Red Hat OpenShift Serverless 1.33

통합

OpenShift Serverless와 Service Mesh 통합 및 비용 관리 서비스

Last Updated: 2024-09-18

Red Hat OpenShift Serverless 1.33 통합

OpenShift Serverless와 Service Mesh 통합 및 비용 관리 서비스

법적 공지

초록

이 문서에서는 Service Mesh를 OpenShift Serverless와 통합하는 방법에 대한 정보를 제공합니다. 또한 비용 관리 서비스를 사용하여 비용을 이해하고 추적하며 서버리스 애플리케이션에서 NVIDIA GPU 리소스를 사용하는 방법을 보여줍니다.

차례

1장. 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 인그레스가 지원하지 않는 추가 네트워킹 및 라우팅 옵션을 구성할 수 있습니다.

다음과 같은 가정 및 제한 사항에 유의하십시오.

- 모든 Knative 내부 구성 요소와 Knative 서비스는 서비스 메시의 일부이며 사이드카 삽입이 활성화되어 있습니다. 즉, 엄격한 mTLS가 전체 메시 내에서 적용됩니다. Knative 서비스에 대한 모든 요청에는 OpenShift 라우팅에서 들어오는 호출을 제외하고 클라이언트가 인증서를 보내야 하는 mTLS 연결이 필요합니다.

- Service Mesh를 사용한 OpenShift Serverless 통합은 **하나의** 서비스 메시만 대상으로 할 수 있습니다. 클러스터에 여러 메시가 존재할 수 있지만 OpenShift Serverless는 해당 중 하나에서만 사용할 수 있습니다.

- OpenShift Serverless가 포함된 대상 **ServiceMeshMemberRoll** 을 변경하면 OpenShift Serverless를 다른 메시로 이동하는 것은 지원되지 않습니다. 대상 서비스 메시를 변경하는 유일한 방법은 OpenShift Serverless를 설치 제거하고 다시 설치하는 것입니다.

1.1. 사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 Red Hat OpenShift Serverless 계정에 액세스할 수 있습니다.

- OpenShift CLI(**oc**)가 설치되어 있습니다.

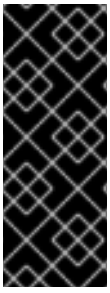
- Serverless Operator를 설치했습니다.

- Red Hat OpenShift Service Mesh Operator가 설치되어 있습니다.

- 다음 절차의 예제에서는 도메인 **example.com**을 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.com**인 경우 도메인이 ***.apps.openshift.example.com**이 되도록 와일드카드 인증서를 구성해야 합니다. 와일드카드 인증서 구성에 대한 자세한 내용은 *수신되는 외부 트래픽을 암호화하기 위한 인증서 생성*에 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.



중요

OpenShift Serverless는 본 안내서에 명시되어 있는 Red Hat OpenShift Service Mesh 기능만 지원하며 문서화되지 않은 다른 기능은 지원하지 않습니다.

Service Mesh에서 Serverless 1.31 사용은 Service Mesh 버전 2.2 이상에서만 지원됩니다. 1.31 이외의 버전에 대한 자세한 내용 및 정보는 "Red Hat OpenShift Serverless 지원 구성" 페이지를 참조하십시오.

1.2. 추가 리소스

- Red Hat OpenShift Serverless 지원 구성
- Kourier 및 Istio 인그레스

1.3. 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

- 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 설치되어 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.
- 프로젝트를 생성하거나 애플리케이션 및 기타 워크로드를 생성할 수 있는 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

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 Serverless를 Service Mesh와 통합할 때 생성된 게이트웨이에서 선택하여 수신 게이트웨이가 이 인증서와 트래픽을 제공하도록 합니다.

1.4. OPENSHIFT SERVERLESS와 SERVICE MESH 통합

1.4.1. 설치 사전 요구 사항 확인

Serverless와 Service Mesh 통합을 설치하고 구성하기 전에 사전 요구 사항이 충족되었는지 확인합니다.

프로세스

1. 충돌하는 게이트웨이를 확인합니다.

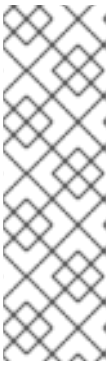
명령 예

```
$ oc get gateway -A -o jsonpath='{range .items[*]}{@.metadata.namespace}{"/"}
{@.metadata.name}{" "}{@.spec.servers}{"\n"}{end}' | column -t
```

출력 예

```
knative-serving/knative-ingress-gateway  [{"hosts":["*"],"port":
{"name":"https","number":443,"protocol":"HTTPS"},"tls":{"credentialName":"wildcard-
certs","mode":"SIMPLE"}}]
knative-serving/knative-local-gateway    [{"hosts":["*"],"port":
{"name":"http","number":8081,"protocol":"HTTP"}}]
```

이 명령은 다른 Service Mesh 인스턴스의 일부인 **knative-serving** 및 **Gateway** 의 **게이트웨이** 를 제외하고 **포트 443** 및 **호스트: ["*"]** 를 바인딩하는 **게이트웨이** 를 반환해서는 안 됩니다.



참고

Serverless에서 속하는 메시는 Serverless 워크로드에만 예약되어 있어야 합니다. 이는 게이트웨이와 같은 추가 구성이 Serverless 게이트웨이 **knative-local-gateway** 및 **knative-ingress-gateway** 를 방해할 수 있기 때문입니다. **Red Hat OpenShift Service Mesh는 하나의 게이트웨이가 동일한 포트(포트:443)에서 와일드카드호스트 바인딩(호스트: ["*"])을 요청할 수 있도록 허용합니다. 다른 게이트웨이가 이미 이 구성을 바인딩하고 있는 경우 Serverless 워크로드에 대해 별도의 메시를 생성해야 합니다.**

2. **Red Hat OpenShift Service Mesh istio-ingressgateway 가 NodePort 또는 LoadBalancer 유형으로 노출되는지 확인합니다.**

명령 예

```
$ oc get svc -A | grep istio-ingressgateway
```

출력 예

```
istio-system   istio-ingressgateway  ClusterIP  172.30.46.146 none>
15021/TCP,80/TCP,443/TCP     9m50s
```

이 명령은 **NodePort** 또는 **LoadBalancer** 유형의 **Service** 오브젝트를 반환하지 않아야 합니다.

참고

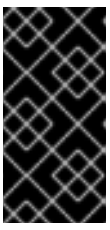
클러스터 외부 **Knative** 서비스는 **OpenShift** 경로를 사용하여 **OpenShift Ingress**를 통해 호출해야 합니다. **NodePort** 또는 **LoadBalancer** 유형의 **Service** 오브젝트를 사용하여 **istio-ingressgateway** 를 노출하는 것과 같이 서비스 메시에 직접 액세스하는 것은 지원되지 않습니다.

1.4.2. 서비스 메시 설치 및 구성

Serverless를 **Service Mesh**와 통합하려면 특정 구성으로 서비스 메시를 설치해야 합니다.

프로세스

1. 다음 구성을 사용하여 **istio-system** 네임스페이스에서 **ServiceMeshControlPlane** 리소스를 생성합니다.



중요

기존 **ServiceMeshControlPlane** 오브젝트가 있는 경우 동일한 구성이 적용되었는지 확인합니다.

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
```

```
  namespace: istio-system
spec:
  profiles:
  - default
  security:
    dataPlane:
      mtls: true (1)
  techPreview:
    meshConfig:
      defaultConfig:
        terminationDrainDuration: 35s (2)
  gateways:
    ingress:
      service:
        metadata:
          labels:
            knative: ingressgateway (3)
  proxy:
    networking:
      trafficControl:
        inbound:
          excludedPorts: (4)
          - 8444 # metrics
          - 8022 # serving: wait-for-drain k8s pre-stop hook
```

(1) 메시에서 엄격한 **mTLS**를 적용합니다. 유효한 클라이언트 인증서를 사용하는 호출만 허용됩니다.

(2) **Serverless**에서 **Knative** 서비스에 대한 정상 종료 시간은 **30**초입니다. **Istio-proxy** 에는 요청이 삭제되지 않도록 하려면 더 긴 종료 기간이 있어야 합니다.

(3) **Knative** 게이트웨이만 대상으로 할 수신 게이트웨이의 특정 선택기를 정의합니다.

(4) 이러한 포트는 **Kubernetes** 및 클러스터 모니터링에 의해 호출되며 메시의 일부가 아니며 **mTLS**를 사용하여 호출할 수 없습니다. 따라서 이러한 포트는 메시에서 제외됩니다.

2. **Service Mesh**와 통합할 네임스페이스를 **ServiceMeshMemberRoll** 오브젝트에 멤버로 추가합니다.

servicemesh-member-roll.yaml 구성 파일 예

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system
spec:
  members: 1
    - knative-serving
    - knative-eventing
    - your-OpenShift-projects
```

1

Service Mesh와 통합할 네임스페이스 목록입니다.

중요

이 네임스페이스 목록에 **knative-serving** 및 **knative-eventing** 네임스페이스가 포함되어야 합니다.

3. **ServiceMeshMemberRoll** 리소스를 적용합니다.

```
$ oc apply -f servicemesh-member-roll.yaml
```

4. 서비스 메시가 트래픽을 수락할 수 있도록 필요한 게이트웨이를 생성합니다. 다음 예제에서는 **ISTIO_MUTUAL** 모드(mTLS)와 함께 **knative-local-gateway** 오브젝트를 사용합니다.

istio-knative-gateways.yaml 구성 파일의 예

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: knative-ingress-gateway
  namespace: knative-serving
spec:
  selector:
    knative: 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:
   knative: ingressgateway
 servers:
   - port:
       number: 8081
       name: https
       protocol: HTTPS (2)
     tls:
       mode: ISTIO_MUTUAL (3)
     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 보안의 이름입니다.

(2) (3)

5. Gateway 리소스를 적용합니다.

```
$ oc apply -f istio-knative-gateways.yaml
```

1.4.3. Serverless 설치 및 구성

Service Mesh를 설치한 후 특정 구성으로 Serverless를 설치해야 합니다.

프로세스

1. Istio 통합을 활성화하는 다음 KnativeServing 사용자 정의 리소스를 사용하여 Knative Serving을 설치합니다.

knative-serving-config.yaml 구성 파일의 예

```
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"
  config:
    istio: (3)
      gateway.knative-serving.knative-ingress-gateway: istio-ingressgateway.<your-istio-namespace>.svc.cluster.local
      local-gateway.knative-serving.knative-local-gateway: knative-local-gateway.<your-istio-namespace>.svc.cluster.local
```

(1)

2

Knative Serving 데이터 플레인 Pod에 사이드카 삽입을 활성화합니다.

3

istio-system 네임스페이스가 istio-system 네임스페이스에서 실행되지 않는 경우 이 두 플래그를 올바른 네임스페이스로 설정해야 합니다.

2. KnativeServing 리소스를 적용합니다.

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

3. Istio 통합을 활성화하는 다음 KnativeEventing 오브젝트를 사용하여 Knative Eventing을 설치합니다.

knative-eventing-config.yaml 구성 파일의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  config:
    features:
      istio: enabled 1
  workloads: 2
  - name: pingsource-mt-adapter
    annotations:
      "sidecar.istio.io/inject": "true"
      "sidecar.istio.io/rewriteAppHTTPProbers": "true"
  - name: imc-dispatcher
    annotations:
      "sidecar.istio.io/inject": "true"
      "sidecar.istio.io/rewriteAppHTTPProbers": "true"
  - name: mt-broker-ingress
    annotations:
      "sidecar.istio.io/inject": "true"
      "sidecar.istio.io/rewriteAppHTTPProbers": "true"
  - name: mt-broker-filter
    annotations:
      "sidecar.istio.io/inject": "true"
      "sidecar.istio.io/rewriteAppHTTPProbers": "true"
```

1 Eventing Istio 컨트롤러를 활성화하여 각 InMemoryChannel 또는 KafkaChannel 서비스에 대한 DestinationRule 을 생성합니다.

2 Knative Eventing Pod에 사이드카 삽입을 활성화합니다.

4. KnativeEventing 리소스를 적용합니다.

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

5. Istio 통합을 활성화하는 다음 KnativeKafka 사용자 정의 리소스를 사용하여 Knative Kafka를 설치합니다.

knative-kafka-config.yaml 구성 파일의 예

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  name: knative-kafka
  namespace: knative-eventing
spec:
  channel:
    enabled: true
    bootstrapServers: <bootstrap_servers> 1
  source:
    enabled: true
  broker:
    enabled: true
    defaultConfig:
      bootstrapServers: <bootstrap_servers> 2
      numPartitions: <num_partitions>
      replicationFactor: <replication_factor>
    sink:
      enabled: true
  workloads: 3
  - name: kafka-controller
    annotations:
      "sidecar.istio.io/inject": "true"
      "sidecar.istio.io/rewriteAppHTTPProbers": "true"
```

```
- name: kafka-broker-receiver
  annotations:
    "sidecar.istio.io/inject": "true"
    "sidecar.istio.io/rewriteAppHTTPProbers": "true"
- name: kafka-broker-dispatcher
  annotations:
    "sidecar.istio.io/inject": "true"
    "sidecar.istio.io/rewriteAppHTTPProbers": "true"
- name: kafka-channel-receiver
  annotations:
    "sidecar.istio.io/inject": "true"
    "sidecar.istio.io/rewriteAppHTTPProbers": "true"
- name: kafka-channel-dispatcher
  annotations:
    "sidecar.istio.io/inject": "true"
    "sidecar.istio.io/rewriteAppHTTPProbers": "true"
- name: kafka-source-dispatcher
  annotations:
    "sidecar.istio.io/inject": "true"
    "sidecar.istio.io/rewriteAppHTTPProbers": "true"
- name: kafka-sink-receiver
  annotations:
    "sidecar.istio.io/inject": "true"
    "sidecar.istio.io/rewriteAppHTTPProbers": "true"
```

1 2

Apache Kafka 클러스터 **URL(**예**: my-cluster-kafka-bootstrap.kafka:9092).**

3

Knative Kafka Pod에 사이드카 삽입을 활성화합니다.

6. **KnativeEventing** 오브젝트를 적용합니다.

```
$ oc apply -f knative-kafka-config.yaml
```

7. **ServiceEntry** 를 설치하여 **KnativeKafka** 구성 요소와 **Apache Kafka** 클러스터 간의 통신에 대해 서비스 메시에 알립니다.

kafka-cluster-serviceentry.yaml 구성 파일의 예

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
 name: kafka-cluster
 namespace: knative-eventing
spec:
 hosts: 1
  - <bootstrap_servers_without_port>
 exportTo:
  - "."
 ports: 2
  - number: 9092
    name: tcp-plain
    protocol: TCP
  - number: 9093
    name: tcp-tls
    protocol: TCP
  - number: 9094
    name: tcp-sasl-tls
    protocol: TCP
  - number: 9095
    name: tcp-sasl-tls
    protocol: TCP
  - number: 9096
    name: tcp-tls
    protocol: TCP
 location: MESH_EXTERNAL
 resolution: NONE
```

1 Apache Kafka 클러스터 호스트 목록(예: my-cluster-kafka-bootstrap.kafka).

2 Apache Kafka 클러스터 리스너 포트.

참고

spec.ports 에서 나열된 포트는 예제 Cryostat 포트입니다. 실제 값은 Apache Kafka 클러스터 구성 방법에 따라 다릅니다.

8. ServiceEntry 리소스를 적용합니다.

```
$ oc apply -f kafka-cluster-serviceentry.yaml
```

1.4.4. 통합 확인

Istio가 활성화된 Service Mesh 및 Serverless를 설치한 후 통합이 작동하는지 확인할 수 있습니다.

프로세스

1. 사이드카 삽입이 활성화되고 패스쓰루(pass-through) 경로를 사용하는 Knative 서비스를 생성합니다.

knative-service.yaml 구성 파일의 예

```
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>
```

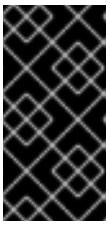
1

2

생성한 인증서가 수신 게이트웨이를 통해 직접 제공되도록 Knative Serving에 패스스루 지원 경로를 생성하도록 지시합니다.

3

서비스 메시 사이드카를 Knative 서비스 Pod에 삽입합니다.



중요

이 예제의 주석을 모든 **Knative** 서비스에 항상 추가하여 서비스 메시에서 작동하도록 합니다.

2. **Service** 리소스를 적용합니다.

```
$ oc apply -f knative-service.yaml
```

3. **CA**에서 신뢰하는 보안 연결을 사용하여 서버리스 애플리케이션에 액세스합니다.

```
$ curl --cacert root.crt <service_url>
```

예를 들어 다음을 실행합니다.

명령 예

```
$ curl --cacert root.crt https://hello-default.apps.openshift.example.com
```

출력 예

```
Hello Openshift!
```

1.5. MTLS와 함께 서비스 메시를 사용할 때 KNATIVE SERVING 및 KNATIVE EVENTING 메트릭 활성화

mTLS(mutual Transport Layer Security)를 사용하여 서비스 메시를 활성화하면 서비스 메시가 **Prometheus**가 메트릭을 스크랩하지 못하기 때문에 **Knative Serving** 및 **Knative Eventing**에 대한 메트릭이 기본적으로 비활성화되어 있습니다. **Service Mesh** 및 **mTLS**를 사용할 때 **Knative Serving** 및 **Knative Eventing** 메트릭을 활성화할 수 있습니다.

사전 요구 사항

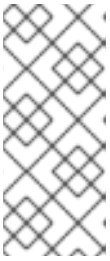
- 클러스터에 액세스할 수 있는 다음 권한 중 하나가 있습니다.
  - OpenShift Container Platform에 대한 클러스터 관리자 권한
  - AWS의 Red Hat OpenShift Service에 대한 클러스터 관리자 권한
  - OpenShift Dedicated에 대한 전용 관리자 권한
- OpenShift CLI(oc)가 설치되어 있습니다.
- 애플리케이션 및 기타 워크로드를 생성할 수 있는 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 클러스터에 OpenShift Serverless Operator, Knative Serving 및 Knative Eventing을 설치했습니다.
- mTLS 기능이 활성화된 Red Hat OpenShift Service Mesh를 설치했습니다.

프로세스

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
  namespace: knative-serving
spec:
  config:
    observability:
      metrics.backend-destination: "prometheus"
...
```

이 단계에서는 메트릭이 기본적으로 비활성화되지 않습니다.



참고

manageNetworkPolicy: false 를 사용하여 **ServiceMeshControlPlane** 을 구성할 때 **KnativeEventing**에서 주석을 사용하여 적절한 이벤트 전달을 보장해야 합니다.

동일한 메커니즘이 **Knative Eventing**에 사용됩니다. **Knative Eventing**에 대한 메트릭을 활성화하려면 다음과 같이 **Knative Eventing CR**(사용자 정의 리소스)의 관찰 가능성 사양에 **prometheus** 를 **metrics.backend-destination** 으로 지정해야 합니다.

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  config:
    observability:
      metrics.backend-destination: "prometheus"
...
```

2. 다음 사양을 포함하도록 **istio-system** 네임스페이스에서 기본 서비스 메시 컨트롤 플레인을 수정하고 다시 적용합니다.

```
...
spec:
  proxy:
    networking:
      trafficControl:
        inbound:
          excludedPorts:
          - 8444
...
```

1.6. 기본 네트워크 정책 비활성화

OpenShift Serverless Operator는 기본적으로 네트워크 정책을 생성합니다. 기본 네트워크 정책 생성을 비활성화하려면 **KnativeEventing** 및 **KnativeServing CR**(사용자 정의 리소스)에 서버리스.**openshift.io/disable-istio-net-policies-generation** 주석을 추가할 수 있습니다.

사전 요구 사항

- 클러스터에 액세스할 수 있는 다음 권한 중 하나가 있습니다.
  - **OpenShift Container Platform**에 대한 클러스터 관리자 권한
  - **AWS**의 **Red Hat OpenShift Service**에 대한 클러스터 관리자 권한
  - **OpenShift Dedicated**에 대한 전용 관리자 권한
- **OpenShift CLI(oc)**가 설치되어 있습니다.
- 애플리케이션 및 기타 워크로드를 생성할 수 있는 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 클러스터에 **OpenShift Serverless Operator, Knative Serving** 및 **Knative Eventing**을 설치했습니다.
- **mTLS** 기능이 활성화된 **Red Hat OpenShift Service Mesh**를 설치했습니다.

프로세스

- 서버리스.**openshift.io/disable-istio-net-policies-generation: "true"** 주석을 **Knative** 사용자 정의 리소스에 추가합니다.

참고

OpenShift Serverless Operator는 기본적으로 필요한 네트워크 정책을 생성합니다. **manageNetworkPolicy: false** 를 사용하여 **ServiceMeshControlPlane**을 구성할 때 적절한 이벤트 전달을 보장하기 위해 기본 네트워크 정책 생성을 비활성화해야 합니다. 기본 네트워크 정책 생성을 비활성화하려면 **KnativeEventing** 및 **KnativeServing CR**(사용자 정의 리소스)에 서버리스.**openshift.io/disable-istio-net-policies-generation** 주석을 추가할 수 있습니다.

a. 다음 명령을 실행하여 **KnativeEventing CR**에 주석을 답니다.

```
$ oc edit KnativeEventing -n knative-eventing
```

KnativeEventing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
  annotations:
    serverless.openshift.io/disable-istio-net-policies-generation: "true"
```

b. 다음 명령을 실행하여 KnativeServing CR에 주석을 답니다.

```
$ oc edit KnativeServing -n knative-serving
```

KnativeServing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
  annotations:
    serverless.openshift.io/disable-istio-net-policies-generation: "true"
```

1.7. SERVICE MESH에 대한 시크릿 필터링을 사용하여 NET-ISTIO 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-istio Ingress 컨트롤러가 실패할 수 있습니다. 그러나 필터링 메커니즘은 Knative net-istio 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-istio** 컨트롤러 포드에 추가됩니다.

중요

보안 필터링을 활성화하는 경우 **networking.internal.knative.dev/certificate-uid: "<id>"** 로 모든 보안에 레이블을 지정해야 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.
- 프로젝트를 생성하거나 애플리케이션 및 기타 워크로드를 생성할 수 있는 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- **Red Hat OpenShift Service Mesh**를 설치합니다. **OpenShift Serverless with Service Mesh**는 **Red Hat OpenShift Service Mesh** 버전 **2.0.5** 이상에서만 지원됩니다.
- **OpenShift Serverless Operator** 및 **Knative Serving**을 설치합니다.
- **OpenShift CLI(oc)**를 설치합니다.

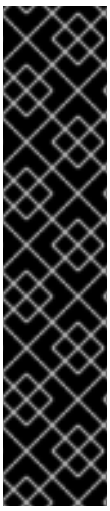
KnativeServing CR(사용자 정의 리소스)의 **workloads** 필드를 사용하여 **ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID** 변수를 **false** 로 설정하여 시크릿 필터링을 비활성화할 수 있습니다.

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-istio-controller
```

2장. SERVICE MESH를 사용하여 OPENSHIFT SERVERLESS에서 네트워크 트래픽 분리



중요

Service Mesh를 사용하여 **OpenShift Serverless**에서 네트워크 트래픽을 분리하는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

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

Service Mesh는 **Service Mesh AuthorizationPolicy** 리소스를 사용하여 공유 **Red Hat OpenShift Serverless** 클러스터에서 테넌트 간에 네트워크 트래픽을 분리하는 데 사용할 수 있습니다. 서버리스는 여러 **Service Mesh** 리소스를 사용하여 이를 활용할 수도 있습니다. 테넌트는 공유 클러스터의 네트워크를 통해 서로 액세스할 수 있는 하나 이상의 프로젝트 그룹입니다.

2.1. 사전 요구 사항

- 클러스터 관리자 액세스 권한이 있는 **Red Hat OpenShift Serverless** 계정에 액세스할 수 있습니다.
- **Service Mesh** 및 **Serverless** 통합을 설정했습니다.
- 각 테넌트에 대해 하나 이상의 **OpenShift** 프로젝트를 생성했습니다.

2.2. 고급 아키텍처

Service Mesh에서 제공하는 **Serverless** 트래픽 격리의 상위 수준 아키텍처는 **knative-serving,knative-eventing** 및 테넌트의 네임스페이스의 **AuthorizationPolicy** 오브젝트로 구성되며 모든 구성 요소는 **Service Mesh**의 일부입니다. 삽입된 서비스 메시 사이드카는 해당 규칙을 적용하여 테넌트 간에 네트워크 트래픽을 분리합니다.

2.3. 서비스 메시 보안

권한 부여 정책 및 **mTLS**를 사용하면 서비스 메시를 보호할 수 있습니다.

프로세스

1. 테넌트의 모든 **Red Hat OpenShift Serverless** 프로젝트가 멤버와 동일한 **ServiceMeshMemberRoll** 오브젝트에 포함되어 있는지 확인합니다.

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
 name: default
 namespace: istio-system
spec:
 members:
   - knative-serving    # static value, needs to be here, see setup page
   - knative-eventing   # static value, needs to be here, see setup page
   - team-alpha-1       # example OpenShift project that belongs to the team-alpha tenant
   - team-alpha-2       # example OpenShift project that belongs th the team-alpha tenant
   - team-bravo-1       # example OpenShift project that belongs to the team-bravo
tenant
   - team-bravo-2       # example OpenShift project that belongs th the team-bravo
tenant
```

메시에 속하는 모든 프로젝트는 **strict** 모드에서 **mTLS**를 적용해야 합니다. 이렇게 하면 **Istio**가 클라이언트-인증서가 있는 연결만 수락하고 **Service Mesh** 사이드카가 **AuthorizationPolicy** 오브젝트를 사용하여 원본을 검증할 수 있습니다.

2. **knative-serving** 및 **knative-eventing** 네임스페이스에서 **AuthorizationPolicy** 오브젝트를 사용하여 구성을 생성합니다.

knative-default-authz-policies.yaml 구성 파일의 예

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all-by-default
  namespace: knative-eventing
spec: { }
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all-by-default
  namespace: knative-serving
spec: { }
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
```

```
  name: allow-mt-channel-based-broker-ingress-to-imc-dispatcher
  namespace: knative-eventing
spec:
  action: ALLOW
  selector:
    matchLabels:
      app.kubernetes.io/component: "imc-dispatcher"
  rules:
    - from:
        - source:
            namespaces: [ "knative-eventing" ]
            principals: [ "cluster.local/ns/knative-eventing/sa/mt-broker-ingress" ]
      to:
        - operation:
            methods: [ "POST" ]
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-mt-channel-based-broker-ingress-to-kafka-channel
  namespace: knative-eventing
spec:
  action: ALLOW
  selector:
    matchLabels:
      app.kubernetes.io/component: "kafka-channel-receiver"
  rules:
    - from:
        - source:
            namespaces: [ "knative-eventing" ]
            principals: [ "cluster.local/ns/knative-eventing/sa/mt-broker-ingress" ]
      to:
        - operation:
            methods: [ "POST" ]
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-kafka-channel-to-mt-channel-based-broker-filter
  namespace: knative-eventing
spec:
  action: ALLOW
  selector:
    matchLabels:
      app.kubernetes.io/component: "broker-filter"
  rules:
    - from:
        - source:
            namespaces: [ "knative-eventing" ]
            principals: [ "cluster.local/ns/knative-eventing/sa/knative-kafka-channel-data-
plane" ]
      to:
        - operation:
            methods: [ "POST" ]
---
apiVersion: security.istio.io/v1beta1
```

```
kind: AuthorizationPolicy
metadata:
  name: allow-imc-to-mt-channel-based-broker-filter
  namespace: knative-eventing
spec:
  action: ALLOW
  selector:
    matchLabels:
      app.kubernetes.io/component: "broker-filter"
  rules:
    - from:
        - source:
            namespaces: [ "knative-eventing" ]
            principals: [ "cluster.local/ns/knative-eventing/sa/imc-dispatcher" ]
      to:
        - operation:
            methods: [ "POST" ]
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-probe-kafka-broker-receiver
  namespace: knative-eventing
spec:
  action: ALLOW
  selector:
    matchLabels:
      app.kubernetes.io/component: "kafka-broker-receiver"
  rules:
    - from:
        - source:
            namespaces: [ "knative-eventing" ]
            principals: [ "cluster.local/ns/knative-eventing/sa/kafka-controller" ]
      to:
        - operation:
            methods: [ "GET" ]
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-probe-kafka-sink-receiver
  namespace: knative-eventing
spec:
  action: ALLOW
  selector:
    matchLabels:
      app.kubernetes.io/component: "kafka-sink-receiver"
  rules:
    - from:
        - source:
            namespaces: [ "knative-eventing" ]
            principals: [ "cluster.local/ns/knative-eventing/sa/kafka-controller" ]
      to:
        - operation:
            methods: [ "GET" ]
---
```

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-probe-kafka-channel-receiver
  namespace: knative-eventing
spec:
  action: ALLOW
  selector:
    matchLabels:
      app.kubernetes.io/component: "kafka-channel-receiver"
  rules:
    - from:
        - source:
            namespaces: [ "knative-eventing" ]
            principals: [ "cluster.local/ns/knative-eventing/sa/kafka-controller" ]
      to:
        - operation:
            methods: [ "GET" ]
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-traffic-to-activator
  namespace: knative-serving
spec:
  selector:
    matchLabels:
      app: activator
  action: ALLOW
  rules:
    - from:
        - source:
            namespaces: [ "knative-serving", "istio-system" ]
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-traffic-to-autoscaler
  namespace: knative-serving
spec:
  selector:
    matchLabels:
      app: autoscaler
  action: ALLOW
  rules:
    - from:
        - source:
            namespaces: [ "knative-serving" ]
```

이러한 정책은 **Serverless** 시스템 구성 요소 간 네트워크 통신에 대한 액세스 규칙을 제한합니다. 특히 다음 규칙을 적용합니다.

- knative-serving 및 knative-eventing 네임스페이스에서 명시적으로 허용되지 않은 모든 트래픽을 거부합니다.
- istio-system 및 knative-serving 네임스페이스의 트래픽을 활성화하도록 허용
- knative-serving 네임스페이스의 트래픽을 자동 스케일러로 허용
- knative-eventing 네임스페이스에서 Apache Kafka 구성 요소에 대한 상태 프로브 허용
- knative-eventing 네임스페이스에서 채널 기반 브로커에 대한 내부 트래픽 허용

3. 권한 부여 정책 구성을 적용합니다.

```
$ oc apply -f knative-default-authz-policies.yaml
```

4. 서로 통신할 수 있는 OpenShift 프로젝트를 정의합니다. 이 통신을 위해 테넌트의 모든 OpenShift 프로젝트에는 다음이 필요합니다.

- 테넌트의 프로젝트로 들어오는 트래픽을 직접 제한하는 하나의 AuthorizationPolicy 오브젝트
- knative-serving 프로젝트에서 실행되는 Serverless의 활성화 구성 요소를 사용하여 들어오는 트래픽을 제한하는 AuthorizationPolicy 오브젝트
- Kubernetes가 Knative 서비스에서 PreStopHooks 를 호출할 수 있도록 하는 하나의 AuthorizationPolicy 오브젝트

이러한 정책을 수동으로 생성하는 대신 helm 유틸리티를 설치하고 각 테넌트에 필요한 리소스를 생성합니다.

helm 유틸리티 설치

```
$ helm repo add openshift-helm-charts https://charts.openshift.io/
```

팀 알파에 대한 구성 예 생성

```
$ helm template openshift-helm-charts/redhat-knative-istio-authz --version 1.31.0 --set
"name=team-alpha" --set "namespaces={team-alpha-1,team-alpha-2}" > team-
alpha.yaml
```

팀 **bravo**의 구성 예 생성

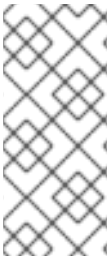
```
$ helm template openshift-helm-charts/redhat-knative-istio-authz --version 1.31.0 --set
"name=team-bravo" --set "namespaces={team-bravo-1,team-bravo-2}" > team-
bravo.yaml
```

5. 권한 부여 정책 구성을 적용합니다.

```
$ oc apply -f team-alpha.yaml team-bravo.yaml
```

2.4. 구성 확인

curl 명령을 사용하여 네트워크 트래픽 격리 구성을 확인할 수 있습니다.



참고

다음 예제에서는 각각 하나의 네임스페이스와 **team-alpha.yaml** 및 **team-bravo.yaml** 파일의 리소스로 구성된 **ServiceMeshMemberRoll** 오브젝트의 두 개의 테넌트가 있다고 가정합니다.

프로세스

1. 두 테넌트의 네임스페이스에 **Knative** 서비스를 배포합니다.

team-alpha의 명령 예

```
$ kn service create test-webapp -n team-alpha-1 \
    --annotation-service serving.knative.openshift.io/enablePassthrough=true \
    --annotation-revision sidecar.istio.io/inject=true \
    --env RESPONSE="Hello Serverless" \
    --image docker.io/openshift/hello-openshift
```

team-bravo의 명령 예

```
$ kn service create test-webapp -n team-bravo-1 \
    --annotation-service serving.knative.openshift.io/enablePassthrough=true \
    --annotation-revision sidecar.istio.io/inject=true \
    --env RESPONSE="Hello Serverless" \
    --image docker.io/openshift/hello-openshift
```

또는 다음 **YAML** 구성을 사용합니다.

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: test-webapp
  namespace: team-alpha-1
  annotations:
    serving.knative.openshift.io/enablePassthrough: "true"
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: 'true'
    spec:
      containers:
        - image: docker.io/openshift/hello-openshift
          env:
```

```
          - name: RESPONSE
            value: "Hello Serverless!"
---
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: test-webapp
  namespace: team-bravo-1
  annotations:
    serving.knative.openshift.io/enablePassthrough: "true"
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: 'true'
    spec:
      containers:
        - image: docker.io/openshift/hello-openshift
          env:
            - name: RESPONSE
              value: "Hello Serverless!"
```

2. 연결을 테스트하기 위해 **curl** 포드를 배포합니다.

```
$ cat <<EOF | oc apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: curl
  namespace: team-alpha-1
  labels:
    app: curl
spec:
  replicas: 1
  selector:
    matchLabels:
      app: curl
  template:
    metadata:
      labels:
        app: curl
      annotations:
        sidecar.istio.io/inject: 'true'
    spec:
      containers:
      - name: curl
        image: curlimages/curl
        command:
        - sleep
        - "3600"
EOF
```

3.

3. **curl** 명령을 사용하여 구성을 확인합니다.

다음을 수행할 수 있는 클러스터 로컬 도메인을 통해 **team-alpha-1** → **team-alpha-1** 을 테스트합니다.

명령 예

```
$ oc exec deployment/curl -n team-alpha-1 -it -- curl -v http://test-webapp.team-alpha-1:80
```

출력 예

```
HTTP/1.1 200 OK
content-length: 18
content-type: text/plain; charset=utf-8
date: Wed, 26 Jul 2023 12:49:59 GMT
server: envoy
x-envoy-upstream-service-time: 9

Hello Serverless!
```

허용되는 외부 도메인을 통해 **team-alpha-1** 을 **team-alpha-1** 연결을 테스트합니다.

명령 예

```
$ EXTERNAL_URL=$(oc get ksvc -n team-alpha-1 test-webapp -o custom-columns=:.status.url --no-headers) && \
oc exec deployment/curl -n team-alpha-1 -it -- curl -ik $EXTERNAL_URL
```

출력 예

```
HTTP/2 200
content-length: 18
content-type: text/plain; charset=utf-8
date: Wed, 26 Jul 2023 12:55:30 GMT
server: istio-envoy
x-envoy-upstream-service-time: 3629

Hello Serverless!
```

클러스터의 로컬 도메인을 통해 **team-alpha-1** 을 **team-bravo-1** 연결을 테스트합니다. 이 연결은 허용되지 않습니다.

명령 예

```
$ oc exec deployment/curl -n team-alpha-1 -it -- curl -v http://test-webapp.team-bravo-1:80
```

출력 예

```
* processing: http://test-webapp.team-bravo-1:80
*   Trying 172.30.73.216:80...
* Connected to test-webapp.team-bravo-1 (172.30.73.216) port 80
> GET / HTTP/1.1
> Host: test-webapp.team-bravo-1
> User-Agent: curl/8.2.0
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< content-length: 19
< content-type: text/plain
< date: Wed, 26 Jul 2023 12:55:49 GMT
< server: envoy
< x-envoy-upstream-service-time: 6
<
* Connection #0 to host test-webapp.team-bravo-1 left intact
RBAC: access denied
```

허용되는 외부 도메인을 통해 **team-alpha-1** 을 **team-bravo-1** 연결을 테스트합니다.

명령 예

```
$ EXTERNAL_URL=$(oc get ksvc -n team-bravo-1 test-webapp -o custom-
columns=:.status.url --no-headers) && \
oc exec deployment/curl -n team-alpha-1 -it -- curl -ik $EXTERNAL_URL
```

출력 예

```
HTTP/2 200
content-length: 18
content-type: text/plain; charset=utf-8
date: Wed, 26 Jul 2023 12:56:22 GMT
server: istio-envoy
x-envoy-upstream-service-time: 2856

Hello Serverless!
```

4. 확인을 위해 생성된 리소스를 삭제합니다.

```
$ oc delete deployment/curl -n team-alpha-1 && \
oc delete ksvc/test-webapp -n team-alpha-1 && \
oc delete ksvc/test-webapp -n team-bravo-1
```

OpenShift Container Platform에 대한 추가 리소스

- Helm 유틸리티

- **Helm 유틸리티의 옵션 참조**

3장. 비용 관리 서비스와 서버리스 통합

비용 관리는 클라우드 및 컨테이너의 비용을 더 잘 이해하고 추적할 수 있는 OpenShift Container Platform 서비스입니다. 이는 오픈 소스 Koku 프로젝트를 기반으로 합니다.

3.1. 사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.
- 비용 관리를 설정하고 OpenShift Container Platform 소스 를 추가했습니다.

3.2. 비용 관리 쿼리에 레이블 사용

비용 관리의 *태그* 라고도 하는 레이블은 노드, 네임스페이스 또는 Pod에 적용할 수 있습니다. 각 레이블은 키 및 값 쌍입니다. 여러 레이블의 조합을 사용하여 보고서를 생성할 수 있습니다. Red Hat 하이브리드 콘솔 을 사용하여 비용에 대한 보고서에 액세스할 수 있습니다.

레이블은 노드에서 네임스페이스로, 네임스페이스에서 Pod로 상속됩니다. 그러나 레이블은 리소스에 이미 있는 경우 재정의되지 않습니다. 예를 들어 Knative 서비스에는 기본 app=<revision_name> 라벨이 있습니다.

Knative 서비스 기본 라벨의 예

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: showcase
spec:
...
     labels:
       app: <revision_name>
...
```

app=my-domain 과 같은 네임스페이스의 레이블을 정의하는 경우 비용 관리 서비스는 app=my-domain 태그를 사용하여 애플리케이션을 쿼리할 때 app=<revision_name > 태그가 있는 Knative 서비

스에서 발생하는 비용을 고려하지 않습니다. 이 태그가 있는 Knative 서비스의 비용을 app=<revision_name> 태그에서 쿼리해야 합니다.

3.3. 추가 리소스

- 소스에 대한 태그 구성
- Cost Explorer를 사용하여 비용을 시각화하십시오.

4장. OPENSHIFT PIPELINES와 SERVERLESS 통합

OpenShift Pipelines와 Serverless를 통합하면 Serverless 서비스에 대한 CI/CD 파이프라인 관리가 가능합니다. 이 통합을 사용하면 Serverless 서비스의 배포를 자동화할 수 있습니다.

4.1. 사전 요구 사항

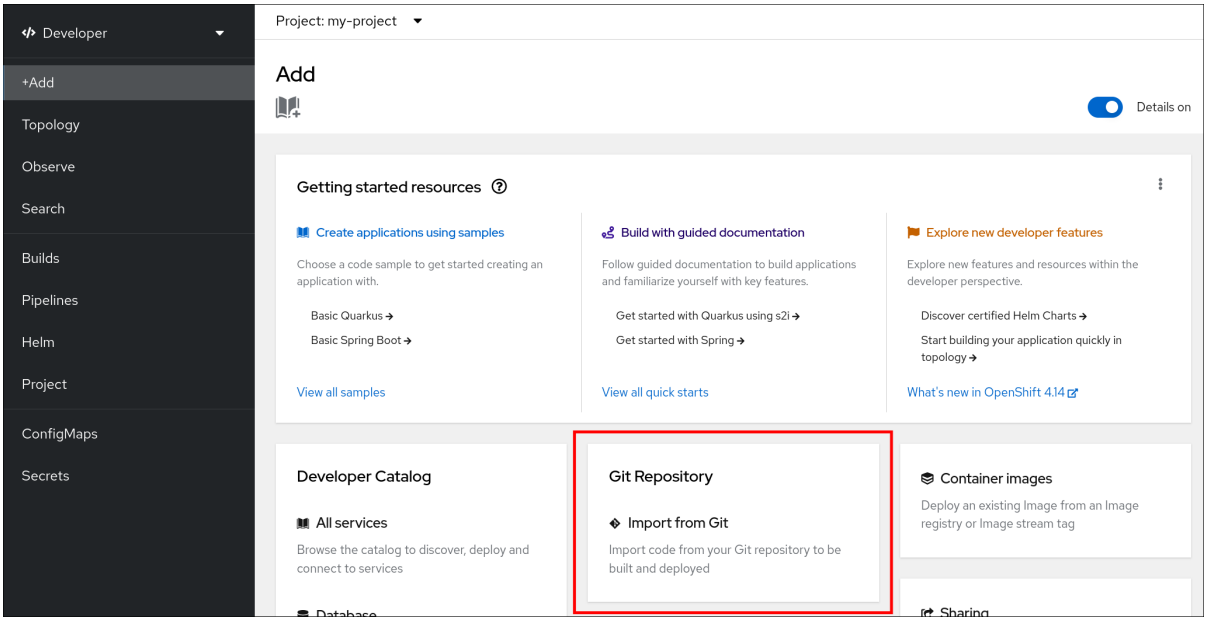
- cluster-admin 권한이 있는 클러스터에 액세스할 수 있습니다.
- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- 클러스터에 OpenShift Pipelines Operator를 설치했습니다.

4.2. OPENSHIFT PIPELINES에서 배포한 서비스 생성

OpenShift Container Platform 웹 콘솔을 사용하여 OpenShift Pipelines에서 배포하는 서비스를 생성할 수 있습니다.

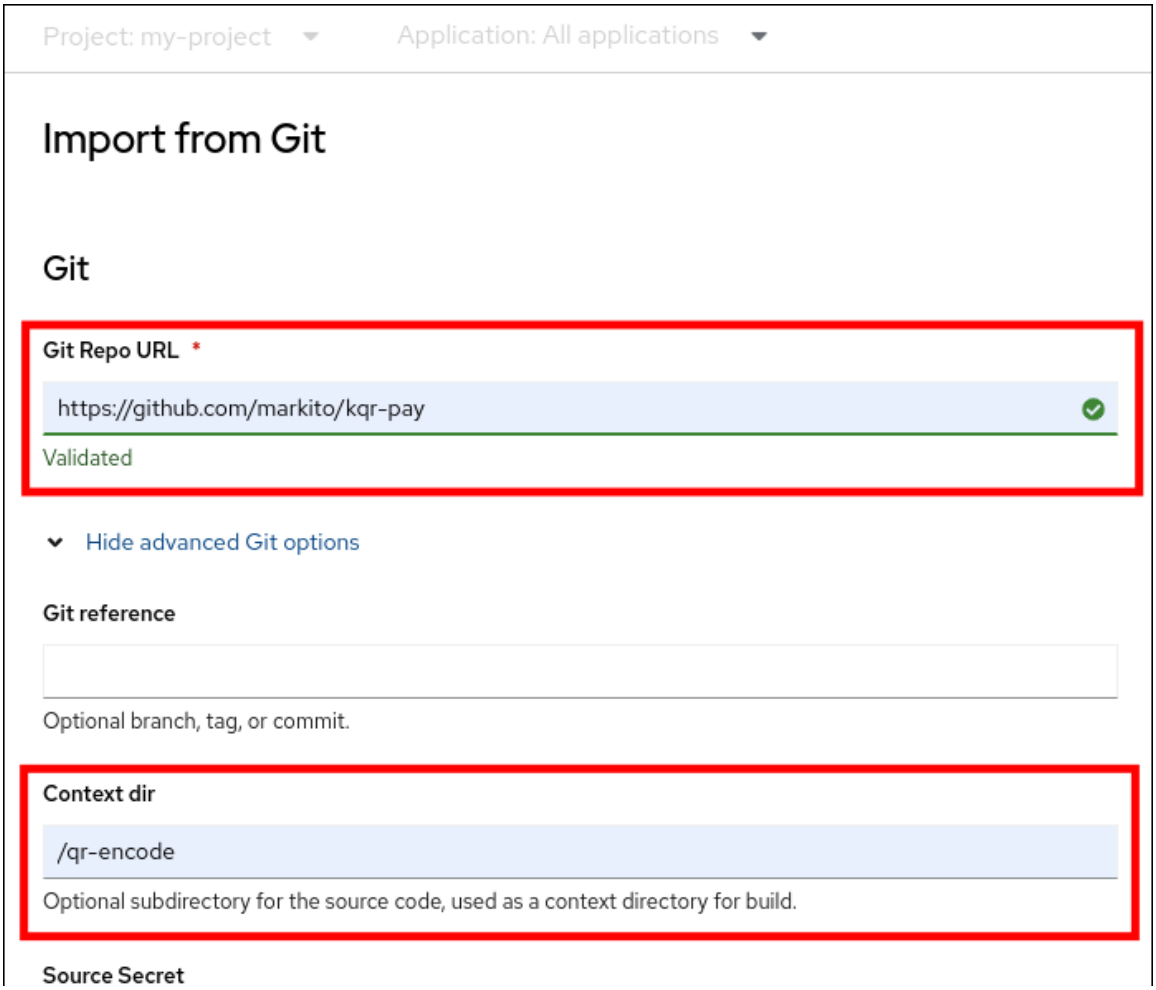
프로세스

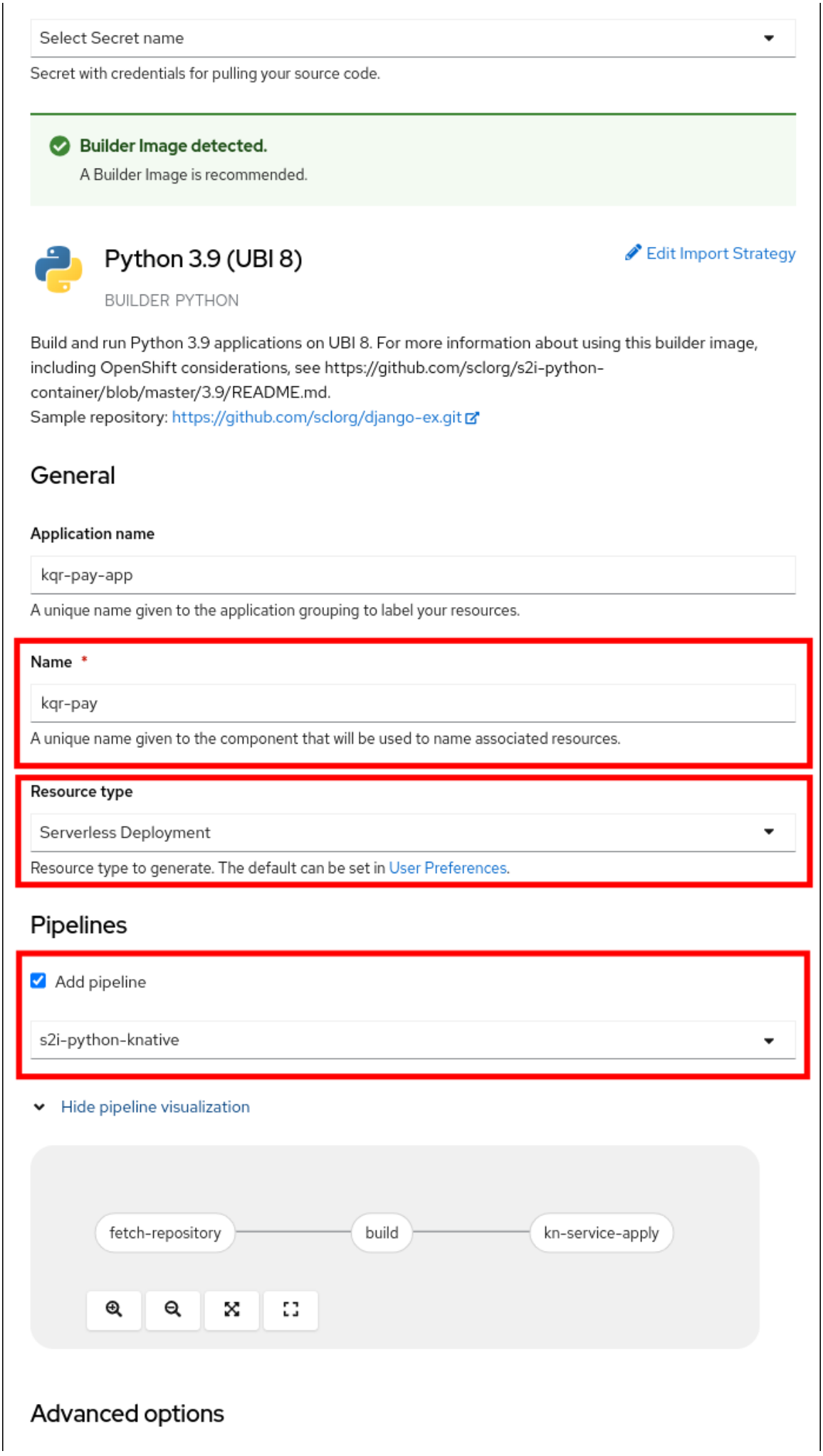
1. OpenShift Container Platform 웹 콘솔 개발자 화면에서 +추가 로 이동하여 Git에서 가져오기 옵션을 선택합니다.

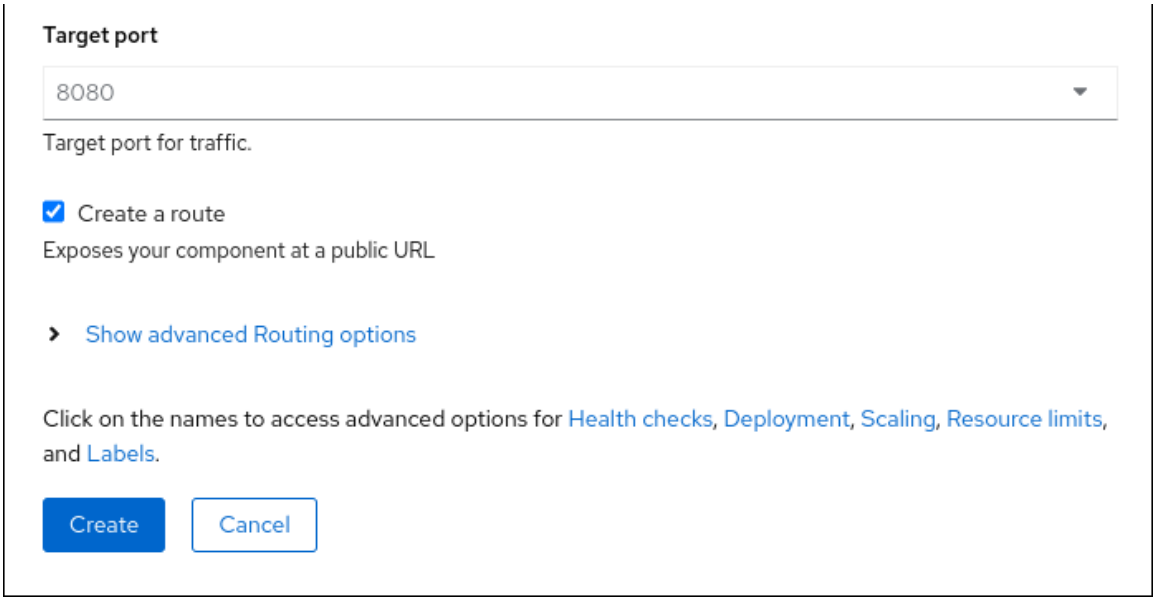


2. **Git에서 가져오기** 대화 상자에서 다음을 수행하여 프로젝트 메타데이터를 지정합니다.

- **Git 리포지토리 URL**을 지정합니다.
- 필요한 경우 컨텍스트 디렉터리를 지정합니다. 이는 애플리케이션 소스 코드의 루트가 포함된 리포지토리 내부의 하위 디렉터리입니다.
- 선택 사항: 애플리케이션 이름을 지정합니다. 기본적으로 리포지토리 이름이 사용됩니다.
- **Serverless Deployment** 리소스 유형을 선택합니다.
- 파이프라인 추가 확인란을 선택합니다. 파이프라인은 소스 코드를 기반으로 자동으로 선택되며 해당 시각화는 스키마에 표시됩니다.
- 기타 관련 설정을 지정합니다.

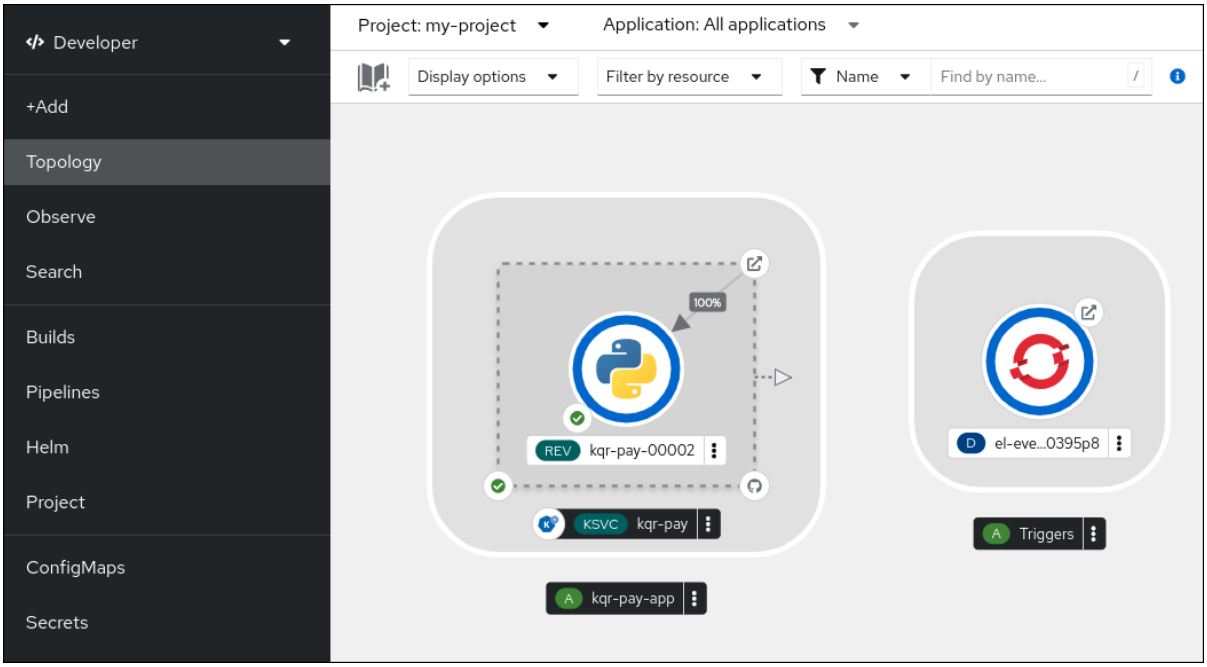



Select Secret name
Secret with credentials for pulling your source code.
Builder Image detected.
A Builder Image is recommended.
Python 3.9 (UBI 8)
BUILDER PYTHON
Edit Import Strategy
Build and run Python 3.9 applications on UBI 8. For more information about using this builder image, including OpenShift considerations, see https://github.com/sclorg/s2i-python-container/blob/master/3.9/README.md.
Sample repository: https://github.com/sclorg/django-ex.git
General
Application name
kqr-pay-app
A unique name given to the application grouping to label your resources.
Name *
kqr-pay
A unique name given to the component that will be used to name associated resources.
Resource type
Serverless Deployment
Resource type to generate. The default can be set in User Preferences.
Pipelines
Add pipeline
s2i-python-knative
Hide pipeline visualization
fetch-repository
build
kn-service-apply
Advanced options

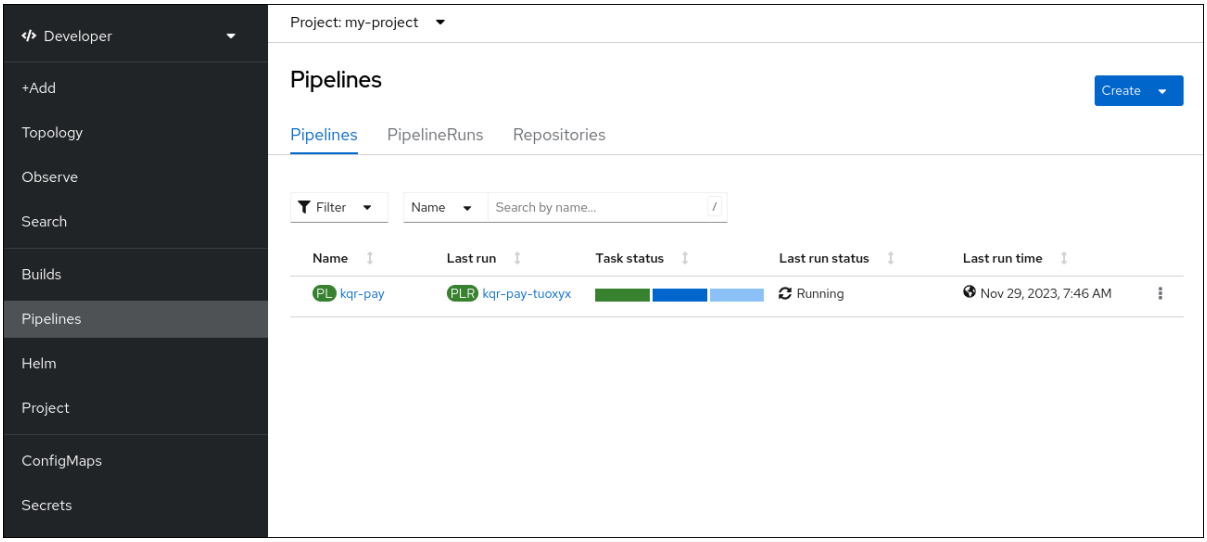


3. **생성 을 클릭하여 서비스를 생성합니다.**

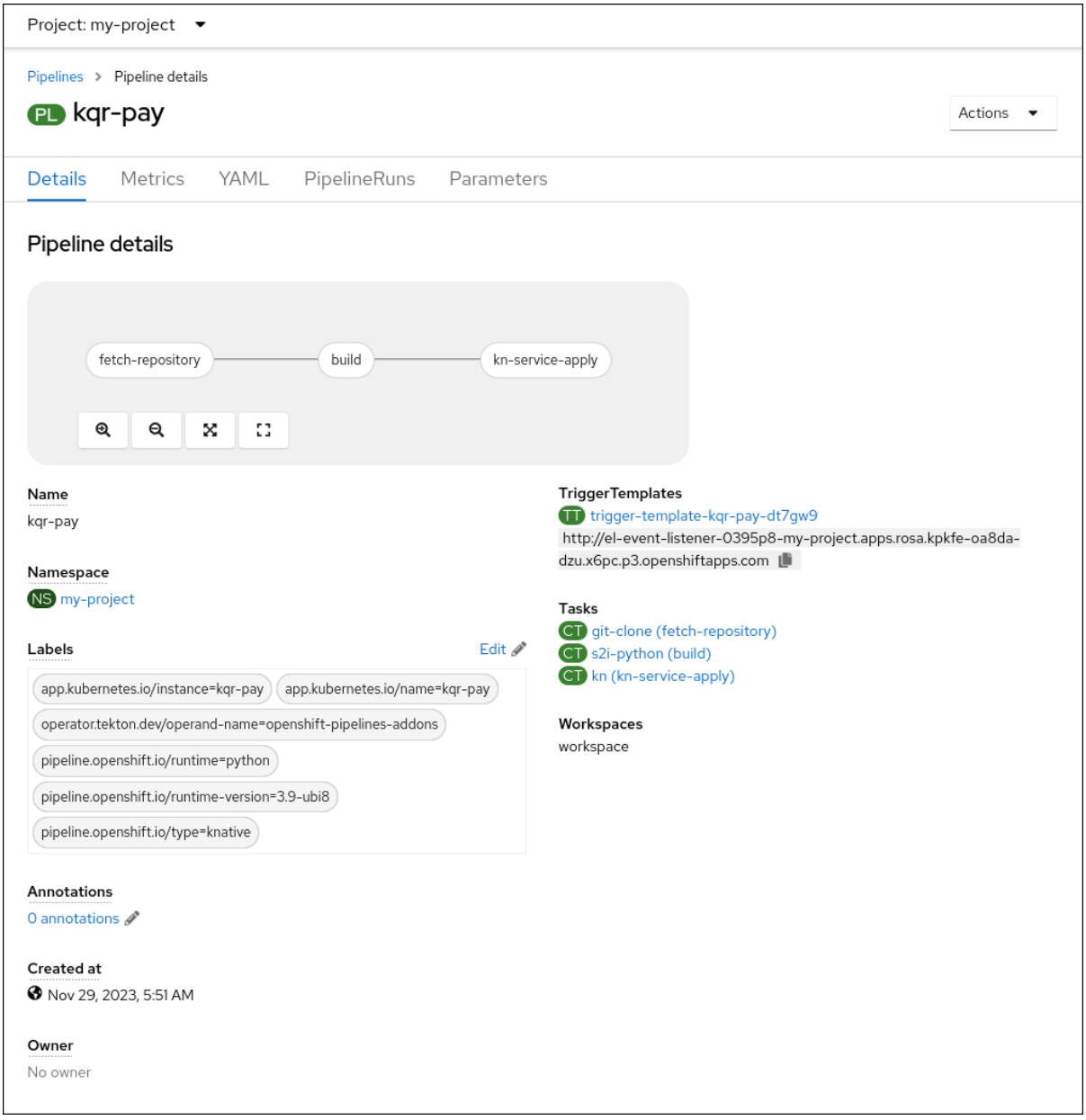
4. **서비스 생성이 시작되면 토폴로지 화면으로 이동합니다. 여기서 서비스와 관련 트리거가 시각화되고 해당 트리거와 상호 작용할 수 있습니다.**



5. **선택 사항: 파이프라인이 생성되고 파이프라인 페이지로 이동하여 서비스가 빌드되고 배포되었는지 확인합니다.**

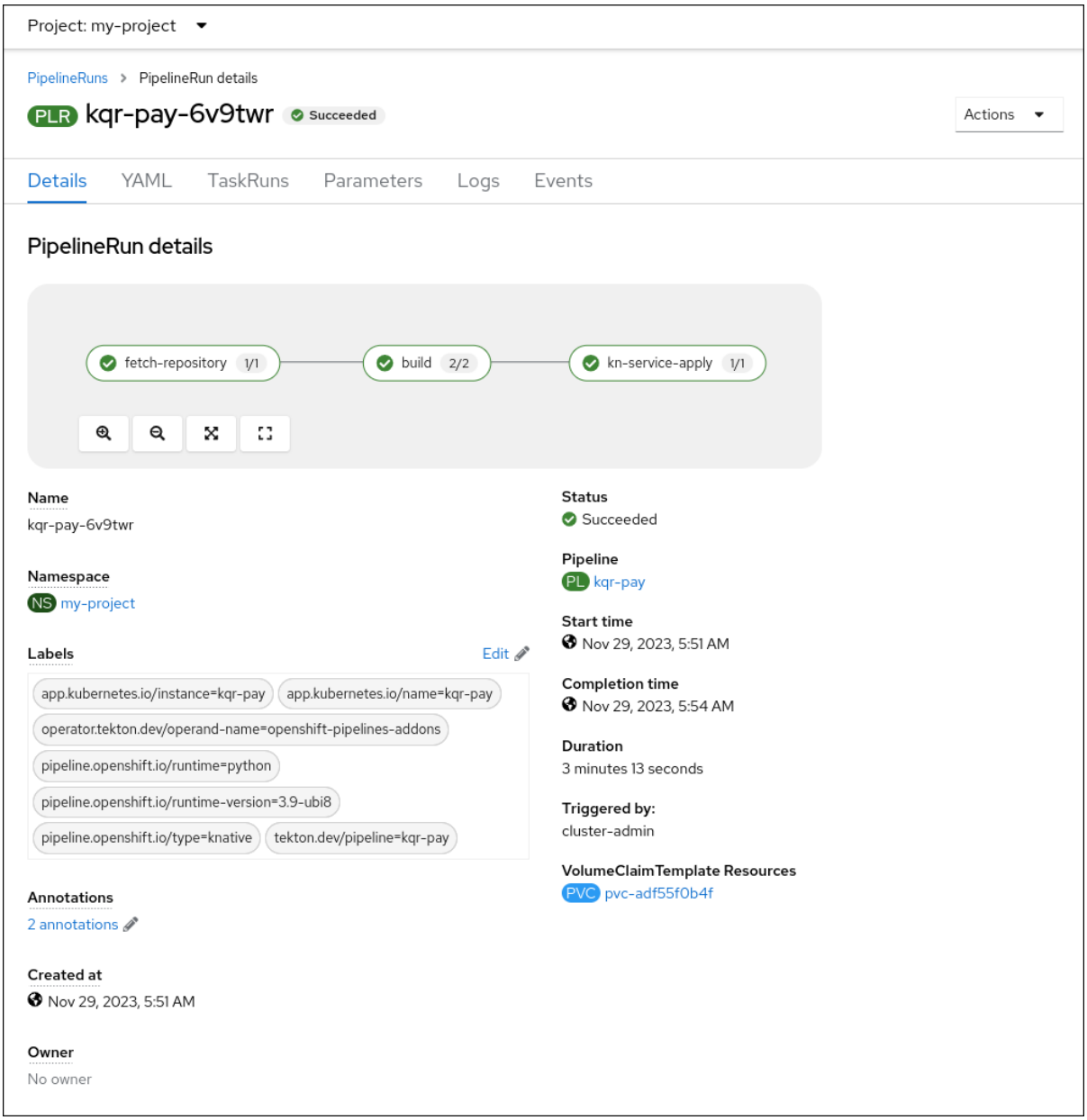


6. 파이프라인의 세부 정보를 보려면 파이프라인 페이지에서 파이프라인을 클릭합니다.



7. 현재 파이프라인 실행에 대한 세부 정보를 보려면 파이프라인 페이지에서 실행 이름을 클릭합

니다.



4.3. 추가 리소스

- **Red Hat OpenShift Pipelines 설명서**

5장. 서버리스 애플리케이션과 함께 NVIDIA GPU 리소스 사용

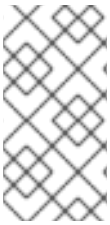
NVIDIA는 OpenShift Container Platform에서 GPU 리소스 사용을 지원합니다. OpenShift Container Platform에서 GPU 리소스를 설정하는 방법에 대한 자세한 내용은 OpenShift의 GPU Operator 를 참조하십시오.

5.1. 서비스에 대한 GPU 요구 사항 지정

OpenShift Container Platform 클러스터에 GPU 리소스가 활성화된 후 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

OpenShift Container Platform 또는 OpenShift Dedicated의 IBM zSystems 및 IBM Power에서는 NVIDIA GPU 리소스를 사용할 수 없습니다.

프로세스

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 리소스

를 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
```

5.2. OPENSHIFT CONTAINER PLATFORM에 대한 추가 리소스

- 확장 리소스에 대한 리소스 할당량 설정