Red Hat OpenShift Serverless 1.30

통합

서비스 메시와 OpenShift Serverless 통합 및 비용 관리 서비스

Last Updated: 2023-09-20

Red Hat OpenShift Serverless 1.30 통합

서비스 메시와 OpenShift Serverless 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.

중요

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

1.1. 사전 요구 사항

- 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.

1.2. 수신 외부 트래픽을 암호화하기 위한 인증서 생성

기본적으로 Service Mesh mTLS 기능은 사이드카가 있는 수신 게이트웨이와 개별 Pod 간에 Service Mesh 자체의 트래픽만 보호합니다. OpenShift Container Platform 클러스터로 전달될 때 트래픽을 암호화하려면 OpenShift Serverless 및 Service Mesh 통합을 활성화하기 전에 인증서를 생성해야 합니다.

사전 요구 사항

- 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
```

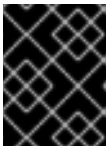
이 인증서는 OpenShift Serverless를 Service Mesh와 통합할 때 생성된 게이트웨이에서 선택하여 수신 게이트웨이가 이 인증서와 트래픽을 제공하도록 합니다.

1.3. OPENSHIFT SERVERLESS와 SERVICE MESH 통합

Kourier를 기본 수신으로 사용하지 않고 OpenShift Serverless와 Service Mesh를 통합할 수 있습니다. 이렇게 하려면 다음 절차를 완료하기 전에 Knative Serving 구성 요소를 설치하지 마십시오. 일반 Knative Serving 설치 절차에서는 **KnativeServing** CRD(사용자 정의 리소스 정의)를 생성하여 Knative Serving을 Service Mesh와 통합할 때 추가 단계가 필요합니다. 이 절차는 Service Mesh를 OpenShift Serverless 설치의 기본 및 수신으로 통합하려는 경우 유용할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- 프로젝트를 생성하거나 애플리케이션 및 기타 워크로드를 생성할 수 있는 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Red Hat OpenShift Service Mesh Operator를 설치하고 **istio-system** 네임스페이스에 **ServiceMeshControlPlane** 리소스를 생성합니다. mTLS 기능을 사용하려면 **ServiceMeshControlPlane** 리소스의 **spec.security.dataPlane.mtls** 필드도 **true** 로 설정해야 합니다.



중요

Service Mesh에서 OpenShift Serverless를 사용하는 것은 Red Hat OpenShift Service Mesh 버전 2.0.5 이상에서만 지원됩니다.

- OpenShift Serverless Operator를 설치합니다.

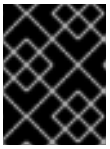
- OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. Service Mesh와 통합할 네임스페이스를 **ServiceMeshMemberRoll** 오브젝트에 멤버로 추가합니다.

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

1 Service Mesh와 통합할 네임스페이스 목록입니다.



중요

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

2. **ServiceMeshMemberRoll** 리소스를 적용합니다.

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

3. Service Mesh가 트래픽을 수락할 수 있도록 필요한 게이트웨이를 생성합니다.

HTTP를 사용하는 knative-local-gateway 오브젝트의 예

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: knative-ingress-gateway
  namespace: knative-serving
spec:
  selector:
    istio: ingressgateway
  servers:
    - port:
        number: 443
        name: https
        protocol: HTTPS
      hosts:
        - "*"
      tls:
        mode: SIMPLE
        credentialName: <wildcard_certs> 1
---
apiVersion: networking.istio.io/v1alpha3
```

```
kind: Gateway
metadata:
 name: knative-local-gateway
 namespace: knative-serving
spec:
 selector:
   istio: ingressgateway
 servers:
   - port:
       number: 8081
       name: http
       protocol: HTTP 2
     hosts:
       - "*"
---
apiVersion: v1
kind: Service
metadata:
 name: knative-local-gateway
 namespace: istio-system
 labels:
   experimental.istio.io/disable-gateway-port-translation: "true"
spec:
 type: ClusterIP
 selector:
   istio: ingressgateway
 ports:
   - name: http2
     port: 80
     targetPort: 8081
```

1 와일드카드 인증서가 포함된 보안의 이름을 추가합니다.

2 **knative-local-gateway**는 HTTP 트래픽을 제공합니다. HTTP를 사용하면 Service Mesh 외부에서 들어오는 트래픽이 표시되지만 **example.default.svc.cluster.local**과 같은 내부 호스트 이름을 사용하는 것은 암호화되지 않습니다. 다른 와일드카드 인증서 및 다른 **protocol** 사양을 사용하는 추가 게이트웨이를 생성하여 이 경로에 대한 암호화를 설정할 수 있습니다.

HTTPS를 사용하는 knative-local-gateway 오브젝트의 예

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: knative-local-gateway
  namespace: knative-serving
spec:
  selector:
    istio: ingressgateway
  servers:
    - port:
        number: 443
        name: https
        protocol: HTTPS
      hosts:
        - "*"
```

```
    tls:
      mode: SIMPLE
      credentialName: <wildcard_certs>
```

4. **Gateway** 리소스를 적용합니다.

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

5. 다음 **KnativeServing** CRD(사용자 정의 리소스 정의)를 생성하여 Knative Serving을 설치하여 Istio 통합을 활성화합니다.

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  ingress:
    istio:
      enabled: true (1)
  deployments: (2)
  - name: activator
    annotations:
      "sidecar.istio.io/inject": "true"
      "sidecar.istio.io/rewriteAppHTTPProbers": "true"
  - name: autoscaler
    annotations:
      "sidecar.istio.io/inject": "true"
      "sidecar.istio.io/rewriteAppHTTPProbers": "true"
```

(1) Istio 통합을 활성화합니다.

(2) Knative Serving 데이터 플레인 Pod에 사이드카 삽입을 활성화합니다.

6. **KnativeServing** 리소스를 적용합니다.

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

7. 다음 **KnativeEventing** CRD(사용자 정의 리소스 정의)를 생성하여 Knative Eventing을 설치하여 Istio 통합을 활성화합니다.

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  config:
    features:
      istio: enabled (1)
  workloads:
  - name: pingsource-mt-adapter
    annotations:
```

```
      "sidecar.istio.io/inject": "true" 2
      "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 컨트롤러에서 각 InMemoryChannel 또는 KafkaChannel 서비스에 대한 **DestinationRule** 을 생성할 수 있습니다.
2. Knative Eventing Pod에 사이드카 삽입을 활성화합니다.

중요

Knative Eventing과 서비스 메시 통합은 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.

1. **KnativeEventing** 리소스를 적용합니다.

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

2. 다음 **KnativeKafka** CRD(사용자 정의 리소스 정의)를 생성하여 Knative Kafka를 설치하여 Istio 통합을 활성화합니다.

```
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.

중요

Knative Eventing과 서비스 메시 통합은 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.

1. **KnativeKafka** 리소스를 적용합니다.

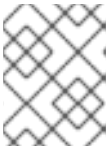
```
$ oc apply -f <filename>
```

2. **ServiceEntry** 를 설치하여 Red Hat OpenShift Service Mesh가 **KnativeKafka** 구성 요소와 Apache Kafka 클러스터 간의 통신을 인식하도록 합니다.

```
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-plain
      protocol: TCP
    - number: 9096
      name: tcp-noauth
      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 에 나열된 포트는 예제 TCP 포트이며 Apache Kafka 클러스터 구성 방법에 따라 달라집니다.

3. **ServiceEntry** 리소스를 적용합니다.

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

중요

주소가 ServiceEntry에 설정되어 있는지 확인합니다. 주소가 설정되지 않은 경우 호스트에 관계없이 ServiceEntry에 정의된 포트의 모든 트래픽이 일치합니다.

검증

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

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: <service_name>
  namespace: <namespace> 1
  annotations:
    serving.knative.openshift.io/enablePassthrough: "true" 2
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true" 3
        sidecar.istio.io/rewriteAppHTTPProbers: "true"
    spec:
      containers:
      - image: <image_url>
```

1. Service Mesh 멤버 롤에 포함된 네임스페이스입니다.
2. 생성한 인증서가 수신 게이트웨이를 통해 직접 제공되도록 Knative Serving에 OpenShift Container Platform 패스스루 지원 경로를 생성하도록 지시합니다.
3. Service Mesh 사이드카를 Knative 서비스 Pod에 삽입합니다.

2. **Service** 리소스를 적용합니다.

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

검증

- CA에서 신뢰하는 보안 연결을 사용하여 서버리스 애플리케이션에 액세스합니다.

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

명령 예

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

출력 예

```
Hello Openshift!
```

1.4. MTLS와 함께 서비스 메시를 사용할 때 KNATIVE SERVING 메트릭 활성화

Service Mesh가 mTLS를 사용하여 활성화된 경우 Service Mesh가 Prometheus 메트릭을 스크랩하지 못하기 때문에 기본적으로 Knative Serving에 대한 메트릭이 비활성화됩니다. 이 섹션에서는 Service Mesh 및 mTLS를 사용할 때 Knative Serving 메트릭을 활성화하는 방법을 보여줍니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving을 클러스터에 설치했습니다.
- mTLS 기능이 활성화된 Red Hat OpenShift Service Mesh를 설치했습니다.
- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 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 Serving 사용자 정의 리소스(CR)의 **observability** 사양에서 **prometheus**를 **metrics.backend-destination**으로 지정합니다.

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

   이 단계에서는 메트릭이 기본적으로 비활성화되지 않습니다.

2. Prometheus 네임스페이스의 트래픽을 허용하려면 다음 네트워크 정책을 적용합니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-monitoring-ns
  namespace: knative-serving
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: "openshift-monitoring"
  podSelector: {}
...
```

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

```
...
spec:
  proxy:
    networking:
      trafficControl:
        inbound:
```

```
        excludedPorts:
        - 8444
...
```

1.5. KOURIER가 활성화된 경우 OPENSHIFT SERVERLESS와 SERVICE MESH 통합

Kourier가 이미 활성화된 경우에도 OpenShift Serverless에서 Service Mesh를 사용할 수 있습니다. 이 절차는 Kourier를 사용하여 Knative Serving을 이미 설치했지만 나중에 Service Mesh 통합을 추가하기로 결정한 경우 유용할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform에 대한 클러스터 관리자 권한이 있거나 AWS 또는 OpenShift Dedicated의 Red Hat OpenShift Service에 대한 클러스터 또는 전용 관리자 권한이 있습니다.
- 프로젝트를 생성하거나 애플리케이션 및 기타 워크로드를 생성할 수 있는 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.
- 클러스터에 OpenShift Serverless Operator 및 Knative Serving을 설치합니다.
- Red Hat OpenShift Service Mesh를 설치합니다. OpenShift Serverless with Service Mesh and Kourier는 Red Hat OpenShift Service Mesh 버전 1.x 및 2.x에서 모두 사용이 지원되고 있습니다.

프로세스

1. Service Mesh와 통합할 네임스페이스를 **ServiceMeshMemberRoll** 오브젝트에 멤버로 추가합니다.

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system
spec:
  members:
    - <namespace> 1
...
```

1 Service Mesh와 통합할 네임스페이스 목록입니다.

2. **ServiceMeshMemberRoll** 리소스를 적용합니다.

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

3. Knative 시스템 Pod에서 Knative 서비스로의 트래픽 흐름을 허용하는 네트워크 정책을 생성합니다.

   a. Service Mesh와 통합할 각 네임스페이스에 **NetworkPolicy** 리소스를 생성합니다.

```
apiVersion: networking.k8s.io/v1
```

```
kind: NetworkPolicy
metadata:
  name: allow-from-serving-system-namespace
  namespace: <namespace> 1
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          knative.openshift.io/part-of: "openshift-serverless"
  podSelector: {}
  policyTypes:
  - Ingress
...
```

1 Service Mesh와 통합할 네임스페이스를 추가합니다.

참고

knative.openshift.io/part-of: "openshift-serverless" 레이블이 OpenShift Serverless 1.22.0에 추가되었습니다. OpenShift Serverless 1.21.1 또는 이전 버전을 사용하는 경우 **knative.openshift.io/part-of** 레이블을 knative-serving 및 **knative-serving -ingress** 네임스페이스에 추가합니다.

knative-serving 네임스페이스에 라벨을 추가합니다.

```
$ oc label namespace knative-serving knative.openshift.io/part-of=openshift-serverless
```

knative-serving-ingress 네임스페이스에 라벨을 추가합니다.

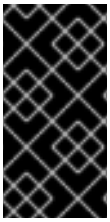
```
$ oc label namespace knative-serving-ingress knative.openshift.io/part-of=openshift-serverless
```

b. **NetworkPolicy** 리소스를 적용합니다.

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

1.6. 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. **KnativeServing** 사용자 정의 리소스(CR)에 주석을 추가하여 이 메커니즘을 활성화할 수 있습니다.



중요

보안 필터링을 활성화하는 경우 모든 시크릿에 **networking.internal.knative.dev/certificate-uid: "<id>"** 로 레이블이 지정되어야 합니다. 그렇지 않으면 Knative Serving이 탐지되지 않아 오류가 발생합니다. 새 보안 및 기존 보안에 레이블을 지정해야 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.
- 프로젝트를 생성하거나 애플리케이션 및 기타 워크로드를 생성할 수 있는 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- Red Hat OpenShift Service Mesh를 설치합니다. OpenShift Serverless with Service Mesh는 Red Hat OpenShift Service Mesh 버전 2.0.5 이상에서만 지원됩니다.
- OpenShift Serverless Operator 및 Knative Serving을 설치합니다.
- OpenShift CLI(**oc**)를 설치합니다.

프로세스

- **서버리스.openshift.io/enable-secret-informer-filtering** 주석을 **KnativeServing** CR에 추가합니다.

  KnativeServing CR의 예

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
  annotations:
    serverless.openshift.io/enable-secret-informer-filtering: "true" (1)
spec:
  ingress:
    istio:
      enabled: true
  deployments:
    - annotations:
        sidecar.istio.io/inject: "true"
        sidecar.istio.io/rewriteAppHTTPProbers: "true"
      name: activator
    - annotations:
        sidecar.istio.io/inject: "true"
        sidecar.istio.io/rewriteAppHTTPProbers: "true"
      name: autoscaler
```

(1) 이 주석을 추가하면 환경 변수 **ENABLE_SECRET_INFORMER_BY_CERT_UID=true** 를 **net-istio** 컨트롤러 포드에 삽입합니다.

참고

배포를 재정의하여 다른 값을 설정하면 이 주석은 무시됩니다.

2장. 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.

2.1. 사전 요구 사항

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

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

2.3. 추가 리소스

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

3장. 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.

3.1. 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 **1**로 설정합니다.

```
$ kn service create hello --image <service-image> --limit nvidia.com/gpu=1
```

   GPU 리소스 요구 사항 제한이 **1**이면 서비스의 전용 GPU 리소스가 1개임을 나타냅니다. 서비스에서는 GPU 리소스를 공유하지 않습니다. GPU 리소스가 필요한 기타 서비스는 GPU 리소스를 더 이상 사용하지 않을 때까지 기다려야 합니다.

   또한 GPU가 1개로 제한되면 GPU 리소스를 2개 이상 사용하는 애플리케이션이 제한됩니다. 서비스에서 GPU 리소스를 1개 이상 요청하는 경우 GPU 리소스 요구 사항을 충족할 수 있는 노드에 배포됩니다.

2. 선택 사항: 기존 서비스의 경우 **--limit nvidia.com/gpu=3** 플래그를 사용하여 GPU 리소스 요구 사항 제한을 **3**으로 변경할 수 있습니다.

```
$ kn service update hello --limit nvidia.com/gpu=3
```

3.2. OPENSHIFT CONTAINER PLATFORM에 대한 추가 리소스

- 확장 리소스에 대한 리소스 할당량 설정