는 작업은 스크랩 요청이 활성화를 통과하지 않기 때문에 Knative 서비스의 자동 확장에 영향을 미치지 않습니다. 결과적으로 실행 중인 Pod가 없는 경우 스크랩이 수행되지 않습니다.

8.2.1.1. 추가 리소스

- 모니터링 개요
- 사용자 정의 프로젝트 모니터링 활성화
- 서비스 모니터링 방법 지정

8.2.2. 기본적으로 노출되는 Knative 서비스 메트릭

표 8.1. 포트 9090의 각 Knative 서비스에 대해 기본적으로 노출되는 메트릭

메트릭 이름, 단위 및 유형	설명	메트릭 태그
queue_requests_per_second 메트릭 단위: 무차원 단위 메트릭 유형: 게이지	큐 프록시에 도달하는 초당 요청 수입니다. Formula: **stats.RequestCount / r.reportingPeriodSeconds** **stats.RequestCount**는 지정된 보고 기간 동안 네트워킹 **pkg** 통계에서 직접 계산됩니다.	destination_configuration="event-display", destination_namespace="pingsource1", destination_pod="event-display-00001-deployment-6b455479cb-75p6w", destination_revision="event-display-00001"

메트릭 이름, 단위 및 유형	설명	메트릭 태그
queue_proxied_operations_per_second 메트릭 단위: 무차원 단위 메트릭 유형: 게이지	초당 프록시된 요청 수입니다. 공식: **stats.ProxiedRequestCount / r.reportingPeriodSeconds** **stats.ProxiedRequestCount** 는 지정된 보고 기간 동안 네트워킹 **pkg** 통계에서 직접 계산됩니다.	
queue_average_concurrent_requests 메트릭 단위: 무차원 단위 메트릭 유형: 게이지	이 Pod에서 현재 처리 중인 요청 수입니다. 평균 동시성은 다음과 같이 네트워킹 **pkg** 측에서 계산됩니다. ● **req** 변경이 발생하면 변경 사이의 시간이 계산됩니다. 결과에 따라 delta를 초과하는 현재 동시성이 계산되어 현재 계산된 동시성에 추가됩니다. 또한 delta의 합계가 유지됩니다. delta를 통한 현재 동시성은 다음과 같이 계산됩니다. **global_concurrency** × delta ● 보고가 완료되면 합계와 현재 계산된 동시성이 재설정됩니다. ● 평균 동시성을 보고할 때 현재 계산된 동시성은 deltas의 합계로 나뉩니다. ● 새 요청이 도착하면 글로벌 동시성 카운터가 증가합니다. 요청이 완료되면 카운터가 줄어듭니다.	destination_configuration="event-display", destination_namespace="pingsource1", destination_pod="event-display-00001-deployment-6b455479cb-75p6w", destination_revision="event-display-00001"
queue_average_proxied_concurrent_requests 메트릭 단위: 무차원 단위 메트릭 유형: 게이지	이 Pod에서 현재 처리하는 프록시된 요청 수입니다. **stats.AverageProxiedConcurrency**	destination_configuration="event-display", destination_namespace="pingsource1", destination_pod="event-display-00001-deployment-6b455479cb-75p6w", destination_revision="event-display-00001"

메트릭 이름, 단위 및 유형	설명	메트릭 태그
process_uptime 메트릭 단위: 초 메트릭 유형: 게이지	프로세스가 작동된 시간(초)입니다.	destination_configuration="event-display", destination_namespace="pingsource1", destination_pod="event-display-00001-deployment-6b455479cb-75p6w", destination_revision="event-display-00001"

표 8.2. 포트 9091의 각 Knative 서비스에 대해 기본적으로 노출되는 메트릭

메트릭 이름, 단위 및 유형	설명	메트릭 태그
request_count 메트릭 단위: 무차원 단위 메트릭 유형: 카운터	**queue-proxy**로 라우팅되는 요청 수입니다.	configuration_name="event-display", container_name="queue-proxy", namespace_name="apiserversource1", pod_name="event-display-00001-deployment-658fd4f9cf-qcnr5", response_code="200", response_code_class="2xx", revision_name="event-display-00001", service_name="event-display"
request_latencies 메트릭 단위: 밀리초 메트릭 유형: 히스토그램	응답 시간(밀리초)입니다.	configuration_name="event-display", container_name="queue-proxy", namespace_name="apiserversource1", pod_name="event-display-00001-deployment-658fd4f9cf-qcnr5", response_code="200", response_code_class="2xx", revision_name="event-display-00001", service_name="event-display"

메트릭 이름, 단위 및 유형	설명	메트릭 태그
app_request_count 메트릭 단위: 무차원 단위 메트릭 유형: 카운터	**user-container**로 라우팅되는 요청 수입니다.	configuration_name="event-display", container_name="queue-proxy", namespace_name="apiserversource1", pod_name="event-display-00001-deployment-658fd4f9cf-qcnr5", response_code="200", response_code_class="2xx", revision_name="event-display-00001", service_name="event-display"
app_request_latencies 메트릭 단위: 밀리초 메트릭 유형: 히스토그램	응답 시간(밀리초)입니다.	configuration_name="event-display", container_name="queue-proxy", namespace_name="apiserversource1", pod_name="event-display-00001-deployment-658fd4f9cf-qcnr5", response_code="200", response_code_class="2xx", revision_name="event-display-00001", service_name="event-display"
queue_depth 메트릭 단위: 무차원 단위 메트릭 유형: 게이지	서비스 및 대기열에 있는 현재 항목 수 또는 무제한 동시 실행인 경우 보고되지 않은 항목 수입니다. **breaker.inFlight**가 사용됩니다.	configuration_name="event-display", container_name="queue-proxy", namespace_name="apiserversource1", pod_name="event-display-00001-deployment-658fd4f9cf-qcnr5", response_code="200", response_code_class="2xx", revision_name="event-display-00001", service_name="event-display"

8.2.3. 사용자 정의 애플리케이션 메트릭이 있는 Knative 서비스

***Knative** 서비스에서 내보낸 메트릭 집합을 확장할 수 있습니다. 정확한 구현은 애플리케이션 및 사용된 언어에 따라 다릅니다.*

*다음 목록에서는 처리된 이벤트 사용자 지정 메트릭을 내보내는 샘플 **Go** 애플리케이션을 구현합니다.*

```
package main

import (
  "fmt"
```

```
  "log"
  "net/http"
  "os"

  "github.com/prometheus/client_golang/prometheus" 1
  "github.com/prometheus/client_golang/prometheus/promauto"
  "github.com/prometheus/client_golang/prometheus/promhttp"
)

var (
  opsProcessed = promauto.NewCounter(prometheus.CounterOpts{ 2
    Name: "myapp_processed_ops_total",
    Help: "The total number of processed events",
  })
)


func handler(w http.ResponseWriter, r *http.Request) {
  log.Print("helloworld: received a request")
  target := os.Getenv("TARGET")
  if target == "" {
    target = "World"
  }
  fmt.Fprintf(w, "Hello %s!\n", target)
  opsProcessed.Inc() 3
}

func main() {
  log.Print("helloworld: starting server...")

  port := os.Getenv("PORT")
  if port == "" {
    port = "8080"
  }

  http.HandleFunc("/", handler)

  // Separate server for metrics requests
  go func() { 4
    mux := http.NewServeMux()
    server := &http.Server{
      Addr: fmt.Sprintf(":%s", "9095"),
      Handler: mux,
    }
    mux.Handle("/metrics", promhttp.Handler())
    log.Printf("prometheus: listening on port %s", 9095)
    log.Fatal(server.ListenAndServe())
  }()

   // Use same port as normal requests for metrics
  //http.Handle("/metrics", promhttp.Handler()) 5
  log.Printf("helloworld: listening on port %s", port)
  log.Fatal(http.ListenAndServe(fmt.Sprintf(":%s", port), nil))
}
```

1

Prometheus 패키지 포함.

2

opsProcessed 메트릭 정의.

3

opsProcessed 메트릭 증가.

4

메트릭 요청에 대해 별도의 서버를 사용하도록 구성.

5

메트릭 및 metrics 하위 경로에 대한 일반 요청과 동일한 포트를 사용하도록 구성.

8.2.4. 사용자 정의 메트릭 스크랩 구성

사용자 정의 메트릭 스크랩은 사용자 워크로드 모니터링을 위해 Prometheus의 인스턴스에서 수행합니다. 사용자 워크로드 모니터링을 활성화하고 애플리케이션을 생성한 후에는 모니터링 스택에서 메트릭을 스크랩하는 방법을 정의하는 구성이 필요합니다.

다음 샘플 구성은 애플리케이션에 대한 ksvc를 정의하고 서비스 모니터를 구성합니다. 정확한 구성은 애플리케이션과 메트릭을 내보내는 방법에 따라 다릅니다.

```
apiVersion: serving.knative.dev/v1 1
kind: Service
metadata:
  name: helloworld-go
spec:
  template:
    metadata:
      labels:
        app: helloworld-go
      annotations:
    spec:
      containers:
      - image: docker.io/skonto/helloworld-go:metrics
        resources:
          requests:
            cpu: "200m"
        env:
```

```
        - name: TARGET
          value: "Go Sample v1"
---
apiVersion: monitoring.coreos.com/v1 2
kind: ServiceMonitor
metadata:
  labels:
  name: helloworld-go-sm
spec:
  endpoints:
  - port: queue-proxy-metrics
    scheme: http
  - port: app-metrics
    scheme: http
  namespaceSelector: {}
  selector:
    matchLabels:
      name:  helloworld-go-sm
---
apiVersion: v1 3
kind: Service
metadata:
  labels:
    name:  helloworld-go-sm
  name:  helloworld-go-sm
spec:
  ports:
  - name: queue-proxy-metrics
    port: 9091
    protocol: TCP
    targetPort: 9091
  - name: app-metrics
    port: 9095
    protocol: TCP
    targetPort: 9095
  selector:
    serving.knative.dev/service: helloworld-go
  type: ClusterIP
```

1

애플리케이션 사양.

2

스크랩되는 애플리케이션의 메트릭 구성.

3

메트릭을 스크랩하는 방식의 구성.

8.2.5. 서비스의 메트릭 검사

메트릭을 내보내도록 애플리케이션을 구성하고 모니터링 스택을 스크랩하면 웹 콘솔에서 메트릭을 검사할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 웹 콘솔에 로그인했습니다.
- OpenShift Serverless Operator 및 Knative Serving이 설치되어 있습니다.

절차

1. 선택 사항: 메트릭에서 볼 수 있는 애플리케이션에 대한 요청을 실행합니다.

```
$ hello_route=$(oc get ksvc helloworld-go -n ns1 -o jsonpath='{.status.url}') && \
  curl $hello_route
```

출력 예

```
Hello Go Sample v1!
```

2. 웹 콘솔에서 **Observe** → **Metrics** 인터페이스로 이동합니다.
3. 입력 필드에 모니터링할 메트릭의 쿼리를 입력합니다. 예를 들면 다음과 같습니다.

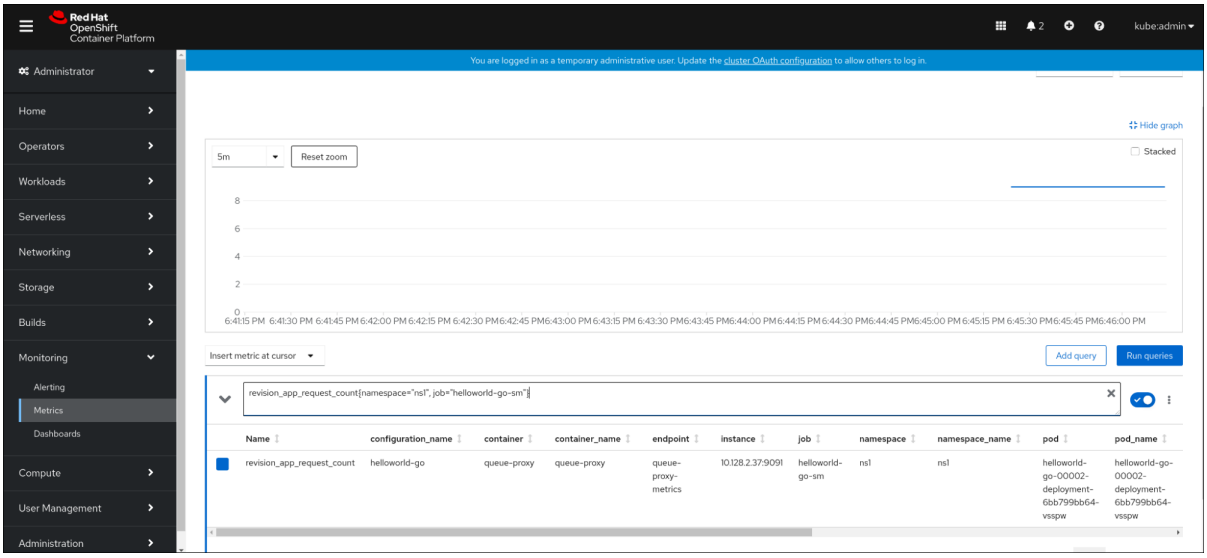
```
revision_app_request_count{namespace="ns1", job="helloworld-go-sm"}
```

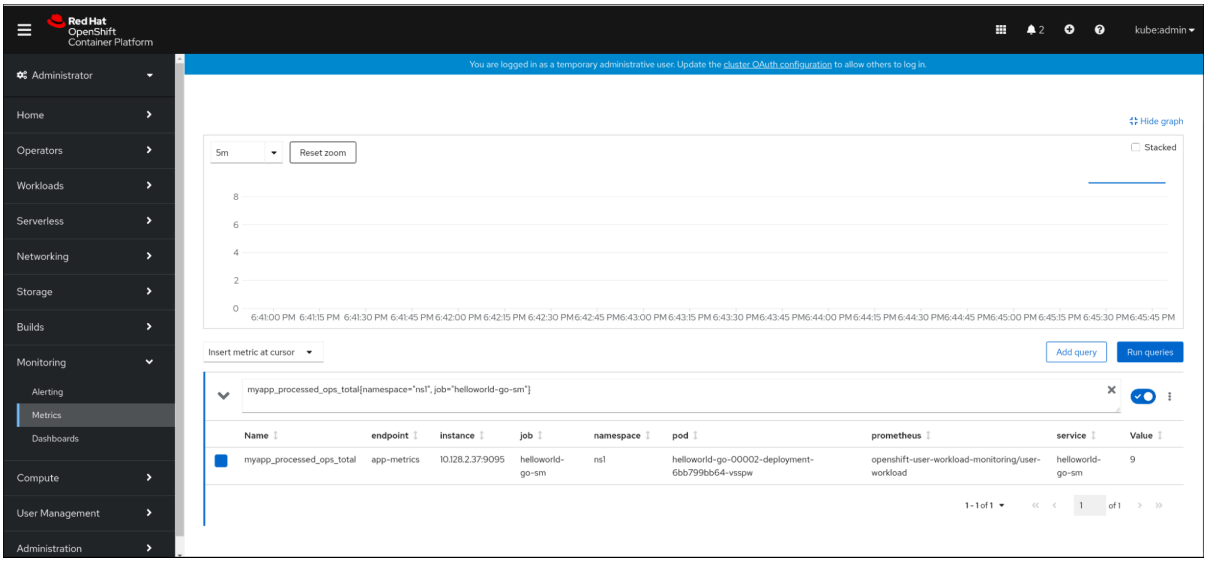
다른 예:

```
myapp_processed_ops_total{namespace="ns1", job="helloworld-go-sm"}
```

4.

시각화된 메트릭을 모니터링합니다.





8.2.5.1. 대기열 프록시 메트릭

*각 **Knative** 서비스에는 애플리케이션 컨테이너에 대한 연결을 프록시하는 프록시 컨테이너가 있습니다. 큐 프록시 성능에 대해 여러 메트릭이 보고됩니다.*

다음 메트릭을 사용하여 요청이 프록시 측에 대기되었는지 및 애플리케이션에서 요청을 처리할 때의 실제 지연을 측정할 수 있습니다.

메트릭 이름	설명	유형	태그	단위

메트릭 이름	설명	유형	태그	단위
revision_request_count	**queue-proxy** Pod로 라우팅되는 요청 수입니다.	카운터	**configuration_name**, **container_name**, **namespace_name**, **pod_name**, **response_code**, **response_code_class**, **revision_name**, **service_name**	정수(단위 없음)
revision_request_latencies	수정 버전 요청의 응답 시간입니다.	히스토그램	**configuration_name**, **container_name**, **namespace_name**, **pod_name**, **response_code**, **response_code_class**, **revision_name**, **service_name**	밀리초
revision_app_request_count	**user-container** pod로 라우팅되는 요청 수입니다.	카운터	**configuration_name**, **container_name**, **namespace_name**, **pod_name**, **response_code**, **response_code_class**, **revision_name**, **service_name**	정수(단위 없음)
revision_app_request_latencies	수정 버전 앱 요청의 응답 시간입니다.	히스토그램	**configuration_name**, **namespace_name**, **pod_name**, **response_code**, **response_code_class**, **revision_name**, **service_name**	밀리초

메트릭 이름	설명	유형	태그	단위
revision_queue_depth	**serving** 및 **waiting** 대기열의 현재 항목 수입니다. 무제한 동시 실행이 구성된 경우 이 메트릭은 보고되지 않습니다.	게이지	**configuration_name**, **event-display**, **container_name**, **namespace_name**, **pod_name**, **response_code_class**, **revision_name**, **service_name**	정수(단위 없음)

8.2.6. 서비스 메트릭 대시보드

네임스페이스별로 큐 프록시 메트릭을 집계하는 전용 대시보드를 사용하여 메트릭을 검사할 수 있습니다.

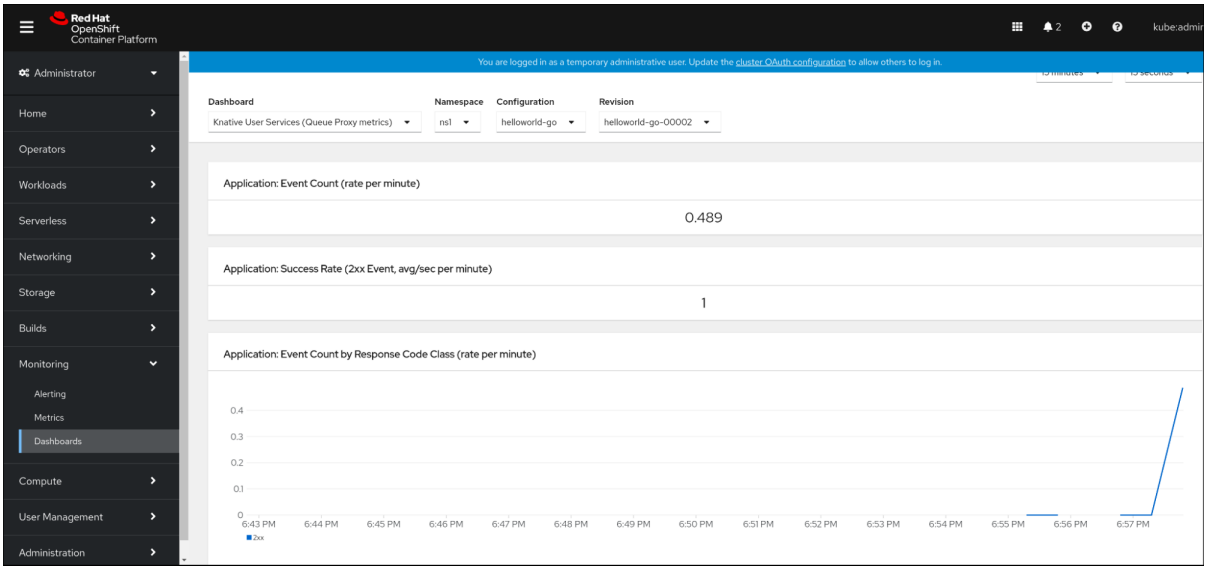
8.2.6.1. 대시보드에서 서비스의 메트릭 검사

사전 요구 사항

- ***OpenShift Container Platform** 웹 콘솔에 로그인했습니다.*
- ***OpenShift Serverless Operator** 및 **Knative Serving**이 설치되어 있습니다.*

절차

1. *웹 콘솔에서 **Observe** → **Metrics** 인터페이스로 이동합니다.*
2. ***Knative** 사용자 서비스(**Queue** 프록시 메트릭) 대시보드를 선택합니다.*
3. *애플리케이션에 해당하는 네임스페이스,구성 및 개정 버전을 선택합니다.*
4. *시각화된 메트릭을 모니터링합니다.*



8.3. 클러스터 로깅

8.3.1. OpenShift Serverless에서 OpenShift Logging 사용

8.3.1.1. Red Hat OpenShift의 로깅 하위 시스템 배포 정보

OpenShift Container Platform 클러스터 관리자는 OpenShift Container Platform 웹 콘솔 또는 CLI를 사용하여 로깅 하위 시스템을 배포하여 OpenShift Elasticsearch Operator 및 Red Hat OpenShift Logging Operator를 설치할 수 있습니다. Operator가 설치되면 ClusterLogging 사용자 정의 리소스(CR)를 생성하여 로깅 하위 시스템 Pod 및 로깅 하위 시스템을 지원하는 데 필요한 기타 리소스를 예약합니다. Operator는 로깅 하위 시스템의 배포, 업그레이드 및 유지 관리를 담당합니다.

ClusterLogging CR은 로그를 수집, 저장 및 시각화하기 위해 로깅 스택의 모든 구성 요소를 포함하는 완전한 로깅 하위 시스템 환경을 정의합니다. Red Hat OpenShift Logging Operator는 로깅 하위 시스템 CR을 감시하고 그에 따라 로깅 배포를 조정합니다.

관리자와 애플리케이션 개발자는 보기 권한이 있는 프로젝트의 로그를 볼 수 있습니다.

8.3.1.2. Red Hat OpenShift의 로깅 하위 시스템 배포 및 구성 정보

로깅 하위 시스템은 중소 규모의 OpenShift Container Platform 클러스터에 맞게 튜닝된 기본 구성과 함께 사용하도록 설계되었습니다.

다음 설치 명령에는 로깅 하위 시스템 인스턴스를 생성하고 로깅 하위 시스템 환경을 구성하는 데 사용할 수 있는 샘플 ClusterLogging 사용자 정의 리소스(CR)가 포함되어 있습니다.

기본 로깅 하위 시스템 설치를 사용하려면 샘플 **CR**을 직접 사용할 수 있습니다.

배포를 사용자 정의하려면 필요에 따라 샘플 **CR**을 변경합니다. 이어지는 내용에서는 **OpenShift Logging** 인스턴스를 설치할 때 수행하거나 설치 후 수정할 수 있는 구성에 대해 설명합니다. **ClusterLogging** 사용자 정의 리소스 외부에서 수행할 수 있는 수정을 포함하여 각 구성 요소의 작업에 대한 자세한 내용은 구성 섹션을 참조하십시오.

8.3.1.2.1. 로깅 하위 시스템 구성 및 튜닝

openshift-logging 프로젝트에 배포된 **ClusterLogging** 사용자 정의 리소스를 수정하여 로깅 하위 시스템을 구성할 수 있습니다.

설치 시 또는 설치 후 다음 구성 요소를 수정할 수 있습니다.

메모리 및 **CPU**

유효한 메모리 및 **CPU** 값으로 **resources** 블록을 수정하여 각 구성 요소의 **CPU** 및 메모리 제한을 모두 조정할 수 있습니다.

```
spec:
  logStore:
    elasticsearch:
      resources:
        limits:
          cpu:
          memory: 16Gi
        requests:
          cpu: 500m
          memory: 16Gi
      type: "elasticsearch"
  collection:
    logs:
      fluentd:
        resources:
          limits:
            cpu:
            memory:
          requests:
            cpu:
            memory:
        type: "fluentd"
  visualization:
    kibana:
      resources:
        limits:
          cpu:
          memory:
```

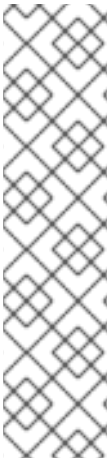
```
      requests:
        cpu:
        memory:
      type: kibana
```

Elasticsearch 스토리지

storageClass name 및 size 매개변수를 사용하여 Elasticsearch 클러스터의 영구 스토리지 클래스 및 크기를 구성할 수 있습니다. Red Hat OpenShift Logging Operator는 이러한 매개변수를 기반으로 Elasticsearch 클러스터의 각 데이터 노드에 대한 PVC(영구 볼륨 클레임)를 생성합니다.

```
spec:
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 3
      storage:
        storageClassName: "gp2"
        size: "200G"
```

이 예제에서는 클러스터의 각 데이터 노드가 "gp2" 스토리지의 "200G"를 요청하는 PVC에 바인딩되도록 지정합니다. 각 기본 분할은 단일 복제본에서 지원합니다.



참고

storage 블록을 생략하면 임시 스토리지만 포함하는 배포가 생성됩니다.

```
spec:
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 3
      storage: {}
```

Elasticsearch 복제 정책

Elasticsearch 분할이 클러스터의 여러 데이터 노드에 복제되는 방법을 정의하는 정책을 설정할 수 있습니다.

- FullRedundancy. 각 인덱스의 분할이 모든 데이터 노드에 완전히 복제됩니다.
- MultipleRedundancy. 각 인덱스의 분할이 데이터 노드의 1/2에 걸쳐 있습니다.
-

SingleRedundancy. 각 분할의 단일 사본입니다. 두 개 이상의 데이터 노드가 존재하는 한 항상 로그를 사용할 수 있고 복구할 수 있습니다.

- **ZeroRedundancy.** 분할 복사본이 없습니다. 노드가 다운되거나 실패하는 경우 로그를 사용할 수 없거나 로그가 손실될 수 있습니다.

8.3.1.2.2. 수정된 ClusterLogging 사용자 정의 리소스 샘플

다음은 이전에 설명한 옵션을 사용하여 수정한 **ClusterLogging** 사용자 정의 리소스의 예입니다.

수정된 **ClusterLogging** 사용자 정의 리소스 샘플

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"
  namespace: "openshift-logging"
spec:
  managementState: "Managed"
  logStore:
    type: "elasticsearch"
    retentionPolicy:
      application:
        maxAge: 1d
      infra:
        maxAge: 7d
      audit:
        maxAge: 7d
    elasticsearch:
      nodeCount: 3
      resources:
        limits:
          memory: 32Gi
        requests:
          cpu: 3
          memory: 32Gi
        storage:
          storageClassName: "gp2"
          size: "200G"
      redundancyPolicy: "SingleRedundancy"
  visualization:
    type: "kibana"
    kibana:
      resources:
        limits:
          memory: 1Gi
        requests:
```

```
        cpu: 500m
        memory: 1Gi
    replicas: 1
collection:
  logs:
    type: "fluentd"
    fluentd:
      resources:
        limits:
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 1Gi
```

8.3.2. Knative Serving 구성 요소의 로그 검색

다음 절차를 사용하여 **Knative Serving** 구성 요소의 로그를 찾을 수 있습니다.

8.3.2.1. OpenShift 로깅을 사용하여 Knative Serving 구성 요소 로그 찾기

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.

절차

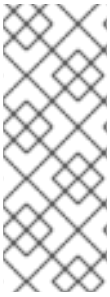
1. **Kibana** 경로를 가져옵니다.

```
$ oc -n openshift-logging get route kibana
```

2. 경로의 **URL**을 사용하여 **Kibana** 대시보드로 이동한 후 로그인합니다.

3. 인덱스가 **.all**로 설정되어 있는지 확인합니다. 인덱스가 **.all**로 설정되어 있지 않으면 **OpenShift Container Platform** 시스템 로그만 나열됩니다.

4. **knative-serving** 네임스페이스를 사용하여 로그를 필터링합니다. 검색 상자에 **kubernetes.namespace_name:knative-serving**을 입력하여 결과를 필터링합니다.



참고

Knative Serving에서는 기본적으로 구조화된 로깅을 사용합니다. OpenShift Logging Fluentd 설정을 사용자 정의하여 이러한 로그의 구문 분석을 활성화할 수 있습니다. 이렇게 하면 로그를 더 쉽게 검색할 수 있고 로그 수준에서 필터링하여 문제를 빠르게 확인할 수 있습니다.

8.3.3. Knative Serving 서비스의 로그 찾기

다음 절차를 사용하여 Knative Serving 서비스의 로그를 확인할 수 있습니다.

8.3.3.1. OpenShift Logging을 사용하여 Knative Serving으로 배포한 서비스의 로그 찾기

OpenShift Logging을 사용하면 애플리케이션에서 콘솔에 쓰는 로그가 Elasticsearch에 수집됩니다. 다음 절차에서는 Knative Serving을 사용하여 배포한 애플리케이션에 이러한 기능을 적용하는 방법을 간략하게 설명합니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.

절차

1. Kibana 경로를 가져옵니다.

```
$ oc -n openshift-logging get route kibana
```

2. 경로의 URL을 사용하여 Kibana 대시보드로 이동한 후 로그인합니다.

3. 인덱스가 .all로 설정되어 있는지 확인합니다. 인덱스가 .all로 설정되어 있지 않으면 OpenShift 시스템 로그만 나열됩니다.

4. knative-serving 네임스페이스를 사용하여 로그를 필터링합니다. 검색 상자에 서비스 필터를 입력하여 결과를 필터링합니다.

필터 예제

```
kubernetes.namespace_name:default AND
kubernetes.labels.serving_knative_dev\/service:{service_name}
```

/configuration 또는 /revision을 사용하여 필터링할 수도 있습니다.

5. 애플리케이션에서 생성한 로그만 표시하려면 kubernetes.container_name: <user_container>를 사용하여 검색 범위를 좁힙니다. 그러지 않으면 큐 프록시의 로그가 표시됩니다.

참고

애플리케이션에서 JSON 기반 구조의 로깅을 사용하면 프로덕션 환경에서 이러한 로그를 빠르게 필터링할 수 있습니다.

8.4. 추적

8.4.1. 요청 추적

분산 추적은 애플리케이션을 구성하는 다양한 서비스를 통해 요청 경로를 기록합니다. 이 기능은 분산 트랜잭션의 전체 이벤트 체인을 이해하기 위해 서로 다른 작업 단위에 대한 정보를 함께 연결하는 데 사용됩니다. 작업 단위는 여러 프로세스 또는 호스트에서 실행될 수 있습니다.

8.4.1.1. 분산 추적 개요

서비스 소유자로 분산 추적을 사용하여 서비스 아키텍처에 대한 정보를 수집할 수 있습니다. 분산 추적을 사용하여 최신 클라우드 네이티브, 마이크로서비스 기반 애플리케이션의 구성 요소 간 상호 작용을 모니터링, 네트워크 프로파일링 및 문제 해결할 수 있습니다.

분산 추적을 사용하면 다음 기능을 수행할 수 있습니다.

- 분산 트랜잭션 모니터링
- 성능 및 대기 시간 최적화

- 근본 원인 분석 수행

Red Hat OpenShift distributed tracing은 다음 두 가지 주요 구성 요소로 구성됩니다.

- Red Hat OpenShift distributed tracing platform - 이 구성 요소는 오픈 소스 Jaeger 프로젝트를 기반으로 합니다.
- Red Hat OpenShift 분산 추적 데이터 수집 - 이 구성 요소는 오픈 소스 OpenTelemetry 프로젝트를 기반으로 합니다.

이러한 두 구성 요소는 모두 벤더 중립 OpenTracing API 및 계측을 기반으로 합니다.

8.4.1.2. 추가 리소스

- Red Hat OpenShift distributed tracing 아키텍처
- 분산 추적 설치

8.4.2. Red Hat OpenShift distributed tracing 사용

OpenShift Serverless와 함께 Red Hat OpenShift distributed tracing을 사용하여 서버리스 애플리케이션을 모니터링하고 문제를 해결할 수 있습니다.

8.4.2.1. Red Hat OpenShift distributed tracing을 사용하여 분산 추적 활성화

Red Hat OpenShift distributed tracing은 추적 데이터를 수집, 저장 및 표시하기 위해 함께 작동하는 여러 구성 요소로 구성됩니다.

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- OpenShift Serverless Operator, Knative Serving 및 Knative Eventing을 아직 설치하지

않았습니다. Red Hat OpenShift distributed tracing 설치 후 설치해야 합니다.

- OpenShift Container Platform "분산 추적 설치" 문서에 따라 Red Hat OpenShift distributed tracing을 설치했습니다.
- OpenShift CLI(oc)가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

절차

1. OpenTelemetryHeaderor CR(사용자 정의 리소스)을 만듭니다.

OpenTelemetry#189or CR의 예

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: cluster-collector
  namespace: <namespace>
spec:
  mode: deployment
  config: |
    receivers:
      zipkin:
    processors:
    exporters:
      jaeger:
        endpoint: jaeger-all-in-one-inmemory-collector-headless.tracing-
system.svc:14250
        tls:
          ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
      logging:
    service:
      pipelines:
        traces:
          receivers: [zipkin]
          processors: []
          exporters: [jaeger, logging]
```

2. *Red Hat OpenShift distributed tracing이 설치된 네임스페이스에서 두 개의 포드가 실행되고 있는지 확인합니다.*

```
$ oc get pods -n <namespace>
```

출력 예

```
NAME                                  READY   STATUS    RESTARTS   AGE
cluster-collector-collector-85c766b5c-b5g99   1/1     Running   0          5m56s
jaeger-all-in-one-inmemory-ccbc9df4b-ndkl5    2/2     Running   0          15m
```

3. *다음과 같은 헤드리스 서비스가 생성되었는지 확인합니다.*

```
$ oc get svc -n <namespace> | grep headless
```

출력 예

```
cluster-collector-collector-headless          ClusterIP   None            <none>
9411/TCP                             7m28s
jaeger-all-in-one-inmemory-collector-headless   ClusterIP   None            <none>
9411/TCP,14250/TCP,14267/TCP,14268/TCP   16m
```

이러한 서비스는 Jaeger, Knative Serving 및 Knative Eventing을 구성하는 데 사용됩니다. Jaeger 서비스의 이름은 다를 수 있습니다.

4. *"OpenShift Serverless Operator 설치" 문서에 따라 OpenShift Serverless Operator를 설치합니다.*

5. *다음 KnativeServing CR을 생성하여 Knative Serving 을 설치합니다.*

KnativeServing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
 config:
  tracing:
   backend: "zipkin"
   zipkin-endpoint: "http://cluster-collector-collector-headless.tracing-
system.svc:9411/api/v2/spans"
   debug: "false"
   sample-rate: "0.1" 1
```

1

*sample-rate*는 샘플링 가능성을 정의합니다. **sample-rate: "0.1"** 을 사용하면 **10**개의 추적 중 **1**개가 샘플링됩니다.

6. 다음 **KnativeEventing CR**을 생성하여 **Knative Eventing**을 설치합니다.

KnativeEventing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
 config:
  tracing:
   backend: "zipkin"
   zipkin-endpoint: "http://cluster-collector-collector-headless.tracing-
system.svc:9411/api/v2/spans"
   debug: "false"
   sample-rate: "0.1" 1
```

1

7. *Knative 서비스를 생성합니다.*

서비스의 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: helloworld-go
spec:
  template:
    metadata:
      labels:
        app: helloworld-go
      annotations:
        autoscaling.knative.dev/minScale: "1"
        autoscaling.knative.dev/target: "1"
    spec:
      containers:
      - image: quay.io/openshift-knative/helloworld:v1.2
        imagePullPolicy: Always
        resources:
          requests:
            cpu: "200m"
        env:
        - name: TARGET
          value: "Go Sample v1"
```

8. *서비스에 대한 일부 요청을 수행합니다.*

HTTPS 요청의 예

```
$ curl https://helloworld-go.example.com
```

9. Jaeger 웹 콘솔의 URL을 가져옵니다.

명령 예

```
$ oc get route jaeger-all-in-one-inmemory  -o jsonpath='{.spec.host}' -n <namespace>
```

이제 Jaeger 콘솔을 사용하여 추적을 검사할 수 있습니다.

8.4.3. Jaeger 분산 추적 사용

Red Hat OpenShift distributed tracing의 모든 구성 요소를 설치하지 않으려면 OpenShift Serverless에서 OpenShift Container Platform에서 분산 추적을 사용할 수 있습니다.

8.4.3.1. 분산 추적을 활성화하도록 Jaeger 구성

Jaeger를 사용하여 분산 추적을 활성화하려면 독립 실행형 통합으로 Jaeger를 설치하고 구성해야 합니다.

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- OpenShift Serverless Operator, Knative Serving 및 Knative Eventing을 설치했습니다.
- Red Hat OpenShift distributed tracing Platform Operator를 설치했습니다.
- OpenShift CLI(oc)가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

절차

1. *다음을 포함하는 Jaeger 사용자 정의 리소스 (CR) 파일을 생성하고 적용합니다.*

Jaeger CR

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger
  namespace: default
```

2. *KnativeServing CR을 편집하고 추적에 필요한 YAML 구성을 추가하여 Knative Serving에 대한 추적을 활성화합니다.*

Serving의 YAML 추적 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  config:
    tracing:
      sample-rate: "0.1" 1
      backend: zipkin 2
      zipkin-endpoint: "http://jaeger-
collector.default.svc.cluster.local:9411/api/v2/spans" 3
      debug: "false" 4
```

1 *sample-rate는 샘플링 가능성을 정의합니다. sample-rate: "0.1" 을 사용하면 10개의 추적 중 1개가 샘플링됩니다.*

2

3

zipkin-endpoint는 jaeger-collector 서비스 끝점을 가리켜야 합니다. 이 끝점을 가져오려면 Jaeger CR이 적용되는 네임스페이스를 대체합니다.

4

디버깅을 false로 설정해야 합니다. debug: "true"를 설정하여 디버그 모드를 활성화하면 모든 범위가 서버에 전송되어 샘플링 단계를 건너뜁니다.

3. *KnativeEventing CR을 편집하여 Knative Eventing에 대한 추적을 활성화합니다.*

Eventing의 YAML 추적 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  config:
    tracing:
      sample-rate: "0.1" 1
      backend: zipkin 2
      zipkin-endpoint: "http://jaeger-
collector.default.svc.cluster.local:9411/api/v2/spans" 3
      debug: "false" 4
```

1

sample-rate는 샘플링 가능성을 정의합니다. sample-rate: "0.1" 을 사용하면 10개의 추적 중 1개가 샘플링됩니다.

2

backend 를 zipkin 으로 설정합니다.

3

4

디버깅을 **false**로 설정해야 합니다. **debug: "true"**를 설정하여 디버그 모드를 활성화하면 모든 범위가 서버에 전송되어 샘플링 단계를 건너뜁니다.

검증

jaeger 경로를 사용하여 **Jaeger** 웹 콘솔에 액세스하여 추적 데이터를 볼 수 있습니다.

1. 다음 명령을 입력하여 **jaeger** 경로의 호스트 이름을 가져옵니다.

```
$ oc get route jaeger -n default
```

출력 예

```
NAME     HOST/PORT                         PATH   SERVICES       PORT    TERMINATION
WILDCARD
jaeger   jaeger-default.apps.example.com          jaeger-query   <all>   reencrypt    None
```

2. 콘솔을 보려면 브라우저에서 끝점 주소를 엽니다.

9장. 통합

9.1. OPENSHIFT SERVERLESS와 SERVICE MESH 통합

OpenShift Serverless Operator는 Kourier를 Knative의 기본 수신으로 제공합니다. 그러나 Kourier가 활성화되어 있는지 여부에 관계없이 OpenShift Serverless에서 Service Mesh를 사용할 수 있습니다. Kourier disabled와 통합하면 mTLS 기능과 같이 Kourier 인그레스에서 지원하지 않는 추가 네트워킹 및 라우팅 옵션을 구성할 수 있습니다.

중요

OpenShift Serverless는 본 안내서에 명시되어 있는 Red Hat OpenShift Service Mesh 기능만 지원하며 문서화되지 않은 다른 기능은 지원하지 않습니다.

9.1.1. 사전 요구 사항

- 다음 절차의 예제에서는 도메인 example.com을 사용합니다. 이 도메인에 사용되는 예제 인증서는 하위 도메인 인증서에 서명하는 CA(인증 기관)로 사용됩니다.

  배포에서 이 절차를 완료하고 확인하려면 널리 신뢰받는 공용 CA에서 서명한 인증서 또는 조직에서 제공하는 CA가 필요합니다. 도메인, 하위 도메인, CA에 따라 예제 명령을 조정해야 합니다.

- OpenShift Container Platform 클러스터의 도메인과 일치하도록 와일드카드 인증서를 구성해야 합니다. 예를 들어 OpenShift Container Platform 콘솔 주소가 https://console-openshift-console.apps.openshift.example.com인 경우 도메인이 *.apps.openshift.example.com이 되도록 와일드카드 인증서를 구성해야 합니다. 와일드카드 인증서 구성에 대한 자세한 내용은 수신되는 외부 트래픽을 암호화하기 위한 인증서 생성에 관한 다음의 내용을 참조하십시오.

- 기본 OpenShift Container Platform 클러스터 도메인의 하위 도메인이 아닌 도메인 이름을 사용하려면 해당 도메인에 대한 도메인 매핑을 설정해야 합니다. 자세한 내용은 사용자 정의 도메인 매핑 생성에 대한 OpenShift Serverless 설명서를 참조하십시오.

9.1.2. 수신 외부 트래픽을 암호화하기 위한 인증서 생성

기본적으로 Service Mesh mTLS 기능은 사이드카가 있는 수신 게이트웨이와 개별 Pod 간에 Service Mesh 자체의 트래픽만 보호합니다. OpenShift Container Platform 클러스터로 전달될 때 트래픽을 암호화하려면 OpenShift Serverless 및 Service Mesh 통합을 활성화하기 전에 인증서를 생성해야 합니다.

사전 요구 사항

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

절차

1. **Knative** 서비스의 인증서에 서명할 **root** 인증서 및 개인 키를 생성합니다.

```
$ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 \
  -subj '/O=Example Inc./CN=example.com' \
  -keyout root.key \
  -out root.crt
```

2. 와일드카드 인증서를 만듭니다.

```
$ openssl req -nodes -newkey rsa:2048 \
  -subj "/CN=*.apps.openshift.example.com/O=Example Inc." \
  -keyout wildcard.key \
  -out wildcard.csr
```

3. 와일드카드 인증서를 서명합니다.

```
$ openssl x509 -req -days 365 -set_serial 0 \
  -CA root.crt \
  -CAkey root.key \
  -in wildcard.csr \
  -out wildcard.crt
```

4. 와일드카드 인증서를 사용하여 시크릿을 생성합니다.

```
$ oc create -n istio-system secret tls wildcard-certs \
  --key=wildcard.key \
  --cert=wildcard.crt
```

-

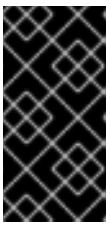
이 인증서는 **OpenShift Serverless**를 **Service Mesh**와 통합할 때 생성된 게이트웨이에서 선택하여 수신 게이트웨이가 이 인증서와 트래픽을 제공하도록 합니다.

9.1.3. OpenShift Serverless와 Service Mesh 통합

Kourier를 기본 수신으로 사용하지 않고 **OpenShift Serverless**와 **Service Mesh**를 통합할 수 있습니다. 이렇게 하려면 다음 절차를 완료하기 전에 **Knative Serving** 구성 요소를 설치하지 마십시오. 일반 **Knative Serving** 설치 절차에서는 **Knative Serving**을 **Service Mesh**와 통합하기 위해 **KnativeServing CRD(사용자 정의 리소스 정의)**를 생성할 때 추가 단계가 필요합니다. 이 절차는 **Service Mesh**를 기본값으로 통합하고 **OpenShift Serverless** 설치에만 수신하려는 경우에 유용할 수 있습니다.

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 **OpenShift Container Platform** 계정에 액세스할 수 있습니다.
- 프로젝트를 생성했거나 **OpenShift Container Platform**에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- **Red Hat OpenShift Service Mesh Operator**를 설치하고 **istio-system** 네임스페이스에 **ServiceMeshControlPlane** 리소스를 생성합니다. **mTLS** 기능을 사용하려면 **ServiceMeshControlPlane** 리소스의 **spec.security.dataPlane.mtls** 필드도 **true** 로 설정해야 합니다.



중요

Service Mesh에서 **OpenShift Serverless** 사용은 **Red Hat OpenShift Service Mesh** 버전 **2.0.5** 이상에서만 지원됩니다.

- **OpenShift Serverless Operator**를 설치합니다.
- **OpenShift CLI(oc)**를 설치합니다.

절차

1. **Service Mesh**와 통합할 네임스페이스를 **ServiceMeshMemberRoll** 오브젝트에 멤버로 추가합니다.

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system
spec:
  members: 1
    - knative-serving
    - <namespace>
```

1

Service Mesh와 통합할 네임스페이스 목록입니다.

중요

이 네임스페이스 목록에 knative-serving 네임스페이스가 포함되어야 합니다.

2. *ServiceMeshMemberRoll 리소스를 적용합니다.*

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

3. *Service Mesh가 트래픽을 수락할 수 있도록 필요한 게이트웨이를 생성합니다.*

HTTP를 사용하는 knative-local-gateway 오브젝트의 예

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: knative-ingress-gateway
  namespace: knative-serving
spec:
  selector:
    istio: ingressgateway
  servers:
    - port:
        number: 443
        name: https
        protocol: HTTPS
      hosts:
        - "*"
```

```
    tls:
      mode: SIMPLE
      credentialName: <wildcard_certs> (1)
---
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
 name: knative-local-gateway
 namespace: knative-serving
spec:
 selector:
   istio: ingressgateway
 servers:
   - port:
       number: 8081
       name: http
       protocol: HTTP (2)
     hosts:
       - "*"
---
apiVersion: v1
kind: Service
metadata:
 name: knative-local-gateway
 namespace: istio-system
 labels:
   experimental.istio.io/disable-gateway-port-translation: "true"
spec:
 type: ClusterIP
 selector:
   istio: ingressgateway
 ports:
   - name: http2
     port: 80
     targetPort: 8081
```

(1) 와일드카드 인증서가 포함된 시크릿 이름을 추가합니다.

(2) knative-local-gateway는 HTTP 트래픽을 제공합니다. HTTP를 사용하면 Service Mesh 외부에서 들어오는 트래픽이 표시되지만 example.default.svc.cluster.local과 같은 내부 호스트 이름을 사용하는 것은 암호화되지 않습니다. 다른 와일드카드 인증서 및 다른 protocol 사양을 사용하는 추가 게이트웨이를 생성하여 이 경로에 대한 암호화를 설정할 수 있습니다.

HTTPS를 사용하는 knative-local-gateway 오브젝트의 예

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: knative-local-gateway
  namespace: knative-serving
spec:
  selector:
    istio: ingressgateway
  servers:
    - port:
        number: 443
        name: https
        protocol: HTTPS
      hosts:
        - "*"
      tls:
        mode: SIMPLE
        credentialName: <wildcard_certs>
```

4. *Gateway 리소스를 적용합니다.*

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

5. *다음 KnativeServing CRD(사용자 정의 리소스 정의)를 생성하여 Knative Serving을 설치하여 Istio 통합을 활성화합니다.*

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  ingress:
    istio:
      enabled: true 1
  deployments: 2
  - name: activator
    annotations:
      "sidecar.istio.io/inject": "true"
      "sidecar.istio.io/rewriteAppHTTPProbers": "true"
  - name: autoscaler
    annotations:
      "sidecar.istio.io/inject": "true"
      "sidecar.istio.io/rewriteAppHTTPProbers": "true"
```

1

Istio 통합을 활성화합니다.

2

Knative Serving 데이터 플레인 Pod에 사이드카 삽입을 활성화합니다.

6. *KnativeServing 리소스를 적용합니다.*

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

7. *사이드카 삽입이 활성화되고 패스쓰루(pass-through) 경로를 사용하는 Knative 서비스를 생성합니다.*

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
 name: <service_name>
 namespace: <namespace> 1
 annotations:
  serving.knative.openshift.io/enablePassthrough: "true" 2
spec:
 template:
  metadata:
   annotations:
    sidecar.istio.io/inject: "true" 3
    sidecar.istio.io/rewriteAppHTTPProbers: "true"
  spec:
   containers:
   - image: <image_url>
```

1

Service Mesh 멤버 롤에 포함된 네임스페이스입니다.

2

생성한 인증서가 수신 게이트웨이를 통해 직접 제공되도록 Knative Serving에 OpenShift Container Platform 패스스루 지원 경로를 생성하도록 지시합니다.

3

Service Mesh 사이드카를 Knative 서비스 Pod에 삽입합니다.

8. *Service 리소스를 적용합니다.*

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

검증

- *CA에서 신뢰하는 보안 연결을 사용하여 서버리스 애플리케이션에 액세스합니다.*

```
$ curl --cacert root.crt <service_url>
```

명령 예

```
$ curl --cacert root.crt https://hello-default.apps.openshift.example.com
```

출력 예

```
Hello Openshift!
```

9.1.4. mTLS와 함께 서비스 메시를 사용할 때 Knative Serving 메트릭 활성화

Service Mesh가 mTLS를 사용하여 활성화된 경우 Service Mesh가 Prometheus가 메트릭을 스크랩하지 못하도록 하므로 기본적으로 Knative Serving에 대한 메트릭이 비활성화됩니다. 이 섹션에서는 서비스 메시 및 mTLS를 사용할 때 Knative Serving 지표를 활성화하는 방법을 보여줍니다.

사전 요구 사항

- *OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.*
- *mTLS 기능이 활성화된 Red Hat OpenShift Service Mesh를 설치했습니다.*

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.

- OpenShift CLI(oc)를 설치합니다.

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

절차

1. Knative Serving 사용자 정의 리소스(CR)의 observability 사양에서 prometheus를 metrics.backend-destination으로 지정합니다.

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    observability:
      metrics.backend-destination: "prometheus"
...
```

이 단계에서는 메트릭이 기본적으로 비활성화되지 않습니다.

2. Prometheus 네임스페이스의 트래픽을 허용하려면 다음 네트워크 정책을 적용합니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-monitoring-ns
  namespace: knative-serving
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: "openshift-monitoring"
  podSelector: {}
...
```

3. 다음 사양을 포함하도록 istio-system 네임스페이스에서 기본 서비스 메시 컨트롤 플레인을

수정하고 다시 적용합니다.

```
...
spec:
  proxy:
    networking:
      trafficControl:
        inbound:
          excludedPorts:
          - 8444
...
```

9.1.5. Kourier가 활성화된 경우 OpenShift Serverless와 Service Mesh 통합

Kourier가 이미 활성화된 경우에도 OpenShift Serverless에서 Service Mesh를 사용할 수 있습니다. 이 절차는 Kourier를 사용하여 Knative Serving을 이미 설치했지만 나중에 서비스 메시 통합을 추가하려는 경우 유용할 수 있습니다.

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- OpenShift CLI(oc)를 설치합니다.
- 클러스터에 OpenShift Serverless Operator 및 Knative Serving을 설치합니다.
- Red Hat OpenShift Service Mesh를 설치합니다. OpenShift Serverless with Service Mesh and Kourier는 Red Hat OpenShift Service Mesh 버전 1.x 및 2.x에서 모두 사용이 지원되고 있습니다.

절차

1. Service Mesh와 통합할 네임스페이스를 ServiceMeshMemberRoll 오브젝트에 멤버로 추가합니다.

```
apiVersion: maistra.io/v1
```

```
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system
spec:
  members:
    - <namespace> 1
...
```

1

Service Mesh와 통합할 네임스페이스 목록입니다.

2. *ServiceMeshMemberRoll 리소스를 적용합니다.*

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

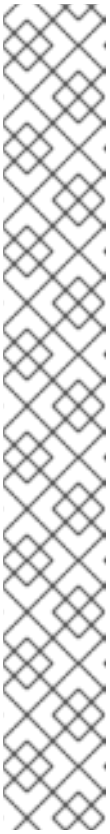
3. *Knative 시스템 Pod에서 Knative 서비스로의 트래픽 흐름을 허용하는 네트워크 정책을 생성합니다.*

   a. *Service Mesh와 통합할 각 네임스페이스에 NetworkPolicy 리소스를 생성합니다.*

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-serving-system-namespace
  namespace: <namespace> 1
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          knative.openshift.io/part-of: "openshift-serverless"
  podSelector: {}
  policyTypes:
  - Ingress
...
```

1

Service Mesh와 통합할 네임스페이스를 추가합니다.



참고

knative.openshift.io/part-of: "openshift-serverless" 레이블이 OpenShift Serverless 1.22.0에 추가되었습니다. OpenShift Serverless 1.21.1 이하를 사용하는 경우 knative.openshift.io/part-of 레이블을 knative-serving 및 knative-serving-ingress 네임스페이스에 추가합니다.

knative-serving 네임스페이스에 라벨을 추가합니다.

```
$ oc label namespace knative-serving knative.openshift.io/part-of=openshift-serverless
```

knative-serving-ingress 네임스페이스에 레이블을 추가합니다.

```
$ oc label namespace knative-serving-ingress knative.openshift.io/part-of=openshift-serverless
```

b. NetworkPolicy 리소스를 적용합니다.

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

9.1.6. Service Mesh에 대한 시크릿 필터링을 사용하여 메모리 사용량 개선

기본적으로 Kubernetes 클라이언트-go 라이브러리에 대한 정보자 구현에서는 특정 유형의 모든 리소스를 가져옵니다. 이로 인해 많은 리소스를 사용할 수 있을 때 상당한 오버헤드가 발생하여 메모리 누수로 인해 Knative net-istio Ingress 컨트롤러가 대규모 클러스터에서 실패할 수 있습니다. 그러나 컨트롤러가 Knative 관련 보안만 가져올 수 있도록 Knative net-istio 수신 컨트롤러에서 필터링 메커니즘을 사용할 수 있습니다. KnativeServing 사용자 정의 리소스(CR)에 주석을 추가하여 이 메커니즘을 활성화할 수 있습니다.

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Red Hat OpenShift Service Mesh를 설치합니다. OpenShift Serverless with Service

Mesh는 Red Hat OpenShift Service Mesh 버전 2.0.5 이상에서만 지원됩니다.

- OpenShift Serverless Operator 및 Knative Serving을 설치합니다.
- OpenShift CLI(oc)를 설치합니다.

절차

- KnativeServing CR에 serverless.openshift.io/enable-secret-informer-filtering 주석을 추가합니다.

KnativeServing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
  annotations:
    serverless.openshift.io/enable-secret-informer-filtering: "true" 1
spec:
  ingress:
    istio:
      enabled: true
  deployments:
    - annotations:
        sidecar.istio.io/inject: "true"
        sidecar.istio.io/rewriteAppHTTPProbers: "true"
      name: activator
    - annotations:
        sidecar.istio.io/inject: "true"
        sidecar.istio.io/rewriteAppHTTPProbers: "true"
      name: autoscaler
```

1

이 주석을 추가하면 환경 변수 ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID=true.

9.2. COST MANAGEMENT SERVICE와 SERVERLESS 통합

Cost Management 는 클라우드 및 컨테이너의 비용을 보다 잘 이해하고 추적할 수 있는 OpenShift Container Platform 서비스입니다. 오픈 소스 Koku 프로젝트를 기반으로 합니다.

9.2.1. 사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.
- 비용 관리를 설정하고 OpenShift Container Platform 소스를 추가했습니다.

9.2.2. 비용 관리 쿼리에 레이블 사용

비용 관리에서 태그 라고도 하는 라벨은 노드, 네임스페이스 또는 Pod에 적용할 수 있습니다. 각 레이블은 키와 값 쌍입니다. 여러 레이블을 결합하여 보고서를 생성할 수 있습니다. Red Hat 하이브리드 콘솔을 사용하여 비용에 대한 보고서에 액세스할 수 있습니다.

레이블은 노드에서 네임스페이스로 상속되며 네임스페이스에서 Pod로 상속됩니다. 그러나 리소스에 이미 존재하는 경우 레이블은 재정의되지 않습니다. 예를 들어 Knative 서비스에는 기본 app=<revision_name> 레이블이 있습니다.

Knative 서비스 기본 라벨의 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
spec:
...
    labels:
      app: <revision_name>
...
```

app=my-domain 등 네임스페이스에 레이블을 정의하는 경우, app=my-domain 태그를 사용하여 애플리케이션을 쿼리할 때 cost 관리 서비스에서 app=<revision_name > 태그를 사용하여 Knative 서비스

에서 들어오는 비용을 고려하지 않습니다. 이 태그가 있는 Knative 서비스의 비용은 app=<revision_name> 태그에서 쿼리해야 합니다.

9.2.3. 추가 리소스

- 소스에 대한 태그 구성
- 비용 탐색기를 사용하여 비용을 시각화합니다.

9.3. 서버리스 애플리케이션과 함께 NVIDIA GPU 리소스 사용

NVIDIA는 OpenShift Container Platform에서 GPU 리소스 사용을 지원합니다. OpenShift Container Platform에서 GPU 리소스를 설정하는 방법에 대한 자세한 내용은 OpenShift의 GPU Operator 를 참조하십시오.

9.3.1. 서비스에 대한 GPU 요구 사항 지정

OpenShift Container Platform 클러스터에 GPU 리소스를 활성화하면 Knative(kn) CLI를 사용하여 Knative 서비스에 대한 GPU 요구 사항을 지정할 수 있습니다.

사전 요구 사항

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

참고

IBM Z 및 IBM Power에서는 NVIDIA GPU 리소스 사용이 지원되지 않습니다.

절차

1. *Knative 서비스를 생성하고 --limit nvidia.com/gpu=1 플래그를 사용하여 GPU 리소스 요구 사항 제한을 1로 설정합니다.*

```
$ kn service create hello --image <service-image> --limit nvidia.com/gpu=1
```

GPU 리소스 요구 사항 제한이 1이면 서비스의 전용 GPU 리소스가 1개임을 나타냅니다. 서비스에서는 GPU 리소스를 공유하지 않습니다. GPU 리소스가 필요한 기타 서비스는 GPU 리소스를 더 이상 사용하지 않을 때까지 기다려야 합니다.

또한 GPU가 1개로 제한되면 GPU 리소스를 2개 이상 사용하는 애플리케이션이 제한됩니다. 서비스에서 GPU 리소스를 1개 이상 요청하는 경우 GPU 리소스 요구 사항을 충족할 수 있는 노드에 배포됩니다.

2. *선택 사항: 기존 서비스의 경우 --limit nvidia.com/gpu=3 플래그를 사용하여 GPU 리소스 요구 사항 제한을 3으로 변경할 수 있습니다.*

```
$ kn service update hello --limit nvidia.com/gpu=3
```

9.3.2. 추가 리소스

- *확장 리소스에 대한 리소스 할당량 설정*

10장. 서버리스 제거

10.1. OPENSHIFT SERVERLESS 제거 개요

클러스터에서 OpenShift Serverless를 제거해야 하는 경우 OpenShift Serverless Operator 및 기타 OpenShift Serverless 구성 요소를 수동으로 제거하여 이를 수행할 수 있습니다. OpenShift Serverless Operator를 제거하려면 먼저 Knative Serving 및 Knative Eventing을 제거해야 합니다.

OpenShift Serverless를 설치 제거한 후 클러스터에 남아 있는 Operator 및 API CRD(사용자 정의 리소스 정의)를 제거할 수 있습니다.

OpenShift Serverless를 완전히 제거하는 단계는 다음 절차에 설명되어 있습니다.

- Knative Eventing 설치 제거.
- Knative Serving 설치 제거.
- OpenShift Serverless Operator 제거.
- OpenShift Serverless 사용자 정의 리소스 정의 삭제

10.2. OPENSHIFT SERVERLESS KNATIVE EVENTING 설치 제거

OpenShift Serverless Operator를 제거하려면 먼저 Knative Eventing을 제거해야 합니다. Knative Eventing을 설치 제거하려면 KnativeEventing 사용자 정의 리소스(CR)를 제거하고 knative-eventing 네임스페이스를 삭제해야 합니다.

10.2.1. Knative Eventing 설치 제거

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.

- OpenShift CLI(oc)를 설치합니다.

절차

1. KnativeEventing CR을 삭제합니다.

```
$ oc delete knativeeventings.operator.knative.dev knative-eventing -n knative-eventing
```

2. 명령이 완료되고 knative-eventing 네임스페이스에서 모든 Pod가 제거된 후 네임스페이스를 삭제합니다.

```
$ oc delete namespace knative-eventing
```

10.3. OPENSHIFT SERVERLESS KNATIVE SERVING 설치 제거

OpenShift Serverless Operator를 제거하려면 먼저 Knative Serving을 제거해야 합니다. Knative Serving을 설치 제거하려면 KnativeServing 사용자 정의 리소스(CR)를 제거하고 knative-serving 네임스페이스를 삭제해야 합니다.

10.3.1. Knative Serving 설치 제거

사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- OpenShift CLI(oc)를 설치합니다.

절차

1. KnativeServing CR을 삭제합니다.

```
$ oc delete knativeservings.operator.knative.dev knative-serving -n knative-serving
```

2. 명령이 완료되고 knative-serving 네임스페이스에서 모든 Pod가 제거되면 네임스페이스를 삭제합니다.

```
$ oc delete namespace knative-serving
```

10.4. OPENSHIFT SERVERLESS OPERATOR 제거

Knative Serving 및 Knative Eventing을 제거한 후 OpenShift Serverless Operator를 제거할 수 있습니다. OpenShift Container Platform 웹 콘솔 또는 oc CLI를 사용하여 이 작업을 수행할 수 있습니다.

10.4.1. 웹 콘솔을 사용하여 클러스터에서 Operator 삭제

클러스터 관리자는 웹 콘솔을 사용하여 선택한 네임스페이스에서 설치된 Operator를 삭제할 수 있습니다.

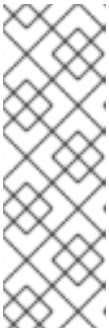
사전 요구 사항

- cluster-admin 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터 웹 콘솔에 액세스할 수 있습니다.

절차

1. Operator → 설치된 Operator 페이지로 이동합니다.
2. 제거하려는 Operator를 찾으려면 이름으로 필터링 필드에 키워드를 스크롤하거나 입력합니다. 그런 다음 해당 Operator를 클릭합니다.
3. Operator 세부 정보 페이지 오른쪽에 있는 작업 목록에서 Operator 제거를 선택합니다.

   Operator를 설치 제거하시겠습니까? 대화 상자가 표시됩니다.
4. 설치 제거를 선택하여 Operator, Operator 배포 및 Pod를 제거합니다. 이 작업 후에 Operator는 실행을 중지하고 더 이상 업데이트가 수신되지 않습니다.



참고

이 작업은 CRD(사용자 정의 리소스 정의) 및 CR(사용자 정의 리소스)을 포함하여 Operator에서 관리하는 리소스를 제거하지 않습니다. 웹 콘솔에서 활성화된 대시보드 및 탐색 항목과 계속 실행되는 클러스터 외부 리소스는 수동 정리가 필요할 수 있습니다. Operator를 설치 제거한 후 해당 항목을 제거하려면 Operator CRD를 수동으로 삭제해야 할 수 있습니다.

10.4.2. CLI를 사용하여 클러스터에서 Operator 삭제

클러스터 관리자는 CLI를 사용하여 선택한 네임스페이스에서 설치된 Operator를 삭제할 수 있습니다.

사전 요구 사항

- cluster-admin 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.
- oc 명령이 워크스테이션에 설치되어 있습니다.

절차

1. currentCSV 필드에서 구독한 Operator(예: jaeger)의 현재 버전을 확인합니다.

```
$ oc get subscription jaeger -n openshift-operators -o yaml | grep currentCSV
```

출력 예

```
currentCSV: jaeger-operator.v1.8.2
```

2. 서브스크립션을 삭제합니다(예: jaeger).

```
$ oc delete subscription jaeger -n openshift-operators
```

출력 예

```
subscription.operators.coreos.com "jaeger" deleted
```

3. 이전 단계의 currentCSV 값을 사용하여 대상 네임스페이스에서 Operator의 CSV를 삭제합니다.

```
$ oc delete clusterserviceversion jaeger-operator.v1.8.2 -n openshift-operators
```

출력 예

```
clusterserviceversion.operators.coreos.com "jaeger-operator.v1.8.2" deleted
```

10.4.3. 실패한 서브스크립션 새로 고침

OLM(Operator Lifecycle Manager)에서는 네트워크상에서 액세스할 수 없는 이미지를 참조하는 Operator를 구독하는 경우 openshift-marketplace 네임스페이스에 다음 오류로 인해 실패하는 작업을 확인할 수 있습니다.

출력 예

```
ImagePullBackOff for
Back-off pulling image "example.com/openshift4/ose-elasticsearch-operator-bundle@sha256:6d2587129c846ec28d384540322b40b05833e7e00b25cca584e004af9a1d292e"
```

출력 예

```
rpc error: code = Unknown desc = error pinging docker registry example.com: Get "https://example.com/v2/": dial tcp: lookup example.com on 10.0.0.1:53: no such host
```

결과적으로 서브스크립션이 이러한 장애 상태에 고착되어 **Operator**를 설치하거나 업그레이드할 수 없습니다.

서브스크립션, **CSV(**클러스터 서비스 버전**)** 및 기타 관련 오브젝트를 삭제하여 실패한 서브스크립션을 새로 고칠 수 있습니다. 서브스크립션을 다시 생성하면 **OLM**에서 올바른 버전의 **Operator**를 다시 설치합니다.

사전 요구 사항

- 액세스할 수 없는 번들 이미지를 가져올 수 없는 실패한 서브스크립션이 있습니다.
- 올바른 번들 이미지에 액세스할 수 있는지 확인했습니다.

절차

1. **Operator**가 설치된 네임스페이스에서 **Subscription** 및 **ClusterServiceVersion** 오브젝트의 이름을 가져옵니다.

```
$ oc get sub,csv -n <namespace>
```

출력 예

```
NAME                                         PACKAGE                  SOURCE             CHANNEL
subscription.operators.coreos.com/elasticsearch-operator   elasticsearch-operator   redhat-operators   5.0

NAME                                                  DISPLAY                            VERSION    REPLACES   PHASE
clusterserviceversion.operators.coreos.com/elasticsearch-operator.5.0.0-65   OpenShift Elasticsearch Operator   5.0.0-65              Succeeded
```

2. 서브스크립션을 삭제합니다.

-

```
$ oc delete subscription <subscription_name> -n <namespace>
```

3. 클러스터 서비스 버전을 삭제합니다.

```
$ oc delete csv <csv_name> -n <namespace>
```

4. openshift-marketplace 네임스페이스에서 실패한 모든 작업 및 관련 구성 맵의 이름을 가져옵니다.

```
$ oc get job,configmap -n openshift-marketplace
```

출력 예

```
NAME                                                                COMPLETIONS   DURATION   AGE
job.batch/1de9443b6324e629ddf31fed0a853a121275806170e34c926d69e53a7fcbccb   1/1           26s        9m30s

NAME                                                                DATA   AGE
configmap/1de9443b6324e629ddf31fed0a853a121275806170e34c926d69e53a7fcbccb   3      9m30s
```

5. 작업을 삭제합니다.

```
$ oc delete job <job_name> -n openshift-marketplace
```

이렇게 하면 액세스할 수 없는 이미지를 가져오려는 Pod가 다시 생성되지 않습니다.

6. 구성 맵을 삭제합니다.

```
$ oc delete configmap <configmap_name> -n openshift-marketplace
```

7. 웹 콘솔에서 OperatorHub를 사용하여 Operator를 다시 설치합니다.

검증

- Operator가 제대로 다시 설치되었는지 확인합니다.

```
$ oc get sub,csv,installplan -n <namespace>
```

10.5. OPENSHIFT SERVERLESS 사용자 정의 리소스 정의 삭제

OpenShift Serverless를 설치 제거해도 Operator 및 API CRD(사용자 정의 리소스 정의)는 클러스터에 남아 있습니다. 다음 절차를 사용하여 남아 있는 CRD를 제거할 수 있습니다.

중요

Operator 및 API CRD를 제거하면 Knative 서비스를 포함하여 이를 사용하여 정의한 모든 리소스도 제거됩니다.

10.5.1. OpenShift Serverless Operator 및 API CRD 제거

다음 절차를 사용하여 Operator 및 API CRD를 삭제합니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.
- 클러스터 관리자 액세스 권한이 있는 OpenShift Container Platform 계정에 액세스할 수 있습니다.
- Knative Serving을 설치 제거하고 OpenShift Serverless Operator를 제거했습니다.

절차

- 남아 있는 OpenShift Serverless CRD를 삭제하려면 다음 명령을 입력합니다.

```
$ oc get crd -oname | grep 'knative.dev' | xargs oc delete
```

11장. OPENSHIFT SERVERLESS 지원

이 설명서에 설명된 절차를 수행하는 데 어려움이 있는 경우 Red Hat Customer Portal(http://access.redhat.com)을 방문하십시오. Red Hat 고객 포털을 사용하여 Red Hat 제품에 대한 기술 지원 문서의 Red Hat 지식베이스를 검색하거나 살펴볼 수 있습니다. Red Hat GSS(글로벌 지원 서비스)에 지원 케이스를 제출하거나 다른 제품 설명서에 액세스할 수도 있습니다.

이 가이드를 개선하기 위한 제안이 있거나 오류를 발견한 경우 가장 관련 있는 문서 구성 요소에 대한 Jira 문제를 제출할 수 있습니다. 콘텐츠를 쉽게 찾을 수 있도록 섹션 번호, 가이드 이름, OpenShift Serverless 버전과 같은 구체적인 세부 사항을 입력하십시오.

11.1. RED HAT 지식베이스 정보

Red Hat 지식베이스는 Red Hat의 제품과 기술을 최대한 활용할 수 있도록 풍부한 콘텐츠를 제공합니다. Red Hat 지식베이스는 Red Hat 제품 설치, 설정 및 사용에 대한 기사, 제품 문서 및 동영상으로 구성되어 있습니다. 또한 알려진 문제에 대한 솔루션을 검색할 수 있으며, 간결한 근본 원인 설명 및 해결 단계를 제공합니다.

11.2. RED HAT 지식베이스 검색

OpenShift Container Platform 문제가 발생한 경우 초기 검색을 수행하여 솔루션이 이미 Red Hat Knowledgebase 내에 존재하는지 확인할 수 있습니다.

사전 요구 사항

- Red Hat 고객 포털 계정이 있어야 합니다.

절차

1. Red Hat 고객 포털에 로그인합니다.
2. 기본 Red Hat 고객 포털 검색 필드에 다음과 같이 문제와 관련된 키워드 및 문자열을 입력하십시오.
   - OpenShift Container Platform 구성 요소 (etcd 등)

- 관련 절차 (예: installation 등)
- 명시적 실패와 관련된 경고, 오류 메시지 및 기타 출력

3. Search를 클릭합니다
4. OpenShift Container Platform 제품 필터를 선택합니다.
5. Knowledgebase 콘텐츠 유형 필터를 선택합니다.

11.3. 지원 케이스 제출

사전 요구 사항

- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- OpenShift CLI(oc)가 설치되어 있습니다.
- Red Hat 고객 포털 계정이 있어야 합니다.
- Red Hat 표준 또는 프리미엄 서브스크립션이 있습니다.

절차

1. Red Hat 고객 포털에 로그인하고 SUPPORT CASES → Open a case를 선택합니다.
2. 문제에 대한 적절한 카테고리(예: Defect/Bug), 제품(OpenShift Container Platform) 및 제품 버전(예: 자동 입력되어 있지 않은 경우4.11)을 선택합니다.
3. 보고되는 문제와 관련이 있을 수 있는 권장 Red Hat 지식베이스 솔루션 목록을 확인합니다. 제안된 문서로 문제가 해결되지 않으면 Continue을 클릭합니다.

4. *문제의 증상 및 예상 동작에 대한 자세한 정보와 함께 간결하지만 구체적인 문제 요약을 입력합니다.*

5. *보고되는 문제와 관련있는 제안된 **Red Hat** 지식베이스 솔루션 목록을 확인하십시오. 케이스 작성 과정에서 더 많은 정보를 제공하면 목록이 구체화됩니다. 제안된 문서로 문제가 해결되지 않으면 **Continue**을 클릭합니다.*

6. *제시된 계정 정보가 정확한지 확인하고 필요한 경우 적절하게 수정합니다.*

7. *자동 입력된 **OpenShift Container Platform** 클러스터 **ID**가 올바른지 확인합니다. 그렇지 않은 경우 클러스터 **ID**를 수동으로 가져옵니다.*

   - *OpenShift Container Platform 웹 콘솔을 사용하여 클러스터 ID를 수동으로 가져오려면 다음을 수행합니다.*

     a. *Home → Dashboards → Overview로 이동합니다.*

     b. *Details 섹션의 Cluster ID 필드에서 값을 찾습니다.*

   - *또는 OpenShift Container Platform 웹 콘솔을 통해 새 지원 케이스를 열고 클러스터 ID를 자동으로 입력할 수 있습니다.*

     a. *툴바에서 (?) Help → Open Support Case로 이동합니다.*

     b. *Cluster ID 값이 자동으로 입력됩니다.*

   - *OpenShift CLI (oc)를 사용하여 클러스터 ID를 얻으려면 다음 명령을 실행합니다.*

     ```
     $ oc get clusterversion -o jsonpath='{.items[].spec.clusterID}{"\n"}'
     ```

8. *프롬프트가 표시되면 다음 질문을 입력한 후 **Continue**를 클릭합니다.*

- 이 문제가 어디에서 발생했습니까? 어떤 시스템 환경을 사용하고 있습니까?

- 이 동작이 언제 발생했습니까? 발생 빈도는 어떻게 됩니까? 반복적으로 발생합니까? 특정 시간에만 발생합니까?

- 이 문제의 발생 기간 및 비즈니스에 미치는 영향에 대한 정보를 제공해주십시오.

9. 관련 진단 데이터 파일을 업로드하고 **Continue**를 클릭합니다. **oc adm must-gather** 명령을 사용하여 수집된 데이터와 해당 명령으로 수집되지 않은 특정 문제와 관련된 데이터를 제공하는 것이 좋습니다

10. 관련 케이스 관리 세부 정보를 입력하고 **Continue**를 클릭합니다.

11. 케이스 세부 정보를 미리보고 **Submit**을 클릭합니다.

11.4. 지원을 위한 진단 정보 수집

지원 케이스를 열면 클러스터에 대한 디버깅 정보를 **Red Hat** 지원에 제공하면 도움이 됩니다. **must-gather** 툴을 사용하면 **OpenShift Serverless** 관련 데이터를 포함하여 **OpenShift Container Platform** 클러스터에 대한 진단 정보를 수집할 수 있습니다. 즉각 지원을 받을 수 있도록 **OpenShift Container Platform** 및 **OpenShift Serverless** 둘 다에 대한 진단 정보를 제공하십시오.

11.4.1. must-gather 툴 정보

oc adm must-gather CLI 명령은 다음과 같은 문제를 디버깅하는 데 필요할 가능성이 높은 클러스터에서 정보를 수집합니다.

- 리소스 정의

- 서비스 로그

기본적으로 **oc adm must-gather** 명령은 기본 플러그인 이미지를 사용하고 **./must-gather.local** 에 씁니다.

또는 다음 섹션에 설명된 대로 적절한 인수로 명령을 실행하여 특정 정보를 수집할 수 있습니다.

- 하나 이상의 특정 기능과 관련된 데이터를 수집하려면 다음 섹션에 나열된 대로 이미지와 함께 **--image** 인수를 사용합니다.

  예를 들어 다음과 같습니다.

  ```
  $ oc adm must-gather  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.11.0
  ```

- 감사 로그를 수집하려면 다음 섹션에 설명된 대로 **-- /usr/bin/gather_audit_logs** 인수를 사용합니다.

  예를 들어 다음과 같습니다.

  ```
  $ oc adm must-gather -- /usr/bin/gather_audit_logs
  ```

  참고

  감사 로그는 파일 크기를 줄이기 위해 기본 정보 세트의 일부로 수집되지 않습니다.

oc adm must-gather 을 실행하면 클러스터의 새 프로젝트에 임의의 이름이 있는 새 **Pod**가 생성됩니다. 해당 **Pod**에 대한 데이터가 수집되어 **must-gather.local**로 시작하는 새 디렉터리에 저장됩니다. 이 디렉터리는 현재 작업 중인 디렉터리에 생성되어 있습니다.

예를 들어 다음과 같습니다.

```
NAMESPACE                  NAME              READY   STATUS     RESTARTS     AGE
...
openshift-must-gather-5drcj    must-gather-bklx4    2/2     Running    0          72s
openshift-must-gather-5drcj    must-gather-s8sdh    2/2     Running    0          72s
...
```

11.4.2. OpenShift Serverless 데이터 수집 정보

oc adm must-gather CLI 명령을 사용하면 **OpenShift Serverless**와 연관된 기능 및 오브젝트를 포함하여 클러스터에 대한 정보를 수집할 수 있습니다. **must-gather**로 **OpenShift Serverless** 데이터를 수집

하려면 OpenShift Serverless 이미지 및 설치된 OpenShift Serverless 버전의 이미지 태그를 지정해야 합니다.

사전 요구 사항

- *OpenShift CLI(oc)를 설치합니다.*

절차

- *oc adm must-gather 명령을 사용하여 데이터 수집*

```
$ oc adm must-gather --image=registry.redhat.io/openshift-serverless-1/svls-must-gather-rhel8:<image_version_tag>
```

명령 예

```
$ oc adm must-gather --image=registry.redhat.io/openshift-serverless-1/svls-must-gather-rhel8:1.14.0
```