Red Hat OpenShift Serverless 1.33

서빙

Knative Serving 시작하기 및 서비스 구성

Last Updated: 2024-09-18

Red Hat OpenShift Serverless 1.33 서빙

Knative Serving 시작하기 및 서비스 구성

법적 공지

초록

이 문서에서는 Knative Serving을 시작하는 방법에 대한 정보를 제공합니다. 애플리케이션을 구성하고 자동 스케일링, 트래픽 분할, 외부 및 수신 라우팅과 같은 기능을 다룹니다.

차례

1장. KNATIVE SERVING 시작하기

1.1. 서버리스 애플리케이션 생성

서버리스 애플리케이션은 경로 및 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: showcase 1
  namespace: default 2
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/showcase 3
          env:
            - name: GREET 4
              value: Ciao
```

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. OpenShift Container Platform의 경우 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.

1.1.1. Knative CLI를 사용하여 서버리스 애플리케이션 생성

Knative(**kn**) CLI를 사용하여 서버리스 애플리케이션을 생성하면 YAML 파일을 직접 수정하는 것보다 간소화되고 직관적인 사용자 인터페이스가 제공됩니다. **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.

프로세스

- Knative 서비스를 생성합니다.

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

다음과 같습니다.

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

명령 예

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

출력 예

```
Creating service 'showcase' in namespace 'default':

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

Service 'showcase' created with latest revision 'showcase-00001' and URL:
http://showcase-default.apps-crc.testing
```

1.1.2. YAML을 사용하여 서버리스 애플리케이션 생성

YAML 파일을 사용하여 Knative 리소스를 생성하면 선언적 API를 사용하므로 선언적 및 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 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.

프로세스

1. 다음 샘플 코드를 포함하는 YAML 파일을 생성합니다.

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

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

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

OpenShift Container Platform 웹 콘솔의 **개발자** 화면으로 전환하거나 Knative(**kn**) CLI 또는 YAML 파일을 사용하지 않으려면 OpenShift Container Platform 웹 콘솔의 **관리자** 화면을 사용하여 Knative 구성 요소를 생성할 수 있습니다.

1.1.3. 관리자 화면을 사용하여 서버리스 애플리케이션 생성

서버리스 애플리케이션은 경로 및 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: showcase 1
  namespace: default 2
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/showcase 3
          env:
            - name: GREET 4
              value: Ciao
```

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 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.

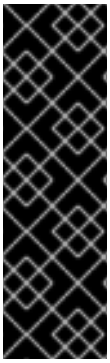
- OpenShift Serverless Operator 및 Knative Serving이 설치되어 있습니다.
- 웹 콘솔에 로그인한 후 **관리자** 화면에 있습니다.

프로세스

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.

1.1.4. 오프라인 모드를 사용하여 서비스 생성

클러스터에서 변경 사항이 발생하지 않도록 **kn service** 명령을 오프라인 모드에서 실행할 수 있으며 대신 로컬 시스템에 서비스 설명자 파일이 생성됩니다. 설명자 파일이 생성되면 파일을 수정한 후 클러스터의 변경 사항을 전파할 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.

프로세스

1. 오프라인 모드에서 로컬 Knative 서비스 설명자 파일을 생성합니다.

```
$ kn service create showcase \
    --image quay.io/openshift-knative/showcase \
    --target ./ \
    --namespace test
```

출력 예

```
Service 'showcase' 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
        └── showcase.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 설명자 파일 **showcase.yaml**이 포함되어 있습니다.

3. 생성된 서비스 기술자 파일을 확인합니다.

```
$ cat test/ksvc/showcase.yaml
```

출력 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  creationTimestamp: null
  name: showcase
  namespace: test
spec:
  template:
    metadata:
      annotations:
```

```
        client.knative.dev/user-image: quay.io/openshift-knative/showcase
      creationTimestamp: null
    spec:
      containers:
      - image: quay.io/openshift-knative/showcase
        name: ""
        resources: {}
status: {}
```

4. 새 서비스에 대한 정보를 나열합니다.

```
$ kn service describe showcase --target ./ --namespace test
```

출력 예

```
Name:       showcase
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/showcase.yaml
```

출력 예

```
Creating service 'showcase' in namespace 'test':

  0.058s The Route is still working to reflect the latest desired specification.
  0.098s ...
  0.168s Configuration "showcase" 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 'showcase' created to latest revision 'showcase-00001' is available at URL:
http://showcase-test.apps.example.com
```

1.1.5. 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 사용하지만 **oc get ksvc** 의 출력은 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.

프로세스

1. 애플리케이션 URL을 찾습니다.

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

출력 예

```
NAME URL LATESTCREATED LATESTREADY READY REASON
showcase http://showcase-default.example.com showcase-00001 showcase-00001 True
```

2. 클러스터에 요청한 후 출력을 확인합니다.

HTTP 요청의 예(HTTPie 툴 사용)

```
$ http showcase-default.example.com
```

HTTPS 요청의 예

```
$ https showcase-default.example.com
```

출력 예

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Quarkus/2.13.7.Final-redhat-00003 Java/17.0.7
X-Config: {"sink":"http://localhost:31111","greet":"Ciao","delay":0}
X-Version: v0.7.0-4-g23d460f
content-length: 49

{
```

```
    "artifact": "knative-showcase",
    "greeting": "Ciao"
}
```

3. 선택 사항: 시스템에 HTTPie 툴이 설치되어 있지 않은 경우 대신 curl 툴을 사용할 수 있습니다.

HTTPS 요청의 예

```
$ curl http://showcase-default.example.com
```

출력 예

```
{"artifact":"knative-showcase","greeting":"Ciao"}
```

4. 선택 사항: 인증서 체인에서 자체 서명된 인증서와 관련된 오류가 발생하면 HTTPie 명령에 **--verify=no** 플래그를 추가하여 오류를 무시할 수 있습니다.

```
$ https --verify=no showcase-default.example.com
```

출력 예

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Quarkus/2.13.7.Final-redhat-00003 Java/17.0.7
X-Config: {"sink":"http://localhost:31111","greet":"Ciao","delay":0}
X-Version: v0.7.0-4-g23d460f
content-length: 49

{
    "artifact": "knative-showcase",
    "greeting": "Ciao"
}
```

중요

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

5. 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://showcase-default.example.com --cacert <file>
```

출력 예

```
{"artifact":"knative-showcase","greeting":"Ciao"}
```

1.1.6. 추가 리소스

- Knative Serving CLI 명령

- Knative 서비스의 JSON Web Token 인증 설정

2장. 자동 확장

2.1. 자동 확장

Knative Serving은 들어오는 수요와 일치하도록 애플리케이션의 *자동 스케일링 또는 자동 스케일링을* 제공합니다. 예를 들어 애플리케이션에 트래픽이 수신되지 않고 scale-to-zero가 활성화된 경우 Knative Serving은 애플리케이션을 복제본 0으로 축소합니다. 스케일 투 0이 비활성화된 경우 애플리케이션은 클러스터의 애플리케이션에 대해 구성된 최소 복제본 수로 축소됩니다. 애플리케이션에 대한 트래픽이 증가하는 경우 필요에 따라 복제본을 확장할 수도 있습니다.

Knative 서비스의 자동 스케일링 설정은 클러스터 관리자(또는 AWS 및 OpenShift Dedicated의 Red Hat OpenShift Service 전용 관리자) 또는 개별 서비스에 대해 구성된 모니터링별 설정일 수 있습니다.

OpenShift Container Platform 웹 콘솔을 사용하거나 서비스의 YAML 파일을 수정하거나 Knative(**kn**) CLI를 사용하여 서비스에 대한 감독별 설정을 수정할 수 있습니다.

참고

서비스에 설정한 제한 또는 대상은 애플리케이션의 단일 인스턴스에 대해 측정됩니다. 예를 들어 **target** 주석을 **50** 으로 설정하면 각 버전에서 50개의 요청을 처리할 수 있도록 애플리케이션을 스케일링하도록 자동 스케일러를 구성합니다.

2.2. 스케일링 경계

스케일 경계는 언제든지 애플리케이션을 제공할 수 있는 최소 및 최대 복제본 수를 결정합니다. 애플리케이션의 스케일 범위를 설정하여 콜드 시작 또는 컴퓨팅 비용을 제어할 수 있습니다.

2.2.1. 최소 스케일 경계

애플리케이션을 제공할 수 있는 최소 복제본 수는 **min-scale** 주석에 따라 결정됩니다. scale to zero가 활성화되지 않은 경우 **min-scale** 값은 기본적으로 **1** 입니다.

다음 조건이 충족되면 **min-scale** 값은 기본적으로 **0** 개의 복제본으로 설정됩니다.

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

min-scale 주석이 있는 서비스 사양의 예

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

2.2.1.1. Knative CLI를 사용하여 min-scale 주석 설정

Knative(**kn**) CLI를 사용하여 **min-scale** 주석을 설정하면 YAML 파일을 직접 수정하는 것보다 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.

프로세스

- **--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 showcase --image quay.io/openshift-knative/showcase --scale-min 2
```

2.2.2. 최대 스케일 경계

애플리케이션을 제공할 수 있는 최대 복제본 수는 **max-scale** 주석에 따라 결정됩니다. **max-scale** 주석이 설정되지 않은 경우 생성된 복제본 수에 대한 상한이 없습니다.

max-scale 주석이 있는 서비스 사양의 예

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

2.2.2.1. Knative CLI를 사용하여 max-scale 주석 설정

Knative(**kn**) CLI를 사용하여 **max-scale** 주석을 설정하면 YAML 파일을 직접 수정하는 것보다 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.

프로세스

- **--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 showcase --image quay.io/openshift-knative/showcase --scale-max 10
```

2.3. 동시성

동시성은 언제든지 애플리케이션의 각 복제본에서 처리할 수 있는 동시 요청 수를 결정합니다. 동시성은 *소프트 제한 또는 하드 제한으로* 구성할 수 있습니다.

- 소프트 제한은 엄격하게 적용된 바인딩이 아닌 대상 요청 제한입니다. 예를 들어, 트래픽이 갑자기 버스트되는 경우 소프트 제한 대상을 초과할 수 있습니다.
- 하드 제한은 엄격하게 적용되는 상한 요청 제한입니다. 동시성이 하드 제한에 도달하면 서+ 요청이 버퍼링되며 요청을 실행할 수 있는 사용 가능한 용량이 충분히 있을 때까지 기다려야 합니다.

중요

하드 제한 구성을 사용하는 것은 애플리케이션에 명확한 사용 사례가 있는 경우에만 권장됩니다. 지정된 하드 제한이 낮으면 애플리케이션의 처리량 및 대기 시간에 부정적인 영향을 미칠 수 있으며 콜드 시작이 발생할 수 있습니다.

소프트 대상과 하드 제한을 추가하면 자동 스케일러가 동시 요청의 소프트 대상 수를 대상으로 하지만 최대 요청 수에 대한 하드 제한 값을 강제 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.

2.3.1. 소프트 동시성 대상 구성

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

프로세스

- 선택 사항: **Service** 사용자 정의 리소스의 사양에서 Knative 서비스에 대한 **autoscaling.knative.dev/target** 주석을 설정합니다.

서비스 사양의 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: showcase
  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 50개의 요청으로 서비스를 생성하는 명령 예

```
$ kn service create showcase --image quay.io/openshift-knative/showcase --concurrency-target 50
```

2.3.2. 하드 동시성 제한 구성

하드 동시성 제한은 엄격하게 적용되는 상한 요청 제한입니다. 동시성이 하드 제한에 도달하면 서+ 요청이 버퍼링되며 요청을 실행할 수 있는 사용 가능한 용량이 충분히 있을 때까지 기다려야 합니다. **containerConcurrency** 사양을 수정하거나 올바른 플래그와 함께 **kn service** 명령을 사용하여 Knative 서비스에 대한 하드 동시성 제한을 지정할 수 있습니다.

프로세스

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

서비스 사양의 예

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

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

0 보다 큰 값은 한 번에 하나의 서비스 복제본으로 전달될 수 있는 정확한 요청 수를 지정합니다. 이 예제에서는 하드 동시성 제한을 50개의 요청을 활성화합니다.

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

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

동시성 제한이 50개의 요청으로 서비스를 생성하는 명령 예

```
$ kn service create showcase --image quay.io/openshift-knative/showcase --concurrency-limit 50
```

2.3.3. 동시성 대상 사용률

이 값은 자동 스케일러가 실제로 대상으로 하는 동시성 제한의 백분율을 지정합니다. 또한 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 설정된 경우, 자동 스케일러는 모든 기존 복제본의 평균 동시 요청 수가 7에 도달하면 새 복제본을 생성합니다. 7에서 10까지 번호가 매겨진 요청은 기존 복제본으로 계속 전송되지만 **containerConcurrency** 값에 도달한 후에도 추가 복제본이 필요할 것으로 예상됩니다.

target-utilization-percentage 주석을 사용하여 구성된 서비스의 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: showcase
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/target-utilization-percentage: "70"
...
```

2.4. SCALE-TO-ZERO

Knative Serving은 들어오는 수요와 일치하도록 애플리케이션의 *자동 스케일링 또는 자동 스케일링을* 제공합니다.

2.4.1. scale-to-zero 활성화

클러스터의 애플리케이션에 대해 **enable-scale-to-zero** 사양을 사용하여 전역적으로 스케일 투 0을 활성화하거나 비활성화할 수 있습니다.

사전 요구 사항

- 클러스터에 OpenShift Serverless Operator 및 Knative Serving이 설치되어 있습니다.
- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- 기본 Knative Pod Autoscaler를 사용하고 있습니다. Kubernetes Horizontal Pod Autoscaler를 사용하는 경우 0으로 스케일링 기능을 사용할 수 없습니다.

프로세스

- **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-zero가 활성화됩니다. false로 설정하면 애플리케이션이 구성된 *최소 스케일 바운드* 로 축소됩니다. 기본값은 **"true"** 입니다.

2.4.2. scale-to-zero 유예 기간 구성

Knative Serving에서는 애플리케이션의 Pod가 0개로 자동 스케일링을 제공합니다. **scale-to-zero-grace-period** 사양을 사용하여 애플리케이션의 마지막 복제본이 제거되기 전에 Knative에서 0으로 스케일링할 때까지 대기하는 상한 시간 제한을 정의할 수 있습니다.

사전 요구 사항

- 클러스터에 OpenShift Serverless Operator 및 Knative Serving이 설치되어 있습니다.
- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- 기본 Knative Pod Autoscaler를 사용하고 있습니다. Kubernetes Horizontal Pod Autoscaler를 사용하는 경우 scale-to-zero 기능을 사용할 수 없습니다.

프로세스

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

3장. OPENSHIFT SERVERLESS 애플리케이션 구성

3.1. SERVING에 대한 멀티컨테이너 지원

단일 Knative 서비스를 사용하여 멀티컨테이너 Pod를 배포할 수 있습니다. 이 방법은 애플리케이션 책임을 작고 특수화된 부분으로 분리하는 데 유용합니다.

3.1.1. 멀티컨테이너 서비스 구성

멀티컨테이너 지원은 기본적으로 활성화되어 있습니다. 서비스에서 여러 컨테이너를 지정하여 멀티컨테이너 Pod를 생성할 수 있습니다.

프로세스

1. 추가 컨테이너를 포함하도록 서비스를 수정합니다. 하나의 컨테이너만 요청을 처리할 수 있으므로 하나의 **컨테이너에 대한 포트**를 지정합니다. 다음은 두 개의 컨테이너가 있는 구성의 예입니다.

여러 컨테이너 구성

```
apiVersion: serving.knative.dev/v1
kind: Service
...
spec:
  template:
    spec:
      containers:
        - name: first-container 1
          image: gcr.io/knative-samples/helloworld-go
          ports:
            - containerPort: 8080 2
        - name: second-container 3
          image: gcr.io/knative-samples/helloworld-java
```

1 첫 번째 컨테이너 구성

2 첫 번째 컨테이너의 포트 사양입니다.

3 두 번째 컨테이너 구성.

3.2. EMPTYDIR 볼륨

emptyDir 볼륨은 Pod가 생성될 때 생성되는 빈 볼륨이며 임시 작업 디스크 공간을 제공하는 데 사용됩니다. **emptyDir** 볼륨은 생성된 Pod가 삭제될 때 삭제됩니다.

3.2.1. EmptyDir 확장 구성

kubernetes.podspec-volumes-emptydir 확장은 **emptyDir** 볼륨을 Knative Serving에서 사용할 수 있는지 여부를 제어합니다. **emptyDir** 볼륨을 사용하려면 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
...
```

3.3. SERVING에 대한 영구 볼륨 클레임

일부 서버리스 애플리케이션에는 영구 데이터 스토리지가 필요합니다. 다양한 볼륨 유형을 구성하면 Knative 서비스에 데이터 스토리지를 제공할 수 있습니다. 서비스 제공에서는 **보안**,**configMap**,**projected**, **emptyDir** 과 같은 볼륨 유형 마운트를 지원합니다.

Knative 서비스에 대한 PVC(영구 볼륨 클레임)를 구성할 수 있습니다. 영구 볼륨 유형은 플러그인으로 구현됩니다. 사용 가능한 영구 볼륨 유형이 있는지 확인하려면 클러스터에 사용 가능한 스토리지 클래스 또는 설치된 스토리지 클래스를 확인할 수 있습니다. 영구 볼륨이 지원되지만 기능 플래그를 활성화해야 합니다.

주의

대용량 볼륨을 마운트하면 애플리케이션 시작 시간이 상당히 지연될 수 있습니다.

3.3.1. PVC 지원 활성화

프로세스

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

   쓰기 액세스 권한이 있는 PVC 활성화

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

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

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

참고

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

ocs-storagecluster-cephfs 스토리지 클래스가 지원되며 Red Hat OpenShift Data Foundation 에서 제공됩니다.

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 클레임하려면 다음과 같이 서비스를 수정합니다.

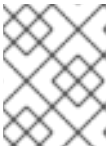
Knative 서비스 PVC 구성

```
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

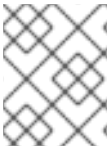
Knative 서비스에서 영구 스토리지를 성공적으로 사용하려면 Knative 컨테이너 사용자의 사용자 권한과 같은 추가 구성이 필요합니다.

3.3.2. OpenShift Container Platform에 대한 추가 리소스

- 영구저장장치 이해

3.4. INIT 컨테이너

Init 컨테이너 는 Pod의 애플리케이션 컨테이너보다 먼저 실행되는 특수 컨테이너입니다. 일반적으로 애플리케이션에 대한 초기화 논리를 구현하는 데 사용되며, 여기에는 설정 스크립트 실행 또는 필수 구성 다운로드가 포함될 수 있습니다. **KnativeServing** 사용자 정의 리소스(CR)를 수정하여 Knative 서비스에 init 컨테이너 사용을 활성화할 수 있습니다.



참고

Init Container는 애플리케이션 시작 시간이 길어질 수 있으며 서버리스 애플리케이션에 주의하여 사용해야 합니다. 이 기능은 자주 확장 및 축소할 것으로 예상됩니다.

3.4.1. init 컨테이너 활성화

사전 요구 사항

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

프로세스

- **kubernetes.podspec-init-containers** 플래그를 **KnativeServing** CR에 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
...
```

3.5. 다이제스트로 이미지 태그 확인

Knative Serving 컨트롤러가 컨테이너 레지스트리에 액세스할 수 있는 경우 서비스 버전을 생성할 때 Knative Serving에서 이미지 태그를 다이제스트로 확인합니다. 이를 *tag-to-digest resolution* 이라고 하며 배포에 대한 일관성을 제공하는 데 도움이 됩니다.

3.5.1. tag-to-digest 해결

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** CR과 동일한 네임스페이스에 있어야 합니다.

KnativeServing CR에 보안이 포함되어 있지 않은 경우 이 설정은 기본적으로 PKI(공개 키 인프라)를 사용합니다. PKI를 사용하면 클러스터 전체 인증서가 **config-service-sa** 구성 맵을 사용하여 Knative Serving 컨트롤러에 자동으로 삽입됩니다. OpenShift Serverless Operator는 **config-service-sa** 구성 맵을 클러스터 전체 인증서로 채우고 구성 맵을 컨트롤러의 볼륨으로 마운트합니다.

3.5.1.1. 시크릿을 사용하여 tag-to-digest 확인 구성

controller-custom-certs 사양에서 **Secret** 유형을 사용하는 경우 보안이 보안 볼륨으로 마운트됩니다. Knative 구성 요소는 보안에 필요한 인증서가 있다고 가정하면 시크릿을 직접 사용합니다.

사전 요구 사항

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

프로세스

1. 보안을 생성합니다.

   명령 예

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

2. **KnativeServing** CR(사용자 정의 리소스)에서 **Secret** 유형을 사용하도록 **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
   ```

3.6. TLS 인증 구성

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

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

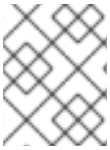
참고

Red Hat OpenShift Service Mesh 통합을 사용하여 내부 TLS를 활성화하려면 다음 절차에 설명된 내부 암호화 대신 mTLS와 Service Mesh를 활성화해야 합니다.

AWS의 OpenShift Container Platform 및 Red Hat OpenShift Service는 mTLS와 함께 Service Mesh를 사용할 때 Knative Serving 메트릭 활성화 문서를 참조하십시오.

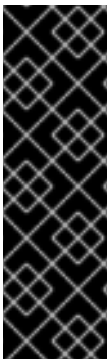
3.6.1. 내부 트래픽에 대한 TLS 인증 활성화

OpenShift Serverless에서는 기본적으로 TLS 에지 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 이 트래픽의 보안이 향상됩니다.



참고

Red Hat OpenShift Service Mesh 통합을 사용하여 내부 TLS를 활성화하려면 다음 절차에 설명된 내부 암호화 대신 mTLS와 Service Mesh를 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**) CLI가 설치되어 있습니다.

프로세스

1. **KnativeServing** 리소스를 생성하거나 업데이트하고 사양에 **internal-encryption: "true"** 필드가 포함되어 있는지 확인합니다.

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

2. **knative-serving** 네임스페이스에서 activator Pod를 다시 시작하여 인증서를 로드합니다.

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

추가 리소스

- Apache Kafka의 Knative 브로커에 대한 TLS 인증 구성
- Apache Kafka 채널의 TLS 인증 구성
- mTLS와 함께 서비스 메시를 사용할 때 Knative Serving 메트릭 활성화

3.7. KOURIER 구성

Kourier는 Knative Serving의 경량 Kubernetes 네이티브 Ingress입니다. Kourier는 Knative의 게이트웨이 역할을 하며 HTTP 트래픽을 Knative 서비스로 라우팅합니다.

3.7.1. Kourier getaways의 kourier-bootstrap 사용자 정의

Kourier의 Envoy 프록시 구성 요소는 Knative 서비스에 대한 인바운드 및 아웃바운드 HTTP 트래픽을 처리합니다. 기본적으로 Kourier에는 **knative-serving-ingress** 네임스페이스의 **kourier-bootstrap** 구성 맵에 Envoy 부트스트랩 구성이 포함되어 있습니다. 이 구성을 변경할 수 있습니다.

사전 요구 사항

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

프로세스

- **KnativeServing** CR(사용자 정의 리소스)에서 **spec.ingress.kourier.bootstrap-configmap** 필드를 변경하여 사용자 정의 부트스트랩 구성 맵을 지정합니다.

  KnativeServing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  config:
    network:
      ingress-class: kourier.ingress.networking.knative.dev
  ingress:
    kourier:
      bootstrap-configmap: my-configmap
      enabled: true
# ...
```

3.8. 제한적인 네트워크 정책

3.8.1. 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: []
```

3.8.2. 제한적인 네트워크 정책을 사용하여 클러스터에서 Knative 애플리케이션과의 통신 활성화

Knative 시스템 Pod에서 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 비Knative 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* 대신 사용할 수 있습니다. Knative 서비스에 대한 JSON 웹 토큰 인증에는 Service Mesh가 필요합니다.

사전 요구 사항

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

프로세스

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.

4장. 서버리스 애플리케이션 디버깅

다양한 방법을 사용하여 Serverless 애플리케이션의 문제를 해결할 수 있습니다.

4.1. 터미널 출력 확인

배포 명령 출력을 확인하여 배포 성공 여부를 확인할 수 있습니다. 배포 프로세스가 종료되면 배포에 실패한 이유를 설명하는 오류 메시지가 출력에 표시되어야 합니다. 이러한 유형의 오류는 매니페스트가 잘못 구성되었거나 잘못된 명령으로 인해 가장 많이 발생합니다.

프로세스

- 애플리케이션을 배포하고 관리하는 클라이언트에서 명령 출력을 엽니다. 다음 예제는 실패한 **oc apply** 명령 뒤에 표시될 수 있는 오류입니다.

```
Error from server (InternalError): error when applying patch:
{"metadata":{"annotations":{"kubectl.kubernetes.io/last-applied-configuration":"
{\"apiVersion\":\"serving.knative.dev/v1\",\"kind\":\"Route\",\"metadata\":{\"annotations\":
{},\"name\":\"route-example\",\"namespace\":\"default\"},\"spec\":{\"traffic\":
[{\"configurationName\":\"configuration-example\",\"percent\":50}]}}\n"}},"spec":{"traffic":
[{"configurationName":"configuration-example","percent":50}]}}
to:
&{0xc421d98240 0xc421e77490 default route-example STDIN 0xc421db0488 264682 false}
for: "STDIN": Internal error occurred: admission webhook "webhook.knative.dev" denied the
request: mutation failed: The route must have traffic percent sum equal to 100.
ERROR: Non-zero return code '1' from command: Process exited with status 1
```

  이 출력은 경로 트래픽 백분율을 100으로 구성해야 함을 나타냅니다.

4.2. POD 상태 확인

Serverless 애플리케이션의 문제를 식별하려면 **Pod** 오브젝트의 상태를 확인해야 할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 배포의 모든 Pod를 나열합니다.

```
$ oc get pods
```

   출력 예

```
NAME                                                 READY     STATUS             RESTARTS   AGE
configuration-example-00001-deployment-659747ff99-9bvr4   2/2       Running            0
3h
configuration-example-00002-deployment-5f475b7849-gxcht   1/2       CrashLoopBackOff   2
36s
```

   출력에서 상태에 대한 선택한 데이터가 있는 모든 Pod를 볼 수 있습니다.

2. 다음 명령을 실행하여 Pod 상태에 대한 자세한 정보를 확인합니다.

   출력 예

```
$ oc get pod <pod_name> --output yaml
```

출력에서 **conditions** 및 **containerStatuses** 필드는 디버깅에 특히 유용할 수 있습니다.

4.3. 버전 상태 확인

Serverless 애플리케이션의 문제를 식별하려면 버전 상태를 확인해야 할 수 있습니다.

프로세스

1. **Configuration** 오브젝트로 경로를 구성하는 경우 다음 명령을 실행하여 배포에 생성된 **Revision** 오브젝트의 이름을 가져옵니다.

```
$ oc get configuration <configuration_name> --output jsonpath="
{.status.latestCreatedRevisionName}"
```

OpenShift **Route** 리소스를 정의하여 라우팅 설정을 지정하는 **Route.yaml** 파일에서 구성 이름을 찾을 수 있습니다.

리버전으로 경로를 직접 구성하는 경우 **Route.yaml** 파일에서 버전 이름을 조회합니다.

2. 다음 명령을 실행하여 버전 상태를 쿼리합니다.

```
$ oc get revision <revision-name> --output yaml
```

준비된 버전에는 **ServiceReady**,**status: "True"**, **type: Ready** 조건이 있어야 합니다. 이러한 조건이 있는 경우 Pod 상태 또는 Istio 라우팅을 확인할 수 있습니다. 그렇지 않으면 리소스 상태에 오류 메시지가 포함됩니다.

4.3.1. 추가 리소스

- 경로 구성

4.4. INGRESS 상태 확인

Serverless 애플리케이션의 문제를 식별하려면 Ingress 상태를 확인해야 할 수 있습니다.

프로세스

- 다음 명령을 실행하여 Ingress의 IP 주소를 확인합니다.

```
$ oc get svc -n istio-system istio-ingressgateway
```

istio-ingressgateway 서비스는 Knative에서 사용하는 **LoadBalancer** 서비스입니다.

외부 IP 주소가 없는 경우 다음 명령을 실행합니다.

```
$ oc describe svc istio-ingressgateway -n istio-system
```

이 명령은 IP 주소가 프로비저닝되지 않은 이유를 출력합니다. 대부분의 경우 할당량 문제로 인해 발생할 수 있습니다.

4.5. 경로 상태 확인

경우에 따라 **Route** 오브젝트에 문제가 있습니다. OpenShift CLI(**oc**)를 사용하여 해당 상태를 확인할 수 있습니다.

프로세스

- 다음 명령을 실행하여 애플리케이션을 배포한 **Route** 오브젝트의 상태를 확인합니다.

```
$ oc get route <route_name> --output yaml
```

& **lt;route_name&** gt;을 **Route** 오브젝트의 이름으로 바꿉니다.

status 오브젝트의 **conditions** 오브젝트는 실패의 경우 이유를 표시합니다.

4.6. INGRESS 및 ISTIO 라우팅 확인

Istio를 Ingress 계층으로 사용하면 Ingress 및 Istio 라우팅에 문제가 있습니다. OpenShift CLI(**oc**)를 사용하여 세부 정보를 확인할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 모든 Ingress 리소스 및 해당 레이블을 나열합니다.

```
$ oc get ingresses.networking.internal.knative.dev -o=custom-
columns='NAME:.metadata.name,LABELS:.metadata.labels'
```

출력 예

```
NAME           LABELS
helloworld-go   map[serving.knative.dev/route:helloworld-go
serving.knative.dev/routeNamespace:default serving.knative.dev/service:helloworld-go]
```

이 출력에서 **serving.knative.dev/route** 및 **serving.knative.dev/routeNamespace** 는 Ingress 리소스가 있는 경로를 나타냅니다. **경로 및 Ingress가 나열되어야 합니다.**

Ingress가 없는 경우 경로 컨트롤러는 경로 또는 Service 오브젝트에서 대상으로 하는 Revision 오브젝트가 준비되지 않은 것으로 가정합니다. 다른 디버깅 절차를 진행하여 버전 준비 상태를 진단합니다.

2. **Ingress가 나열된 경우 다음 명령을 실행하여 경로에 대해 생성된 ClusterIngress 오브젝트를 검사합니다.**

```
$ oc get ingresses.networking.internal.knative.dev <ingress_name> --output yaml
```

출력의 status 섹션에서 type=Ready 조건이 True 인 경우 Ingress가 올바르게 작동합니다. 그렇지 않으면 출력에 오류 메시지가 포함됩니다.

3. Ingress의 상태가 Ready 이면 해당 VirtualService 오브젝트가 있습니다. 다음 명령을 실행하여 VirtualService 오브젝트의 구성을 확인합니다.

```
$ oc get virtualservice -l networking.internal.knative.dev/ingress=<ingress_name> -n
<ingress_namespace> --output yaml
```

VirtualService 오브젝트의 네트워크 구성이 Ingress 및 Route 오브젝트의 네트워크 구성과 일치해야 합니다. VirtualService 오브젝트가 Status 필드를 노출하지 않기 때문에 설정이 전파될 때까지 기다려야 할 수 있습니다.

4.6.1. 추가 리소스

- maistra Service Mesh 문서

5장. KOURIER 및 ISTIO 인그레스

OpenShift Serverless에서는 다음 두 가지 수신 솔루션을 지원합니다.

- Kourier
- Red Hat OpenShift Service Mesh를 사용하는 Istio

기본값은 Kourier입니다.

5.1. KOURIER 및 ISTIO 인그레스 솔루션

5.1.1. Kourier

Kourier는 OpenShift Serverless의 기본 수신 솔루션입니다. 다음과 같은 속성이 있습니다.

- envoy 프록시를 기반으로 합니다.
- 단순하고 가벼움이 있습니다.
- Serverless에서 기능 집합을 제공하는 데 필요한 기본 라우팅 기능을 제공합니다.
- 기본 관찰 기능 및 메트릭을 지원합니다.
- Knative 서비스 라우팅의 기본 TLS 종료를 지원합니다.
- 제한된 구성 및 확장 옵션만 제공합니다.

5.1.2. OpenShift Service Mesh를 사용하는 Istio

Istio를 OpenShift Serverless의 수신 솔루션으로 사용하면 Red Hat OpenShift Service Mesh가 제

공하는 기능을 기반으로 하는 추가 기능 세트가 활성화됩니다.

- 모든 연결 간의 기본 **mTLS**
- 서버리스 구성 요소는 서비스 메시의 일부입니다.
- 추가 관찰 기능 및 메트릭
- 인증 및 인증 지원
- **Red Hat OpenShift Service Mesh**에서 지원하는 사용자 정의 규칙 및 구성

그러나 추가 기능은 더 높은 오버헤드와 리소스 소비를 제공합니다. 자세한 내용은 **Red Hat OpenShift Service Mesh** 설명서를 참조하십시오.

Istio 요구 사항 및 설치 지침은 **Serverless** 문서의 **"OpenShift Serverless**를 사용하여 서비스 메시 통합**"** 섹션을 참조하십시오.

5.1.3. 트래픽 구성 및 라우팅

Kourier 또는 **Istio** 사용 여부에 관계없이 **Knative** 서비스의 트래픽은 **net-kourier-controller** 또는 **net-istio-controller** 에 의해 **knative-serving** 네임스페이스에 구성됩니다.

컨트롤러는 **KnativeService** 및 해당 하위 사용자 정의 리소스를 읽고 수신 솔루션을 구성합니다. 두 수신 솔루션 모두 트래픽 경로의 일부가 되는 수신 게이트웨이 **Pod**를 제공합니다. 두 수신 솔루션은 모두 **Envoy**를 기반으로 합니다. 기본적으로 **Serverless**에는 각 **KnativeService** 오브젝트에 대해 두 개의 경로가 있습니다.

- **OpenShift** 라우터에서 전달하는 클러스터 외부 경로 (예**: myapp-namespace.example.com**)
- 클러스터 도메인을 포함하는 클러스터 로컬 경로 (예**: myapp.namespace.svc.cluster.local**) 이 도메인은 **Knative** 또는 기타 사용자 워크로드에서 **Knative** 서비스를 호출하는 데 사용할 수

있습니다.

수신 게이트웨이는 **serve** 모드 또는 프록시 모드에서 요청을 전달할 수 있습니다.

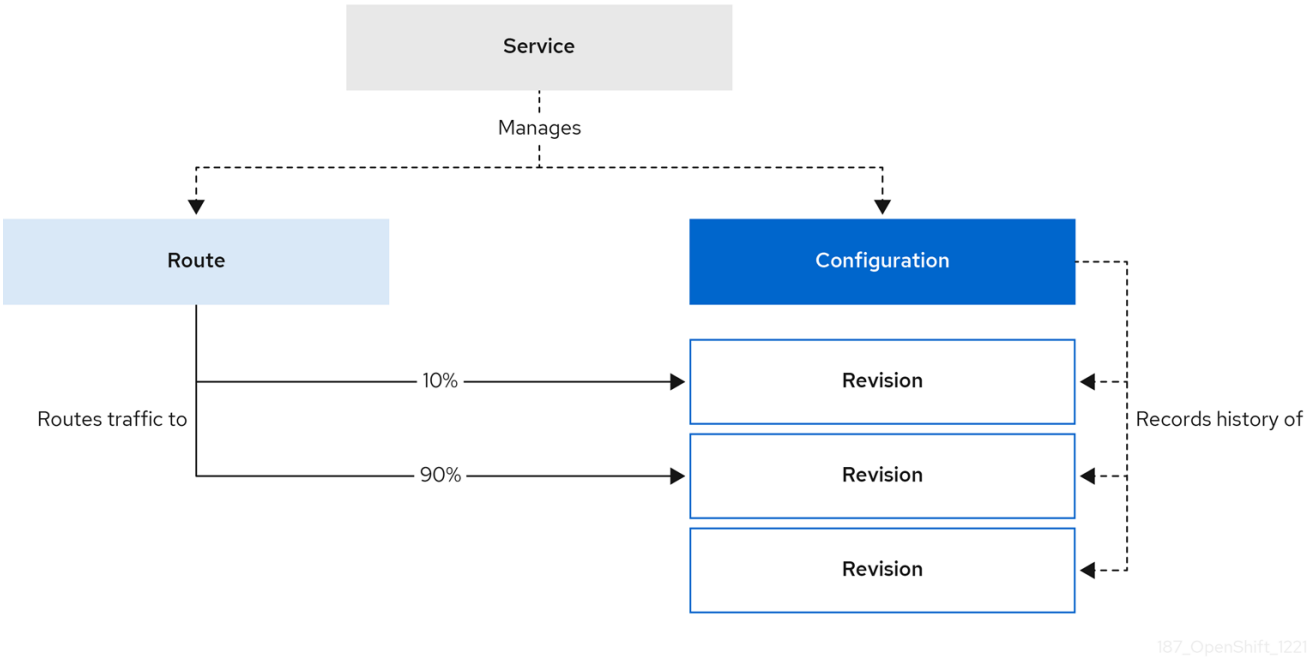
- **serve** 모드에서 요청은 **Knative** 서비스의 **Queue-Proxy** 사이드카 컨테이너로 직접 이동합니다.
- 프록시 모드에서 요청은 먼저 **knative-serving** 네임스페이스의 **Activator** 구성 요소를 통과합니다.

모드 선택은 **Knative, Knative** 서비스 및 현재 트래픽의 구성에 따라 달라집니다. 예를 들어 **Knative** 서비스가 **0**으로 스케일링되면 새 **Knative** 서비스 **Pod**가 시작될 때까지 버퍼 역할을 하는 **Activator** 구성 요소로 요청이 전송됩니다.

6장. 트래픽 분할

6.1. 트래픽 분할 개요

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



경로를 구성하면 요청을 서비스의 다른 버전으로 보낼 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 **Knative** 검증으로 보장되는 최대 **100%**를 추가해야 합니다.

트래픽 사양에 지정된 버전은 수정된 버전 또는 "최신" 버전을 가리킬 수 있으며 서비스의 모든 버전 목록 헤드를 추적합니다. "최신" 버전은 새 버전이 생성되는 경우 업데이트하는 부동 참조 유형입니다. 각 버전에는 해당 버전에 대한 추가 액세스 **URL**을 생성하는 태그가 연결되어 있을 수 있습니다.

트래픽 사양은 다음을 통해 수정할 수 있습니다.

- **Service** 오브젝트의 **YAML**을 직접 편집합니다.

- Knative(kn) CLI --traffic 플래그 사용.
- OpenShift Container Platform 웹 콘솔 사용.

Knative 서비스를 생성할 때 기본 traffic 사양 설정이 없습니다.

6.2. TRAFFIC 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 예에서는 트래픽의 **50%**를 **current** 태그가 지정된 버전에 전송하고 트래픽의 **50%**를 **candidate**로 태그된 버전에 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
```

6.3. KNATIVE CLI를 사용한 트래픽 분할

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

6.3.1. Knative CLI를 사용하여 트래픽 분할 생성

사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Serving**이 클러스터에 설치되어 있습니다.
- **Knative(kn) CLI**가 설치되어 있습니다.
- **Knative** 서비스를 생성했습니다.

프로세스

- 표준 **kn service update** 명령과 함께 **--traffic** 태그를 사용하여 서비스 버전 및 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 버전에 보낼 트래픽의 백분율입니다.

- 선택 사항: --traffic 플래그는 하나의 명령에서 여러 번 지정할 수 있습니다. 예를 들어 @latest 로 태그가 지정된 버전과 stable 이라는 버전이 있는 경우 다음과 같이 각 버전으로 분할하려는 트래픽의 백분율을 지정할 수 있습니다.

명령 예

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

버전이 여러 개 있고 마지막 버전으로 분할해야 하는 트래픽의 백분율을 지정하지 않으면 --traffic 플래그에서 자동으로 계산할 수 있습니다. 예를 들어 example 이라는 세 번째 버전이 있고 다음 명령을 사용합니다.

명령 예

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

나머지 30%의 트래픽은 지정되지 않은 경우에도 예제 버전으로 나뉩니다.

6.4. 트래픽 분할을 위한 CLI 플래그

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

6.4.1. 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
--tag	**@latest=Tag**	최근 준비된 버전에 **Tag** 지정	아니요
--untag	**Tag**	버전에서 **Tag** 제거	제공됨

6.4.1.1. 여러 플래그 및 우선 순위

모든 트래픽 관련 플래그는 단일 kn service update 명령을 사용하여 지정할 수 있습니다. kn은 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.

6.4.1.2. 개정버전 사용자 정의 URL

kn service update 명령을 사용하여 서비스에 --tag 플래그를 할당하면 서비스를 업데이트할 때 생성되는 버전에 대한 사용자 정의 URL이 생성됩니다. 사용자 정의 URL은 https://<tag>-<service_name>-<namespace>.<domain > 또는 http://<tag>-<service_name>-<namespace>.<domain > 패턴을 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.

6.4.1.2.1. 예: 태그를 버전에 할당

다음 예제에서는 latest 태그를 example-revision이라는 버전에 할당합니다.

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

6.4.1.2.2. 예: 버전에서 태그 제거

--untag 플래그를 사용하여 사용자 정의 URL을 제거하도록 태그를 제거할 수 있습니다.

참고

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

다음 명령은 example-revision이라는 버전에서 모든 태그를 제거합니다.

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

6.5. 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.

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

사전 요구 사항

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

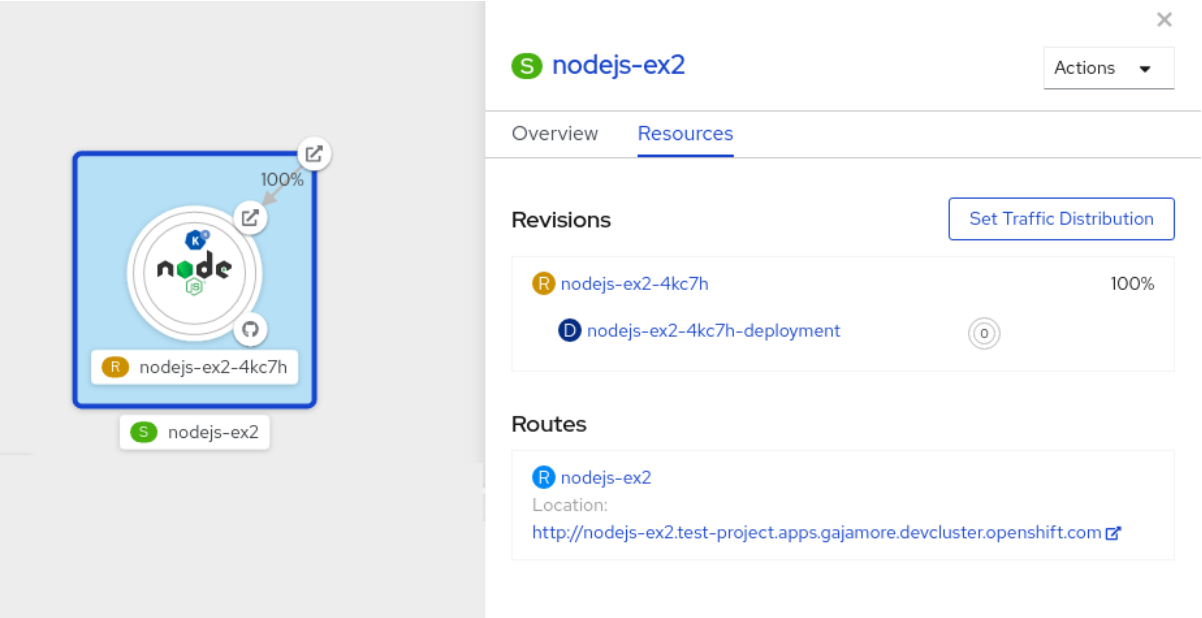
프로세스

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 6.1. 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.

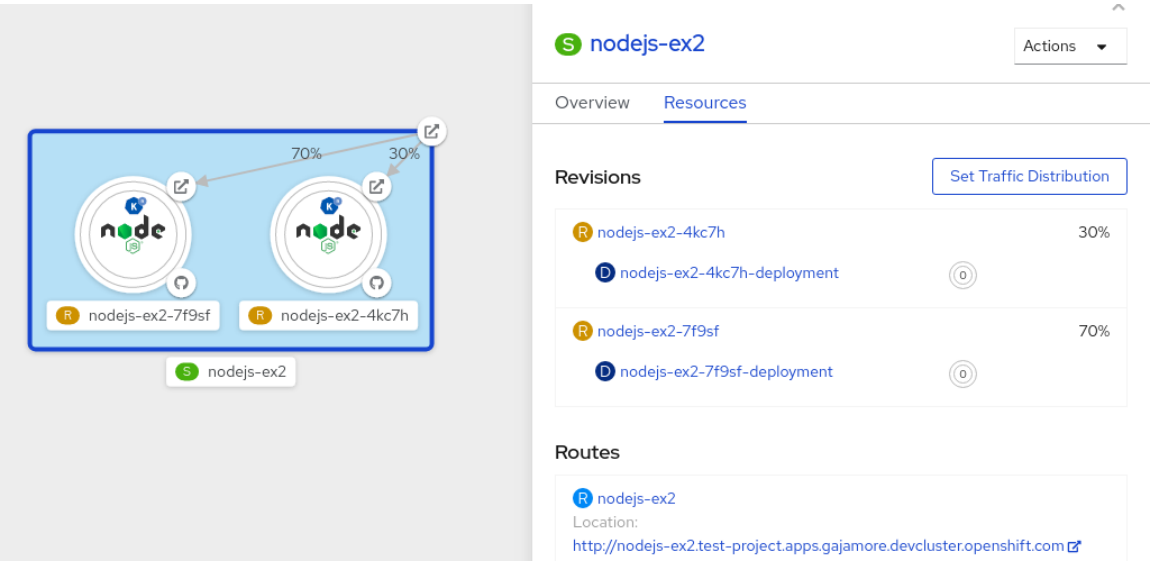
5. 리소스 탭에서 트래픽 배포 설정을 클릭하여 트래픽 배포 대화 상자를 확인합니다.

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

   b. 두 버전에 대한 사용자 정의 **URL**을 생성하도록 태그를 추가합니다.

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

그림 6.2. 서버리스 애플리케이션의 버전



6.6. BLUE-GREEN 전략을 사용하여 트래픽 다시 라우팅

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

6.6.1. 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.

프로세스

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 showcase -o=jsonpath='{.status.latestCreatedRevisionName}'
```

출력 예

```
$ showcase-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.

7장. 외부 및 INGRESS 라우팅

7.1. 라우팅 개요

Knative에서는 OpenShift Container Platform TLS 종료를 활용하여 Knative 서비스에 대한 라우팅을 제공합니다. Knative 서비스가 생성되면 서비스에 대한 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.

7.1.1. OpenShift Container Platform에 대한 추가 리소스

- 경로별 주석

7.2. 라벨 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.

7.2.1. 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.

프로세스

1. **OpenShift Container Platform** 경로에 전달할 레이블 또는 주석이 포함된 **Knative** 서비스를 생성합니다.

- **YAML**을 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) CLI**를 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.

7.3. 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.

7.3.1. 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.

프로세스

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 사항: 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>
```

7.4. 외부 경로의 URL 스키마

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

7.4.1. 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"
...
```

7.5. 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 클러

스터 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.

7.5.1. 클러스터 가용성을 클러스터 로컬로 설정

사전 요구 사항

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

프로세스

- **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 **URL**이 **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
```

7.5.2. 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.

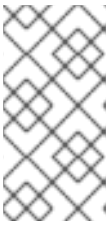
사전 요구 사항

- **OpenShift Serverless Operator** 및 **Knative Serving**이 설치되어 있습니다.
- 관리자 권한이 있습니다.
- **OpenShift(oc) CLI**가 설치되어 있습니다.

프로세스

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

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



참고

이러한 인증서가 **< app_name>.<namespace>.svc.cluster.local** 에 요청을 제공하려면 제목 대체 이름**(SAN)** 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 다시 시작하지 않고 인증서를 설정하여 **Pod**를 다시 시작할 필요가 없도록 합니다.

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

7.6. KOURIER GATEWAY 서비스 유형

Kourier Gateway는 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
...
```

7.6.1. Kourier Gateway 서비스 유형 설정

service-type 사양을 수정하여 로드 밸런서 서비스 유형을 대신 사용하도록 기본 서비스 유형을 덮어쓸 수 있습니다.

LoadBalancer 덮어쓰기 사양

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

7.7. 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 사용

하는 경우 수신 게이트웨이를 사용하여 애플리케이션을 직접 호출해야 합니다. 이를 위해서는 수신 게이트웨이의 공용 주소와 애플리케이션의 특정 호스트를 찾아야 합니다.

7.7.1. HTTP2 및 gRPC를 사용하여 서버리스 애플리케이션과 상호 작용

중요

이 방법은 OpenShift Container Platform 4.10 이상에 적용됩니다. 이전 버전의 경우 다음 섹션을 참조하십시오.

사전 요구 사항

- 클러스터에 OpenShift Serverless Operator 및 Knative Serving을 설치합니다.
- OpenShift CLI(oc)를 설치합니다.
- Knative 서비스를 생성합니다.
- OpenShift Container Platform 4.10 이상을 업그레이드합니다.
- OpenShift Ingress 컨트롤러에서 HTTP/2를 활성화합니다.

프로세스

1. 서버리스.openshift.io/default-enable-http2=true 주석을 KnativeServing 사용자 정의 리소스에 추가합니다.

```
$ oc annotate knativeserving <your_knative_CR> -n knative-serving
serverless.openshift.io/default-enable-http2=true
```

2. 주석이 추가된 후 Kourier 서비스의 appProtocol 값이 h2c:인지 확인할 수 있습니다.

```
$ oc get svc -n knative-serving-ingress kourier -o jsonpath="
{.spec.ports[0].appProtocol}"
```

출력 예

```
h2c
```

3. 이제 외부 트래픽에 HTTP/2 프로토콜을 통해 gRPC 프레임워크를 사용할 수 있습니다. 예를 들면 다음과 같습니다.

```
import "google.golang.org/grpc"

grpc.Dial(
  YOUR_URL, (1)
  grpc.WithTransportCredentials(insecure.NewCredentials())), (2)
)
```

(1) ksvc URL.

(2) 인증서입니다.

추가 리소스

- HTTP/2 수신 연결 사용

7.8. OPENSHIFT INGRESS 샤딩으로 SERVING 사용

OpenShift Ingress 샤딩과 함께 Knative Serving을 사용하여 도메인에 따라 Ingress 트래픽을 분할할 수 있습니다. 이를 통해 네트워크 트래픽을 클러스터의 다른 부분으로 보다 효율적으로 관리하고 라우팅할 수 있습니다.

참고

OpenShift Ingress 샤딩이 있는 경우에도 OpenShift Serverless 트래픽은 여전히 단일 Knative Ingress Gateway 및 knative-serving 프로젝트의 활성화 구성 요소를 통해 라우팅됩니다.

네트워크 트래픽 격리에 대한 자세한 내용은 Service Mesh를 사용하여 OpenShift Serverless에서 네트워크 트래픽을 격리하는 방법을 참조하십시오.

사전 요구 사항

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

7.8.1. OpenShift ingress shard 구성

Knative Serving을 구성하기 전에 OpenShift Ingress shard를 구성해야 합니다.

프로세스

- IngressController CR의 라벨 선택기를 사용하여 다른 도메인과 특정 Ingress shard와 일치하도록 OpenShift Serverless를 구성합니다.

IngressController CR 예

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: ingress-dev 1
  namespace: openshift-ingress-operator
spec:
  routeSelector:
    matchLabels:
      router: dev 2
  domain: "dev.serverless.cluster.example.com" 3
  # ...
---
```

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: ingress-prod 4
  namespace: openshift-ingress-operator
spec:
  routeSelector:
    matchLabels:
      router: prod 5
  domain: "prod.serverless.cluster.example.com" 6
  # ...
```

1 첫 번째 Ingress shard의 이름입니다.

2 ingress-dev shard와 일치하는 라벨 선택기입니다.

3 ingress-dev shard의 사용자 정의 도메인입니다.

4 두 번째 ingress shard의 이름입니다.

5 ingress-prod shard와 일치하는 라벨 선택기입니다.

6 ingress-prod shard의 사용자 정의 도메인입니다.

7.8.2. KnativeServing CR에서 사용자 정의 도메인 구성

OpenShift Ingress shard를 구성한 후 Knative Serving을 해당 shard와 일치하도록 구성해야 합니다.

프로세스

- KnativeServing CR에서 spec.config.domain 필드를 추가하여 Ingress shard와 동일한 도메인 및 레이블을 사용하도록 Serving을 구성합니다.

KnativeServing CR의 예

```
spec:
  config:
    domain: 1
      dev.serverless.cluster.example.com: |
        selector:
          router: dev
      prod.serverless.cluster.example.com: |
        selector:
          router: prod
  # ...
```

1 이러한 값은 ingress shard 구성의 값과 일치해야 합니다.

7.8.3. Knative 서비스에서 특정 Ingress shard를 대상으로

Ingress 샤딩 및 Knative Serving을 구성한 후 라벨을 사용하여 Knative 서비스 리소스의 특정 Ingress shard를 대상으로 할 수 있습니다.

프로세스

- Service CR에서 특정 shard와 일치하는 라벨 선택기를 추가합니다.

Service CR의 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: hello-dev
  labels:
    router: dev 1
```

```
spec:
  template:
    spec:
      containers:
      - image: docker.io/openshift/hello-openshift
---
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: hello-prod
  labels:
    router: prod 2
spec:
  template:
    spec:
      containers:
      - image: docker.io/openshift/hello-openshift
  # ...
```

1 2

레이블은 **KnativeServing CR**의 구성과 일치해야 합니다.

7.8.4. OpenShift Ingress 샤딩 구성으로 Serving 확인

Ingress 샤딩, **Knative Serving** 및 서비스를 구성한 후 서비스가 올바른 경로와 선택한 **Ingress shard**를 사용하는지 확인할 수 있습니다.

프로세스

1. 다음 명령을 실행하여 클러스터의 서비스에 대한 정보를 출력합니다.

```
$ oc get ksvc
```

출력 예

```
NAME        URL                                           LATESTCREATED
LATESTREADY       READY   REASON
hello-dev   https://hello-dev-default.dev.serverless.cluster.example.com    hello-dev-
00001   hello-dev-00001    True
hello-prod   https://hello-prod-default.prod.serverless.cluster.example.com  hello-prod-
00001   hello-prod-00001   True
```

2. 다음 명령을 실행하여 서비스에서 올바른 경로와 선택한 **Ingress shard**를 사용하는지 확인합니다.

```
$ oc get route -n knative-serving-ingress -o jsonpath='{range .items[*]}
{@.metadata.name}{" "}{@.spec.host}{" "}{@.status.ingress[*].routerName}{"\n"}{end}'
```

출력 예

```
route-19e6628b-77af-4da0-9b4c-1224934b2250-323461616533 hello-prod-
default.prod.serverless.cluster.example.com ingress-prod
route-cb5085d9-b7da-4741-9a56-96c88c6adaaa-373065343266 hello-dev-
default.dev.serverless.cluster.example.com ingress-dev
```

8장. HTTP 구성

8.1. 글로벌 HTTPS 리디렉션

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

8.1.1. 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"
...
```

8.2. 서비스당 HTTPS 리디렉션

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

8.2.1. 서비스에 대한 HTTPS 리디렉션

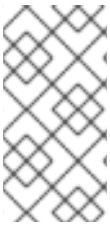
다음 예제에서는 Knative Service YAML 오브젝트에서 이 주석을 사용하는 방법을 보여줍니다.

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example
  namespace: default
  annotations:
```

```
    networking.knative.dev/http-protocol: "redirected"
spec:
  ...
```

8.3. HTTP/1에 대한 전체 중복 지원

features.knative.dev/http-full-duplex 주석을 구성하여 서비스에 대한 HTTP/1 전체 duplex 지원을 활성화할 수 있습니다.



참고

이전 버전 클라이언트가 HTTP/1 전체 중복을 지원하지 않을 수 있으므로 활성화하기 전에 HTTP 클라이언트를 확인합니다.

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

HTTP/1에 대한 전체 중복 지원을 제공하는 KnativeServing CR의 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
  template:
    spec:
      annotations:
        features.knative.dev/http-full-duplex: "Enabled"
...
```

9장. KNATIVE 서비스에 대한 액세스 구성

9.1. KNATIVE 서비스의 JSON WEB TOKEN 인증 설정

OpenShift Serverless에는 현재 사용자 정의 권한 부여 기능이 없습니다. 배포에 사용자 정의 권한을 추가하려면 OpenShift Serverless를 Red Hat OpenShift Service Mesh와 통합한 다음 Knative 서비스에 대한 JSON 웹 토큰(JWT) 인증 및 사이드카 삽입을 구성해야 합니다.

9.2. SERVICE MESH 2.X에서 JSON 웹 토큰 인증 사용

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

9.2.1. 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.

OpenShift Container Platform의 경우 이러한 네임스페이스의 Pod에 사이드카 삽입이 필요한 경우 OpenShift Serverless *와 OpenShift Serverless 통합에 대한 OpenShift Serverless 설명서*를 참조하십시오.

사전 요구 사항

- 클러스터에 OpenShift Serverless Operator, Knative Serving 및 Red Hat OpenShift Service Mesh를 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.

프로세스

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** 이상이 기본적으로 **Knative** 서비스의 준비 상태 프로브로 **HTTP** 프로브를 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 리소스에 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 **URL**을 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 **JWT**로 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!
```

9.3. SERVICE MESH 1.X에서 JSON 웹 토큰 인증 사용

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

9.3.1. 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.

OpenShift Container Platform의 경우 이러한 네임스페이스의 Pod에 사이드카 삽입이 필요한 경우 OpenShift Serverless *와 OpenShift Serverless 통합에 대한 OpenShift Serverless 설명서*를 참조하십시오.

사전 요구 사항

- 클러스터에 OpenShift Serverless Operator, Knative Serving 및 Red Hat OpenShift Service Mesh를 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.

프로세스

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
...
```

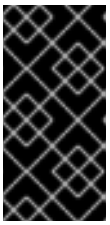
1. sidecar.istio.io/inject="true" 주석을 추가합니다.
2. OpenShift Serverless 버전 1.14.0 이상이 기본적으로 Knative 서비스의 준비 상태 프로브로 HTTP 프로브를 사용하므로 Knative 서비스에서 주석 sidecar.istio.io/rewriteAppHTTPProbers: "true" 를 설정해야 합니다.

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!
```

10장. SERVING에 대한 KUBE-RBAC-PROXY 구성

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

10.1. SERVING에 대한 KUBE-RBAC-PROXY 리소스 구성

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

참고

특정 배포에 대한 리소스 할당을 덮어쓸 수도 있습니다.

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

KnativeServing CR 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  config:
    deployment:
      "kube-rbac-proxy-cpu-request": "10m" 1
      "kube-rbac-proxy-memory-request": "20Mi" 2
      "kube-rbac-proxy-cpu-limit": "100m" 3
      "kube-rbac-proxy-memory-limit": "100Mi" 4
```

1

최소 CPU 할당을 설정합니다.

2

3

최대 **CPU** 할당을 설정합니다.

4

최대 **RAM** 할당을 설정합니다.

11장. NET-KOURIER의 버스 및 QPS 구성

초당 쿼리(QPS) 및 버스트 값은 API 서버에 대한 요청 또는 API 호출 빈도를 결정합니다.

11.1. NET-KOURIER의 BURST 및 QPS 값 구성

QPS(초당 쿼리) 값에 따라 API 서버로 전송되는 클라이언트 요청 또는 API 호출 수가 결정됩니다.

burst 값은 클라이언트에 대해 저장할 수 있는 요청 수를 결정합니다. 이 버퍼를 초과하는 요청은 삭제됩니다. 이 기능은 버스트이며 요청을 시간에 균일하게 분배하지 않는 컨트롤러에 유용합니다.

net-kourier-controller 가 다시 시작되면 클러스터에 배포된 모든 수신 리소스를 구문 분석하므로 많은 API 호출이 발생합니다. 이로 인해 net-kourier-controller 를 시작하는 데 시간이 오래 걸릴 수 있습니다.

KnativeServing CR에서 net-kourier-controller 의 QPS 및 버스트 값을 조정할 수 있습니다.

KnativeServing CR 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  workloads:
  - name: net-kourier-controller
    env:
    - container: controller
      envVars:
      - name: KUBE_API_BURST
        value: "200" (1)
      - name: KUBE_API_QPS
        value: "200" (2)
```

(1)

컨트롤러와 API 서버 간의 통신 속도입니다. 기본값은 200입니다.

2

Kubelet과 API 서버 간의 통신 버스트 용량입니다. 기본값은 200입니다.

12장. KNATIVE 서비스의 사용자 정의 도메인 구성

12.1. KNATIVE 서비스의 사용자 정의 도메인 구성

Knative 서비스에는 클러스터 구성에 따라 기본 도메인 이름이 자동으로 할당됩니다. 예를 들면 <service_name>-<namespace>.example.com 입니다. 소유한 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.

12.2. 사용자 정의 도메인 매핑

소유한 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.

12.2.1. 사용자 정의 도메인 매핑 생성

소유한 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.

프로세스

1. 매핑하려는 대상 **CR**과 동일한 네임스페이스에 **DomainMapping CR**을 포함하는 **YAML** 파일을 생성합니다.

```
apiVersion: serving.knative.dev/v1beta1
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 **CR**에 매핑할 사용자 정의 도메인 이름입니다.

2 **DomainMapping CR** 및 대상 **CR**의 네임스페이스입니다.

3 사용자 정의 도메인에 매핑할 대상 **CR**의 이름입니다.

4 사용자 지정 도메인에 매핑되는 **CR** 유형입니다.

서비스 도메인 매핑 예

```
apiVersion: serving.knative.dev/v1beta1
kind: DomainMapping
metadata:
 name: example.com
```

```
 namespace: default
spec:
 ref:
   name: showcase
   kind: Service
   apiVersion: serving.knative.dev/v1
```

경로 도메인 매핑 예

```
apiVersion: serving.knative.dev/v1beta1
kind: DomainMapping
metadata:
 name: example.com
 namespace: default
spec:
 ref:
   name: example-route
   kind: Route
   apiVersion: serving.knative.dev/v1
```

2. DomainMapping CR을 YAML 파일로 적용합니다.

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

12.3. KNATIVE CLI를 사용한 KNATIVE 서비스의 사용자 정의 도메인

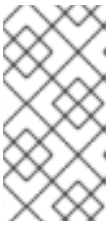
소유한 사용자 정의 도메인 이름을 Knative 서비스에 매핑하여 Knative 서비스의 도메인을 사용자 지정할 수 있습니다. Knative(kn) CLI를 사용하여 Knative 서비스 또는 Knative 경로와 같이 주소 지정 가능 대상 CR에 매핑되는 DomainMapping CR(사용자 정의 리소스)을 생성할 수 있습니다.

12.3.1. 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.

프로세스

- 현재 네임스페이스의 CR에 도메인을 매핑합니다.

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

명령 예

```
$ kn domain create example.com --ref showcase
```

--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.com --ref ksvc:showcase:example-namespace
```

- 도메인을 Knative 경로에 매핑합니다.

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

명령 예

```
$ kn domain create example.com --ref kroute:example-route
```

12.4. 개발자 화면을 사용한 도메인 매핑

소유한 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.

12.4.1. 개발자 화면을 사용하여 사용자 정의 도메인을 서비스에 매핑

사전 요구 사항

- 웹 콘솔에 로그인했습니다.
- 개발자 화면에 있습니다.
- OpenShift Serverless Operator 및 Knative Serving이 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.

프로세스

1. 토폴로지 페이지로 이동합니다.
2. 도메인에 매핑할 서비스를 마우스 오른쪽 버튼으로 클릭하고 서비스 이름이 포함된 편집 옵션을 선택합니다. 예를 들어 서비스 이름이 **show**인 경우 **Edit Show** 옵션을 선택합니다.
3. 고급 옵션 섹션에서 고급 라우팅 옵션 표시를 클릭합니다.
   a. 서비스에 매핑하려는 도메인 매핑 **CR**이 이미 존재하는 경우 도메인 매핑 목록에서 이를 선택할 수 있습니다.
   b. 새 도메인 매핑 **CR**을 생성하려면 도메인 이름을 상자에 입력하고 만들기 옵션을 선택합니다. 예를 들어 **example.com** 을 입력하는 경우 만들기 옵션은 **Create "example.com"** 입니다.
4. 저장을 클릭하여 서비스 변경 사항을 저장합니다.

검증

1. 토폴로지 페이지로 이동합니다.
2. 생성한 서비스를 클릭합니다.

3. 서비스 정보 창의 리소스 탭에서는 도메인 매핑에 나열된 서비스에 매핑한 도메인을 확인할 수 있습니다.

12.5. 관리자 화면을 사용한 도메인 매핑

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

12.5.1. 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. 소유한 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.

OpenShift Container Platform (또는 OpenShift Dedicated 또는 Red Hat OpenShift Service on AWS에서 Red Hat OpenShift Service)에 대한 클러스터 관리자 권한이 있는 경우 웹 콘솔의 관리자 화면을 사용하여 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 생성하거나 애플리케이션 및 기타 워크로드를 생성할 수 있는 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

- Knative 서비스를 생성했으며 해당 서비스에 매핑할 사용자 정의 도메인을 제어할 수 있습니다.

참고

사용자 정의 도메인은 클러스터의 IP 주소를 가리켜야 합니다.

프로세스

1. CustomResourceDefinitions 로 이동하여 검색 상자를 사용하여 DomainMapping CRD(사용자 정의 리소스 정의)를 찾습니다.

2. DomainMapping CRD를 클릭한 다음 Instances 탭으로 이동합니다.

3. 도메인 매핑 생성을 클릭합니다.

4. 인스턴스에 대한 다음 정보를 포함하도록 DomainMapping CR의 YAML을 수정합니다.

```
apiVersion: serving.knative.dev/v1beta1
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 매핑의 예

```
apiVersion: serving.knative.dev/v1beta1
kind: DomainMapping
metadata:
 name: custom-ksvc-domain.example.com
 namespace: default
spec:
 ref:
   name: showcase
   kind: Service
   apiVersion: serving.knative.dev/v1
```

검증

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

명령 예

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

출력 예

```
{"artifact":"knative-showcase","greeting":"Welcome"}
```

12.5.2. 관리자 화면을 사용하여 암호화 제품군 제한

수신에 **net-kourier** 를 지정하고 **DomainMapping** 을 사용하면 **OpenShift** 라우팅의 **TLS**가 **passthrough**로 설정되고 **Kourier Gateway**에서 **TLS**를 처리합니다. 이러한 경우 **Kourier**에 사용할 수 있는 **TLS** 암호화 제품군을 제한해야 할 수 있습니다.

사전 요구 사항

- 웹 콘솔에 로그인했습니다.
- 관리자 화면에 있습니다.
- **OpenShift Serverless Operator**를 설치했습니다.
- **Knative Serving**이 설치되어 있습니다.
- 프로젝트를 생성하거나 애플리케이션 및 기타 워크로드를 생성할 수 있는 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

참고

사용자 정의 도메인은 클러스터의 **IP** 주소를 가리켜야 합니다.

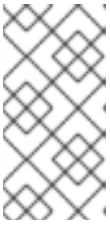
프로세스

- **KnativeServing CR**에서 **cipher-suites** 값을 사용하여 활성화할 암호화 제품군을 지정합니다.

KnativeServing CR 예

```
spec:
  config:
    kourier:
      cipher-suites: ECDHE-ECDSA-AES128-GCM-SHA256,ECDHE-ECDSA-CHACHA20-
POLY1305
```

기타 암호화 제품군은 비활성화됩니다. 여러 제품군을 쉼표로 구분하여 지정할 수 있습니다.



참고

Kourier 게이트웨이의 컨테이너 이미지는 Envoy 프록시 이미지를 사용하며 기본 활성화된 암호화 제품군은 Envoy 프록시 버전에 따라 다릅니다.

12.6. TLS 인증서를 사용하여 매핑된 서비스 보안

12.6.1. TLS 인증서를 사용하여 사용자 정의 도메인으로 서비스 보안

Knative 서비스에 대한 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.

사전 요구 사항

- Knative 서비스에 대한 사용자 정의 도메인을 구성하고 DomainMapping CR이 작동합니다.
- 인증 기관 공급자의 TLS 인증서 또는 자체 서명된 인증서가 있습니다.
- 인증 기관 공급자 또는 자체 서명된 인증서에서 인증서 및 키 파일을 가져왔습니다.
- OpenShift CLI(oc)를 설치합니다.

프로세스

1. Kubernetes TLS 시크릿을 생성합니다.

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

2. Kubernetes TLS 보안에 networking.internal.knative.dev/certificate-uid: <id>' 레이블을

추가합니다.

```
$ oc label secret <tls_secret_name> networking.internal.knative.dev/certificate-uid="
<id>"
```

cert-manager와 같은 타사 시크릿 공급자를 사용하는 경우 Kubernetes TLS 보안에 레이블을 지정하도록 시크릿 관리자를 구성할 수 있습니다. cert-manager 사용자는 제공되는 시크릿 템플릿을 사용하여 올바른 레이블로 시크릿을 자동으로 생성할 수 있습니다. 이 경우 키만 기반으로 보안 필터링이 수행되지만 이 값은 시크릿에 포함된 인증서 ID와 같은 유용한 정보를 전송할 수 있습니다.

참고

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

3. 생성한 TLS 시크릿을 사용하도록 DomainMapping CR을 업데이트합니다.

```
apiVersion: serving.knative.dev/v1beta1
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 와 함께 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.

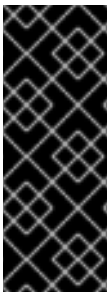
```
$ curl https://<domain_name>
```

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

12.6.2. 시크릿 필터링을 사용하여 net-kourier 메모리 사용량 개선

기본적으로 **Kubernetes client-go** 라이브러리에 대한 정보 제공자 는 특정 유형의 모든 리소스를 가져옵니다. 이로 인해 많은 리소스가 사용 가능한 경우 상당한 오버헤드가 발생할 수 있으므로 메모리 누수로 인해 대규모 클러스터에서 **Knative net-kourier Ingress** 컨트롤러가 실패할 수 있습니다. 그러나 필터링 메커니즘은 **Knative net-kourier** 수신 컨트롤러에서 사용할 수 있으므로 컨트롤러가 **Knative** 관련 시크릿만 가져올 수 있습니다.

OpenShift Serverless Operator 측에서는 기본적으로 보안 필터링이 활성화됩니다. 환경 변수 **ENABLE_SECRET_INFORMER_BY_CERT_UID=true** 가 기본적으로 **net-kourier** 컨트롤러 포드에 추가됩니다.



중요

보안 필터링을 활성화하는 경우 모든 시크릿에 **networking.internal.knative.dev/certificate-uid: "<id>"** 로 레이블이 지정되어야 합니다. 그렇지 않으면 **Knative Serving**이 탐지되지 않아 오류가 발생합니다. 새 보안 및 기존 보안에 레이블을 지정해야 합니다.

사전 요구 사항

- **OpenShift Container Platform**에 대한 클러스터 관리자 권한이 있거나 **AWS** 또는 **OpenShift Dedicated**의 **Red Hat OpenShift Service**에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- 애플리케이션 및 기타 워크로드를 생성할 수 있는 역할 및 권한이 있거나 생성한 프로젝트입니다.

- OpenShift Serverless Operator 및 Knative Serving을 설치합니다.
- OpenShift CLI(oc)를 설치합니다.

KnativeServing CR(사용자 정의 리소스)의 workloads 필드를 사용하여 ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID 변수를 false 로 설정하여 시크릿 필터링을 비활성화할 수 있습니다.

KnativeServing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
...
  workloads:
    - env:
        - container: controller
          envVars:
            - name: ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID
              value: 'false'
      name: net-kourier-controller
```

13장. KNATIVE SERVING의 고가용성 구성

13.1. 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 서비스 중인 API의 처리를 대신합니다.

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

13.2. KNATIVE 배포를 위한 고가용성

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

13.2.1. Knative Serving의 고가용성 복제본 구성

적합한 배포 리소스에 대해 최소 복제본 3개를 지정하려면 사용자 정의 리소스의 spec.high-availability.replicas 필드 값을 3 으로 설정합니다.

사전 요구 사항

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

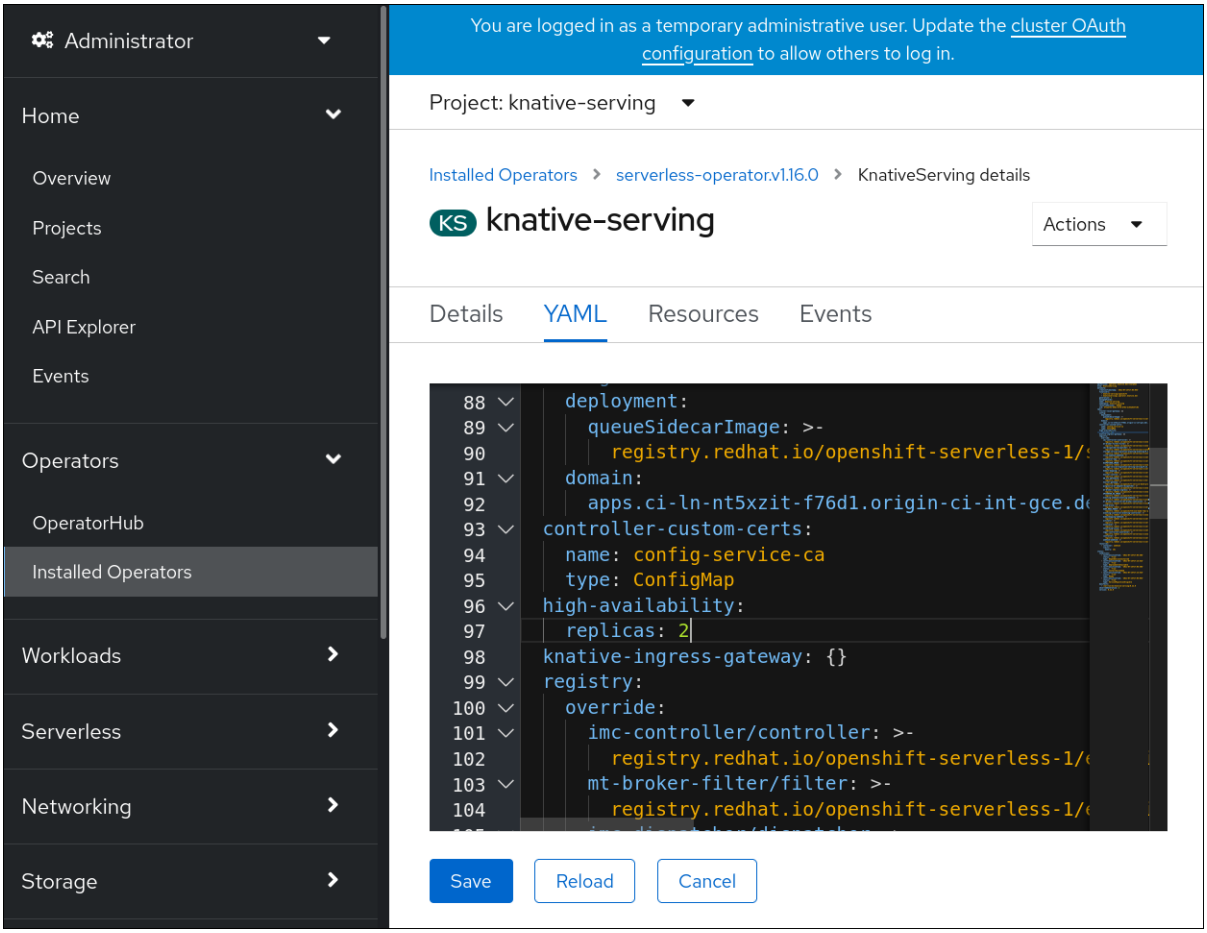
프로세스

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 Serving을 클릭하여 Knative Serving 탭으로 이동합니다.**

4. **knative-serving을 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
```

14장. 튜닝 제공 구성

14.1. KNATIVE SERVING 시스템 배포 구성 덮어쓰기

KnativeServing CR(사용자 정의 리소스)의 **workloads** 사양을 수정하여 일부 특정 배포의 기본 구성을 덮어쓸 수 있습니다.

14.1.1. 시스템 배포 구성 덮어쓰기

현재는 프로브 준비 및 활성 필드뿐만 아니라 리소스,복제본,레이블, **nodeSelector** 필드에 대해 기본 구성 설정을 재정의할 수 있습니다.

다음 예에서 **KnativeServing CR**은 다음과 같이 웹 후크 배포를 덮어씁니다.

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

참고

KnativeServing CR 레이블 및 주석 설정은 배포 자체와 결과 **Pod** 모두에 대한 배포의 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
  workloads:
  - 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 및 활성 상태 프로브 덮어쓰기를 사용하여 프로브 처리기와 관련된 필드를 제외하고 Kubernetes API에 지정된 대로 배포 컨테이너에서 프로브의 모든 필드를 덮어쓸 수 있습니다. exec,grpc,httpGet, tcpSocket.

추가 리소스

- Kubernetes API 문서의 프로브 구성 섹션

15장. 큐 프록시 리소스 구성

Queue 프록시는 서비스 내의 각 애플리케이션 컨테이너에 대한 사이드카 컨테이너입니다. Serverless 워크로드 관리를 개선하여 리소스 사용량을 효율적으로 보장합니다. Queue 프록시를 구성할 수 있습니다.

15.1. KNATIVE 서비스에 대한 큐 프록시 리소스 구성

배포 구성 맵에서 큐 프록시 리소스 요청 및 제한을 전역적으로 구성하는 것 외에도 CPU, 메모리 및 임시 스토리지 리소스 유형을 대상으로 하는 해당 주석을 사용하여 서비스 수준에서 설정할 수 있습니다.

사전 요구 사항

- Red Hat OpenShift Pipelines가 클러스터에 설치되어 있어야 합니다.
- OpenShift(oc) CLI가 설치되어 있습니다.
- Knative(kn) CLI가 설치되어 있습니다.

프로세스

- 리소스 요청 및 제한을 사용하여 서비스의 configmap을 수정합니다.

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
  template:
    metadata:
      annotations:
        queue.sidecar.serving.knative.dev/cpu-resource-request: "1"
        queue.sidecar.serving.knative.dev/cpu-resource-limit: "2"
        queue.sidecar.serving.knative.dev/memory-resource-request: "1Gi"
        queue.sidecar.serving.knative.dev/memory-resource-limit: "2Gi"
        queue.sidecar.serving.knative.dev/ephemeral-storage-resource-request: "400Mi"
        queue.sidecar.serving.knative.dev/ephemeral-storage-resource-limit: "450Mi"
```

또는 Queue 프록시 리소스를 애플리케이션 컨테이너의 백분율로 계산하는 특수 queue.sidecar.serving.knative.dev/resource-percentage 주석을 사용할 수 있습니다. CPU 및

메모리 리소스 요구 사항이 애플리케이션 컨테이너 요구 사항에서 계산되고 아래 경계 외부에 있는 경우 값이 경계에 맞게 조정됩니다. 이 경우 **CPU** 및 메모리 리소스 요구 사항에 다음 최소 및 최대 경계가 적용됩니다.

표 15.1. 리소스 요구 사항 경계

리소스 요구 사항	분	Max
CPU 요청	25m	100m
CPU 제한	40m	500m
메모리 요청	50Mi	200Mi
메모리 제한	200Mi	500Mi

참고

해당 리소스 주석을 사용하여 백분율 주석과 특정 리소스 값을 동시에 설정하는 경우 후자가 우선합니다.

주의

queue.sidecar.serving.knative.dev/resource-percentage 주석이 더 이상 사용되지 않으며 향후 **OpenShift Serverless** 버전에서 제거됩니다.