Red Hat OpenShift Serverless 1.30

インテグレーション

OpenShift Serverless と Service Mesh の統合、およびコスト管理サービスとの統合

Last Updated: 2023-09-20

Red Hat OpenShift Serverless 1.30 インテグレーション

OpenShift Serverless と Service Mesh の統合、およびコスト管理サービスとの統合

法律上の通知

概要

本書では、Service Mesh を OpenShift Serverless と統合する方法を説明します。また、コスト管理サービスを使用したコストの理解および追跡や、サーバーレスアプリケーションでの NVIDIA GPU リソースの使用についても説明します。

目次

第1章 サービスメッシュと OPENSHIFT SERVERLESS の統合 3
1.1. 前提条件 3
1.2. 着信外部トラフィックを暗号化する証明書の作成 3
1.3. サービスメッシュと OPENSHIFT SERVERLESS の統合 4
1.4. MTLS で SERVICE MESH を使用する場合の KNATIVE SERVING メトリックの有効化 12
1.5. KOURIER が有効にされている場合のサービスメッシュの OPENSHIFT SERVERLESS との統合 13
1.6. SERVICE MESH のシークレットフィルタリングを使用した NET-ISTIO のメモリー使用量の改善 15

第2章 サーバーレスと COST MANAGEMENT SERVICE の統合 17
2.1. 前提条件 17
2.2. コスト管理クエリーにラベルを使用する 17
2.3. 関連情報 17

第3章 サーバーレスアプリケーションでの NVIDIA GPU リソースの使用 18
3.1. サービスの GPU 要件の指定 18
3.2. OPENSHIFT CONTAINER PLATFORM の関連情報 18

第1章 サービスメッシュと OPENSHIFT SERVERLESS の統合

OpenShift Serverless Operator は、Knative のデフォルト Ingress として Kourier を提供します。ただし、Kourier が有効であるかどうかにかかわらず、OpenShift Serverless でサービスメッシュを使用できます。Kourier を無効にして統合すると、mTLS 機能など、Kourier イングレスがサポートしない追加のネットワークおよびルーティングオプションを設定できます。

重要

OpenShift Serverless は、本書で明示的に文書化されている Red Hat OpenShift Service Mesh 機能の使用のみをサポートし、文書化されていない他の機能はサポートしません。

1.1. 前提条件

- 以下の手順の例では、ドメイン **example.com** を使用します。このドメインの証明書のサンプルは、サブドメイン証明書に署名する認証局 (CA) として使用されます。
  お使いのデプロイメントでこの手順を完了し、検証するには、一般に信頼されているパブリック CA によって署名された証明書、または組織が提供する CA のいずれかが必要です。コマンドの例は、ドメイン、サブドメイン、および CA に合わせて調整する必要があります。

- ワイルドカード証明書を OpenShift Container Platform クラスターのドメインに一致するように設定する必要があります。たとえば、OpenShift Container Platform コンソールアドレスが **https://console-openshift-console.apps.openshift.example.com** の場合は、ドメインが ***.apps.openshift.example.com** になるようにワイルドカード証明書を設定する必要があります。ワイルドカード証明書の設定に関する詳細は、**着信外部トラフィックを暗号化する証明書の作成**のトピックを参照してください。

- デフォルトの OpenShift Container Platform クラスタードメインのサブドメインではないものを含むドメイン名を使用する必要がある場合は、これらのドメインのドメインマッピングを設定する必要があります。詳細は、OpenShift Serverless ドキュメントのカスタムドメインマッピングの作成を参照してください。

1.2. 着信外部トラフィックを暗号化する証明書の作成

デフォルトでは、サービスメッシュ mTLS 機能は、Ingress ゲートウェイとサイドカーを持つ個々の Pod 間で、サービスメッシュ自体内のトラフィックのみを保護します。OpenShift Container Platform クラスターに流入するトラフィックを暗号化するには、OpenShift Serverless とサービスメッシュの統合を有効にする前に証明書を生成する必要があります。

前提条件

- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。

- OpenShift Serverless Operator および Knative Serving がインストールされている。

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

- アプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションが割り当てられたプロジェクトにアクセスできる。

手順

1. Knative サービスの証明書に署名する root 証明書と秘密鍵を作成します。

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

2. ワイルドカード証明書を作成します。

```
$ openssl req -nodes -newkey rsa:2048 \
    -subj "/CN=*.apps.openshift.example.com/O=Example Inc." \
    -keyout wildcard.key \
    -out wildcard.csr
```

3. ワイルドカード証明書を署名します。

```
$ openssl x509 -req -days 365 -set_serial 0 \
    -CA root.crt \
    -CAkey root.key \
    -in wildcard.csr \
    -out wildcard.crt
```

4. ワイルドカード証明書を使用してシークレットを作成します。

```
$ oc create -n istio-system secret tls wildcard-certs \
    --key=wildcard.key \
    --cert=wildcard.crt
```

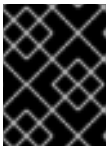
この証明書は、OpenShift Serverless をサービスメッシュと統合する際に作成されるゲートウェイによって取得され、Ingress ゲートウェイはこの証明書でトラフィックを提供します。

1.3. サービスメッシュと OPENSHIFT SERVERLESS の統合

Kourier をデフォルトのイングレスとして使用せずに、Service Mesh を OpenShift Serverless と統合できます。このため、以下の手順を完了する前に、Knative Serving コンポーネントをインストールしないでください。Knative Serving をサービスメッシュと統合するために **KnativeServing** カスタムリソース定義 (CRD) を作成する際に必要な追加の手順があります。これは、一般的な Knative Serving のインストール手順では説明されていません。この手順は、サービスメッシュをデフォルトとして統合し、OpenShift Serverless インストールの唯一のイングレスとして統合する場合に役立ちます。

前提条件

- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。
- アプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションが割り当てられたプロジェクトにアクセスできる。
- Red Hat OpenShift Service Mesh Operator をインストールし、**istio-system** namespace に **ServiceMeshControlPlane** リソースを作成します。mTLS 機能を使用する場合は、**ServiceMeshControlPlane** リソースの **spec.security.dataPlane.mtls** フィールドも **true** に設定する必要があります。



重要

Service Mesh での OpenShift Serverless の使用は、Red Hat OpenShift Service Mesh バージョン 2.0.5 以降でのみサポートされます。

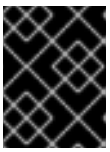
- OpenShift Serverless Operator をインストールしている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. サービスメッシュと統合する必要のある namespace をメンバーとして **ServiceMeshMemberRoll** オブジェクトに追加します。

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

1 サービスメッシュと統合する namespace の一覧。



重要

この namespace のリストには、**knative-serving** namespace と **knative-eventing** namespace が含まれている必要があります。

2. **ServiceMeshMemberRoll** リソースを適用します。

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

3. サービスメッシュがトラフィックを受け入れることができるように、必要なゲートウェイを作成します。

HTTP を使用した knative-local-gateway オブジェクトの例

```
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 ワイルドカード証明書を含むシークレットの名前を追加します。

2 **knative-local-gateway** は HTTP トラフィックに対応します。HTTP を使用するということは、サービスメッシュの外部から来るが、**example.default.svc.cluster.local** などの内部ホスト名を使用するトラフィックが暗号化されていないことを意味します。別のワイルドカード証明書と、異なる **protocol** 仕様を使用する追加のゲートウェイを作成することで、このパスの暗号化を設定できます。

HTTPS を使用した knative-local-gateway オブジェクトの例

```
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** リソースを適用します。

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

5. 以下の **KnativeServing** カスタムリソース定義 (CRD) を作成して Knative Serving をインストールします。これにより、Istio 統合も有効化されます。

```
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 統合を有効にします。

(2) Knative Serving データプレーン Pod のサイドカーの挿入を有効にします。

6. **KnativeServing** リソースを適用します。

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

7. 以下の **KnativeEventing** カスタムリソース定義 (CRD) を作成して Knative Eventing をインストールします。これにより、Istio 統合も有効化されます。

```
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 コントローラーが各 InMemoryChannel または KafkaChannel サービスの **DestinationRule** を作成できるようにします。

2. Knative Eventing Pod のサイドカーインジェクションを有効にします。

重要

Knative Eventing と Service Mesh の統合は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品サービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

1. **KnativeEventing** リソースを適用します。

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

2. 以下の **KnativeKafka** カスタムリソース定義 (CRD) を作成して Knative Kafka をインストールします。これにより、Istio 統合も有効化されます。

```
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 クラスター URL (例: **my-cluster-kafka-bootstrap.kafka:9092**)。

3 Knative Kafka Pod のサイドカーインジェクションを有効にします。

重要

Knative Eventing と Service Mesh の統合は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品サービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

1. **KnativeKafka** リソースを適用します。

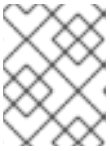
```
$ oc apply -f <filename>
```

2. **ServiceEntry** をインストールして、Red Hat OpenShift Service Mesh が **KnativeKafka** コンポーネントと Apache Kafka クラスター間の通信を認識できるようにします。

```
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 クラスターホストのリスト (例: **my-cluster Kafka -bootstrap.kafka**)。

(2) Apache Kafka クラスターリスナーポート。

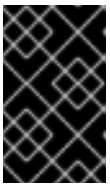


注記

spec.ports にリストされているポートは TCP ポートの例であり、Apache Kafka クラスターの設定方法によって異なります。

3. **ServiceEntry** リソースを適用します。

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



重要

ServiceEntry にアドレスが設定されていることを確認してください。アドレスが設定されていない場合は、ホストに関係なく、ServiceEntry で定義されたポート上のすべてのトラフィックが照合されます。

検証

1. サイドカー挿入が有効で、パススルールートを使用する Knative サービスを作成します。

```
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) サービスメッシュメンバーロールの一部である namespace。

(2) OpenShift Container Platform のパススルーが有効化されたルートを生成するよう Knative Serving に指示します。これにより、生成した証明書は Ingress ゲートウェイ経由で直接提供されます。

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

2. **Service** リソースを適用します。

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

検証

- CA によって信頼されるようになった安全な接続を使用して、サーバーレスアプリケーションにアクセスします。

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

コマンドの例

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

出力例

```
Hello Openshift!
```

1.4. MTLS で SERVICE MESH を使用する場合の KNATIVE SERVING メトリックの有効化

サービスメッシュが mTLS で有効にされている場合は、サービスメッシュが Prometheus のメトリクスの収集を阻止するため、Knative Serving のメトリクスはデフォルトで無効にされます。このセクションでは、Service Mesh および mTLS を使用する際に Knative Serving メトリックを有効にする方法を説明します。

前提条件

- OpenShift Serverless Operator および Knative Serving がクラスターにインストールされている。
- mTLS 機能を有効にして Red Hat OpenShift Service Mesh をインストールしている。
- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。
- OpenShift CLI (**oc**) がインストールされている。
- アプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションが割り当てられたプロジェクトにアクセスできる。

手順

1. **prometheus** を Knative Serving カスタムリソース (CR) の **observability** 仕様で **metrics.backend-destination** として指定します。

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

   この手順により、メトリックがデフォルトで無効になることを防ぎます。

2. 以下のネットワークポリシーを適用して、Prometheus namespace からのトラフィックを許可します。

```
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. **istio-system** namespace のデフォルトのサービスメッシュコントロールプレーンを変更して再適用し、以下の仕様が含まれるようにします。

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

1.5. KOURIER が有効にされている場合のサービスメッシュの OPENSHIFT SERVERLESS との統合

Kourier がすでに有効になっている場合でも、OpenShift Serverless で Service Mesh を使用できます。この手順は、Kourier を有効にして Knative Serving をすでにインストールしているが、後で Service Mesh 統合を追加することにした場合に役立つ可能性があります。

前提条件

- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。
- アプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションが割り当てられたプロジェクトにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift Serverless Operator と Knative Serving をクラスターにインストールしている。
- Red Hat OpenShift Service Mesh をインストールしている。OpenShift Serverless with Service Mesh and Kourier は、Red Hat OpenShift Service Mesh バージョン 1.x および 2.x の両方での使用がサポートされています。

手順

1. サービスメッシュと統合する必要のある namespace をメンバーとして **ServiceMeshMemberRoll** オブジェクトに追加します。

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

1 サービスメッシュと統合する namespace の一覧。

2. **ServiceMeshMemberRoll** リソースを適用します。

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

3. Knative システム Pod から Knative サービスへのトラフィックフローを許可するネットワークポリシーを作成します。

   a. サービスメッシュと統合する必要のある namespace ごとに、**NetworkPolicy** リソースを作成します。

```
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 サービスメッシュと統合する必要のある namespace を追加します。

注記

knative.openshift.io/part-of: "openshift-serverless" ラベルが OpenShift Serverless 1.22.0 で追加されました。OpenShift Serverless 1.21.1 以前を使用している場合は、**knative.openshift.io/part-of** ラベルを **knative-serving** および **knative-serving-ingress** namespace に追加します。

knative-serving namespace にラベルを追加します。

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

knative-serving-ingress namespace にラベルを追加します。

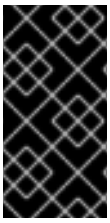
```
$ oc label namespace knative-serving-ingress knative.openshift.io/part-of=openshift-serverless
```

b. **NetworkPolicy** リソースを適用します。

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

1.6. SERVICE MESH のシークレットフィルタリングを使用した NET-ISTIO のメモリー使用量の改善

デフォルトでは、Kubernetes **client-go** ライブラリーの informers の実装は、特定のタイプのすべてのリソースをフェッチします。これにより、多くのリソースが使用可能な場合にかなりのオーバーヘッドが発生する可能性があり、メモリーリークが原因で大規模なクラスターで Knative **net-istio** イングレスコントローラーが失敗する可能性があります。ただし、Knative **net-istio** イングレスコントローラーではフィルタリングメカニズムを使用できます。これにより、コントローラーは Knative 関連のシークレットのみを取得できます。このメカニズムを有効にするには、**KnativeServing** カスタムリソース (CR) にアノテーションを追加します。



重要

シークレットフィルタリングを有効にする場合は、すべてのシークレットに **networking.internal.knative.dev/certificate-uid: "<id>"** というラベルを付ける必要があります。そうしないと、Knative Serving がシークレットを検出しないため、障害が発生します。新規および既存のシークレットの両方にラベルを付ける必要があります。

前提条件

- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。
- アプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションが割り当てられたプロジェクトにアクセスできる。
- Red Hat OpenShift Service Mesh をインストールしている。OpenShift Serverless with Service Mesh は、Red Hat OpenShift Service Mesh バージョン 2.0.5 以降での使用でのみサポートされます。
- OpenShift Serverless Operator および Knative Serving をインストールしている。

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

手順

- **serverless.openshift.io/enable-secret-informer-filtering** アノテーションを **KnativeServing** CR に追加します。

  KnativeServing CR の例

```
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 このアノテーションを追加すると、環境変数 **ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID=true** が **net-istio** コントローラー Pod に挿入されます。

注記

このアノテーションは、デプロイメントを上書きして別の値を設定した場合は無視されます。

第2章 サーバーレスと COST MANAGEMENT SERVICE の統合

Cost Management は OpenShift Container Platform のサービスで、クラウドおよびコンテナーのコストをより正確に把握し、追跡することができます。これは、オープンソースの Koku プロジェクトに基づいています。

2.1. 前提条件

- クラスター管理者パーミッションがある。
- コスト管理を設定し、OpenShift Container Platform source を追加している。

2.2. コスト管理クエリーにラベルを使用する

コスト管理では **タグ** とも呼ばれるラベルは、ノード、namespace、または Pod に適用できます。各ラベルはキーと値のペアです。複数のラベルを組み合わせてレポートを生成できます。Red Hat ハイブリッドコンソール を使用して、コストに関するレポートにアクセスできます。

ラベルは、ノードから namespace に、namespace から Pod に継承されます。ただし、ラベルがリソースにすでに存在する場合、ラベルはオーバーライドされません。たとえば、Knative サービスにはデフォルトの **app=<revision_name>** ラベルがあります。

例 Knative サービスのデフォルトラベル

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

app=my-domain のように namespace のラベルを定義した場合は、**app=my-domain** タグを使用してアプリケーションに問い合わせたときに、**app=<revision_name>** タグの Knative サービスから生じるコストは Cost Management Service では考慮されません。このタグを持つ Knative サービスのコストは、**app=<revision_name>** タグの下で照会する必要があります。

2.3. 関連情報

- ソースへのタグ付けの設定
- Cost Explorer を使用したコストの可視化

第3章 サーバーレスアプリケーションでの NVIDIA GPU リソースの使用

NVIDIA は、OpenShift Container Platform での GPU リソースの使用をサポートしています。OpenShift Container Platform での GPU リソースの設定に関する詳細は、OpenShift の GPU Operator を参照してください。

3.1. サービスの GPU 要件の指定

OpenShift Container Platform クラスターの GPU リソースが有効化された後に、Knative (**kn**) CLI を使用して Knative サービスの GPU 要件を指定できます。

前提条件

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

注記

NVIDIA GPU リソースの使用は、OpenShift Container Platform または OpenShift Dedicated の IBM zSystem および IBM Power ではサポートされていません。

手順

1. Knative サービスを作成し、**--limit nvidia.com/gpu=1** フラグを使用して、GPU リソース要件の制限を **1** に設定します。

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

GPU リソース要件の制限が **1** の場合、サービスには専用の GPU リソースが 1 つ必要です。サービスは、GPU リソースを共有しません。GPU リソースを必要とするその他のサービスは、GPU リソースが使用されなくなるまで待機する必要があります。

1 GPU の制限は、1 GPU リソースの使用を超えるアプリケーションが制限されることも意味します。サービスが 2 つ以上の GPU リソースを要求する場合、これは GPU リソース要件を満たしているノードにデプロイされます。

2. オプション: 既存のサービスの場合は、**--limit nvidia.com/gpu=3** フラグを使用して、GPU リソース要件の制限を **3** に変更できます。

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

3.2. OPENSHIFT CONTAINER PLATFORM の関連情報

- 拡張リソースのリソースクォータの設定