Red Hat OpenShift Serverless 1.30

Integrations

Integrating OpenShift Serverless with Service Mesh, and with the cost management service
Integrating OpenShift Serverless with Service Mesh, and with the cost management service
Abstract

This document provides information on how to integrate Service Mesh with OpenShift Serverless. It also covers using the cost management service to help you understand and track costs, and shows you how to use NVIDIA GPU resources with serverless applications.
Table of Contents

CHAPTER 1. INTEGRATING SERVICE MESH WITH OPENSShift SERVERLESS ........................................ 3
  1.1. PREREQUISITES ......................................................... 3
  1.2. CREATING A CERTIFICATE TO ENCRYPT INCOMING EXTERNAL TRAFFIC ............................ 3
  1.3. INTEGRATING SERVICE MESH WITH OPENSShift SERVERLESS ....................................... 4
  1.4. ENABLING KNATIVE SERVING METRICS WHEN USING SERVICE MESH WITH MTLS ............... 12
  1.5. INTEGRATING SERVICE MESH WITH OPENSShift SERVERLESS WHEN KOURIER IS ENABLED ......... 13
  1.6. IMPROVING NET-ISTIO MEMORY USAGE BY USING SECRET FILTERING FOR SERVICE MESH ....... 15

CHAPTER 2. INTEGRATING SERVERLESS WITH THE COST MANAGEMENT SERVICE .............................. 17
  2.1. PREREQUISITES ......................................................... 17
  2.2. USING LABELS FOR COST MANAGEMENT QUERIES ......................................................... 17
  2.3. ADDITIONAL RESOURCES ........................................... 17

CHAPTER 3. USING NVIDIA GPU RESOURCES WITH SERVERLESS APPLICATIONS ............................. 18
  3.1. SPECIFYING GPU REQUIREMENTS FOR A SERVICE ...................................................... 18
  3.2. ADDITIONAL RESOURCES FOR OPENSShift CONTAINER PLATFORM ................................ 18
CHAPTER 1. INTEGRATING SERVICE MESH WITH OPENSHIFT SERVERLESS

The OpenShift Serverless Operator provides Kourier as the default ingress for Knative. However, you can use Service Mesh with OpenShift Serverless whether Kourier is enabled or not. Integrating with Kourier disabled allows you to configure additional networking and routing options that the Kourier ingress does not support, such as mTLS functionality.

IMPORTANT

OpenShift Serverless only supports the use of Red Hat OpenShift Service Mesh functionality that is explicitly documented in this guide, and does not support other undocumented features.

1.1. PREREQUISITES

- The examples in the following procedures use the domain example.com. The example certificate for this domain is used as a certificate authority (CA) that signs the subdomain certificate.
  To complete and verify these procedures in your deployment, you need either a certificate signed by a widely trusted public CA or a CA provided by your organization. Example commands must be adjusted according to your domain, subdomain, and CA.

- You must configure the wildcard certificate to match the domain of your OpenShift Container Platform cluster. For example, if your OpenShift Container Platform console address is https://console-openshift-console.apps.openshift.example.com, you must configure the wildcard certificate so that the domain is *.apps.openshift.example.com. For more information about configuring wildcard certificates, see the following topic about Creating a certificate to encrypt incoming external traffic.

- If you want to use any domain name, including those which are not subdomains of the default OpenShift Container Platform cluster domain, you must set up domain mapping for those domains. For more information, see the OpenShift Serverless documentation about Creating a custom domain mapping.

1.2. CREATING A CERTIFICATE TO ENCRYPT INCOMING EXTERNAL TRAFFIC

By default, the Service Mesh mTLS feature only secures traffic inside of the Service Mesh itself, between the ingress gateway and individual pods that have sidecars. To encrypt traffic as it flows into the OpenShift Container Platform cluster, you must generate a certificate before you enable the OpenShift Serverless and Service Mesh integration.

Prerequisites

- You have cluster administrator permissions on OpenShift Container Platform, or you have cluster or dedicated administrator permissions on Red Hat OpenShift Service on AWS or OpenShift Dedicated.

- You have installed the OpenShift Serverless Operator and Knative Serving.

- Install the OpenShift CLI (oc).
- You have created a project or have access to a project with the appropriate roles and permissions to create applications and other workloads.

Procedure

1. Create a root certificate and private key that signs the certificates for your Knative services:

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

2. Create a wildcard certificate:

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

3. Sign the wildcard certificate:

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

4. Create a secret by using the wildcard certificate:

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

This certificate is picked up by the gateways created when you integrate OpenShift Serverless with Service Mesh, so that the ingress gateway serves traffic with this certificate.

1.3. INTEGRATING SERVICE MESH WITH OPENSHIFT SERVERLESS

You can integrate Service Mesh with OpenShift Serverless without using Kourier as the default ingress. To do this, do not install the Knative Serving component before completing the following procedure. There are additional steps required when creating the KnativeServing custom resource definition (CRD) to integrate Knative Serving with Service Mesh, which are not covered in the general Knative Serving installation procedure. This procedure might be useful if you want to integrate Service Mesh as the default and only ingress for your OpenShift Serverless installation.

Prerequisites

- You have cluster administrator permissions on OpenShift Container Platform, or you have cluster or dedicated administrator permissions on Red Hat OpenShift Service on AWS or OpenShift Dedicated.

- You have created a project or have access to a project with the appropriate roles and permissions to create applications and other workloads.
• Install the Red Hat OpenShift Service Mesh Operator and create a `ServiceMeshControlPlane` resource in the `istio-system` namespace. If you want to use mTLS functionality, you must also set the `spec.security.dataPlane.mtls` field for the `ServiceMeshControlPlane` resource to `true`.

**IMPORTANT**

Using OpenShift Serverless with Service Mesh is only supported with Red Hat OpenShift Service Mesh version 2.0.5 or later.

• Install the OpenShift Serverless Operator.

• Install the OpenShift CLI (`oc`).

**Procedure**

1. Add the namespaces that you would like to integrate with Service Mesh to the `ServiceMeshMemberRoll` object as members:

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

   A list of namespaces to be integrated with Service Mesh.

2. Apply the `ServiceMeshMemberRoll` resource:

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

3. Create the necessary gateways so that Service Mesh can accept traffic:

   **Example knative-local-gateway object using HTTP**

   ```yaml
   apiVersion: networking.istio.io/v1alpha3
   kind: Gateway
   metadata:
     name: knative-ingress-gateway
     namespace: knative-serving
   spec:
     selector:
   ```
Add the name of the secret that contains the wildcard certificate.

The `knative-local-gateway` serves HTTP traffic. Using HTTP means that traffic coming from outside of Service Mesh, but using an internal hostname, such as `example.default.svc.cluster.local`, is not encrypted. You can set up encryption for this path by creating another wildcard certificate and an additional gateway that uses a different protocol spec.

Example knative-local-gateway object using HTTPS
Apply the **Gateway** resources:

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

5. Install Knative Serving by creating the following **KnativeServing** custom resource definition (CRD), which also enables the Istio integration:

```yaml
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>

---

apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  ingress:
    istio:
      enabled: true
  deployments:
  - 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. Enables Istio integration.
2. Enables sidecar injection for Knative Serving data plane pods.

6. Apply the **KnativeServing** resource:

```bash
$ oc apply -f <filename>
```
7. Install Knative Eventing by creating the following `KnativeEventing` custom resource definition (CRD), which also enables the Istio integration:

```yaml
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  config:
    features:
      istio: enabled
      workloads:
        - name: pingsource-rt-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. Enables Eventing istio controller to create a `DestinationRule` for each InMemoryChannel or KafkaChannel service.

2. Enables sidecar injection for Knative Eventing pods.

**IMPORTANT**

The Knative Eventing integration with Service Mesh is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

1. Apply the `KnativeEventing` resource:

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

2. Install Knative Kafka by creating the following `KnativeKafka` custom resource definition (CRD), which also enables the Istio integration:

   ```yaml
""
The Apache Kafka cluster URL, for example: `my-cluster-kafka-bootstrap.kafka:9092`.

Enables sidecar injection for Knative Kafka pods.
IMPORTANT

The Knative Eventing integration with Service Mesh is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope.

1. Apply the KnativeKafka resource:

   $ oc apply -f <filename>

2. Install ServiceEntry to make Red Hat OpenShift Service Mesh aware of the communication between KnativeKafka components and an Apache Kafka cluster:

   ```yaml
   apiVersion: networking.istio.io/v1alpha3
   kind: ServiceEntry
   metadata:
     name: kafka-cluster
     namespace: knative-eventing
   spec:
     hosts: 1
     - <bootstrap_servers_without_port>
     exportTo: 2
     - "."
     ports:
     - 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. The list of Apache Kafka cluster hosts, for example: my-cluster-kafka-bootstrap.kafka.

2. Apache Kafka cluster listeners ports.
NOTE

The listed ports in `spec.ports` are example TCP ports and depend on how the Apache Kafka cluster is configured.

3. Apply the ServiceEntry resource:

    $ oc apply -f <filename>

IMPORTANT

Ensure that addresses are set in the ServiceEntry. If the addresses are not set, all the traffic on the port defined in the ServiceEntry is matched regardless of the host.

Verification

1. Create a Knative Service that has sidecar injection enabled and uses a pass-through route:

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

1. A namespace that is part of the Service Mesh member roll.

2. Instructs Knative Serving to generate an OpenShift Container Platform pass-through enabled route, so that the certificates you have generated are served through the ingress gateway directly.

3. Injects Service Mesh sidecars into the Knative service pods.

2. Apply the Service resource:

    $ oc apply -f <filename>

Verification

- Access your serverless application by using a secure connection that is now trusted by the CA:
1.4. ENABLING KNATIVE SERVING METRICS WHEN USING SERVICE MESH WITH MTLS

If Service Mesh is enabled with mTLS, metrics for Knative Serving are disabled by default, because Service Mesh prevents Prometheus from scraping metrics. This section shows how to enable Knative Serving metrics when using Service Mesh and mTLS.

**Prerequisites**

- You have installed the OpenShift Serverless Operator and Knative Serving on your cluster.
- You have installed Red Hat OpenShift Service Mesh with the mTLS functionality enabled.
- You have cluster administrator permissions on OpenShift Container Platform, or you have cluster or dedicated administrator permissions on Red Hat OpenShift Service on AWS or OpenShift Dedicated.
- Install the OpenShift CLI (`oc`).
- You have created a project or have access to a project with the appropriate roles and permissions to create applications and other workloads.

**Procedure**

1. Specify `prometheus` as the `metrics.backend-destination` in the `observability` spec of the Knative Serving custom resource (CR):

   ```yaml
   apiVersion: operator.knative.dev/v1beta1
   kind: KnativeServing
   metadata:
     name: knative-serving
   spec:
     config:
       observability:
         metrics.backend-destination: "prometheus"
   ...
   
   This step prevents metrics from being disabled by default.

2. Apply the following network policy to allow traffic from the Prometheus namespace:

   ```yaml
   apiVersion: networking.k8s.io/v1
   kind: NetworkPolicy
   ```
Modify and reapply the default Service Mesh control plane in the `istio-system` namespace, so that it includes the following spec:

```yaml
metadata:
  name: allow-from-openshift-monitoring-ns
namespace: knative-serving
spec:
ingress:
  - from:
    namespaceSelector:
      matchLabels:
        name: "openshift-monitoring"
podSelector: {}...
```

3. Modify and reapply the default Service Mesh control plane in the `istio-system` namespace, so that it includes the following spec:

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

### 1.5. INTEGRATING SERVICE MESH WITH OPENSHIFT SERVERLESS WHEN KOURIER IS ENABLED

You can use Service Mesh with OpenShift Serverless even if Kourier is already enabled. This procedure might be useful if you have already installed Knative Serving with Kourier enabled, but decide to add a Service Mesh integration later.

**Prerequisites**

- You have cluster administrator permissions on OpenShift Container Platform, or you have cluster or dedicated administrator permissions on Red Hat OpenShift Service on AWS or OpenShift Dedicated.

- You have created a project or have access to a project with the appropriate roles and permissions to create applications and other workloads.

- Install the OpenShift CLI (`oc`).

- Install the OpenShift Serverless Operator and Knative Serving on your cluster.

- Install Red Hat OpenShift Service Mesh. OpenShift Serverless with Service Mesh and Kourier is supported for use with both Red Hat OpenShift Service Mesh versions 1.x and 2.x.

**Procedure**

1. Add the namespaces that you would like to integrate with Service Mesh to the `ServiceMeshMemberRoll` object as members:

```yaml
apiVersion: maistra.io/v1
```
A list of namespaces to be integrated with Service Mesh.

2. Apply the `ServiceMeshMemberRoll` resource:

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

3. Create a network policy that permits traffic flow from Knative system pods to Knative services:

   a. For each namespace that you want to integrate with Service Mesh, create a `NetworkPolicy` resource:

   ```yaml
   apiVersion: networking.k8s.io/v1
   kind: NetworkPolicy
   metadata:
     name: allow-from-serving-system-namespace
     namespace: <namespace>  
   spec:
     ingress:
       - from:
         - namespaceSelector:
           matchLabels:
             knative.openshift.io/part-of: "openshift-serverless"
           podSelector: {} 
         policyTypes:
           - Ingress  
   ```

   Add the namespace that you want to integrate with Service Mesh.
NOTE

The `knative.openshift.io/part-of: "openshift-serverless"` label was added in OpenShift Serverless 1.22.0. If you are using OpenShift Serverless 1.21.1 or earlier, add the `knative.openshift.io/part-of` label to the `knative-serving` and `knative-serving-ingress` namespaces.

Add the label to the `knative-serving` namespace:

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

Add the label to the `knative-serving-ingress` namespace:

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

b. Apply the NetworkPolicy resource:

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

1.6. IMPROVING NET-ISTIO MEMORY USAGE BY USING SECRET FILTERING FOR SERVICE MESH

By default, the informers implementation for the Kubernetes client-go library fetches all resources of a particular type. This can lead to a substantial overhead when many resources are available, which can cause the Knative net-istio ingress controller to fail on large clusters due to memory leaking. However, a filtering mechanism is available for the Knative net-istio ingress controller, which enables the controller to only fetch Knative related secrets. You can enable this mechanism by adding an annotation to the KnativeServing custom resource (CR).

IMPORTANT

If you enable secret filtering, all of your secrets need to be labeled with `networking.internal.knative.dev/certificate-uid: "<id>"`. Otherwise, Knative Serving does not detect them, which leads to failures. You must label both new and existing secrets.

Prerequisites

- You have cluster administrator permissions on OpenShift Container Platform, or you have cluster or dedicated administrator permissions on Red Hat OpenShift Service on AWS or OpenShift Dedicated.
- You have created a project or have access to a project with the appropriate roles and permissions to create applications and other workloads.
- Install Red Hat OpenShift Service Mesh. OpenShift Serverless with Service Mesh only is supported for use with Red Hat OpenShift Service Mesh version 2.0.5 or later.
- Install the OpenShift Serverless Operator and Knative Serving.
- Install the OpenShift CLI (oc).
Procedure

- Add the `serverless.openshift.io/enable-secret-informer-filtering` annotation to the `KnativeServing` CR:

Example KnativeServing CR

```yaml
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
annotations:
  serverless.openshift.io/enable-secret-informer-filtering: "true"
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] Adding this annotation injects an environment variable, `ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID=true`, to the net-istio controller pod.

NOTE

This annotation is ignored if you set a different value by overriding deployments.
CHAPTER 2. INTEGRATING SERVERLESS WITH THE COST MANAGEMENT SERVICE

Cost management is an OpenShift Container Platform service that enables you to better understand and track costs for clouds and containers. It is based on the open source Koku project.

2.1. PREREQUISITES

- You have cluster administrator permissions.
- You have set up cost management and added an OpenShift Container Platform source.

2.2. USING LABELS FOR COST MANAGEMENT QUERIES

Labels, also known as tags in cost management, can be applied for nodes, namespaces or pods. Each label is a key and value pair. You can use a combination of multiple labels to generate reports. You can access reports about costs by using the Red Hat hybrid console.

Labels are inherited from nodes to namespaces, and from namespaces to pods. However, labels are not overridden if they already exist on a resource. For example, Knative services have a default app=<revision_name> label:

Example Knative service default label

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

If you define a label for a namespace, such as app=my-domain, the cost management service does not take into account costs coming from a Knative service with the tag app=<revision_name> when querying the application using the app=my-domain tag. Costs for Knative services that have this tag must be queried under the app=<revision_name> tag.

2.3. ADDITIONAL RESOURCES

- Configure tagging for your sources
- Use the Cost Explorer to visualize your costs
CHAPTER 3. USING NVIDIA GPU RESOURCES WITH SERVERLESS APPLICATIONS

NVIDIA supports using GPU resources on OpenShift Container Platform. See GPU Operator on OpenShift for more information about setting up GPU resources on OpenShift Container Platform.

3.1. SPECIFYING GPU REQUIREMENTS FOR A SERVICE

After GPU resources are enabled for your OpenShift Container Platform cluster, you can specify GPU requirements for a Knative service using the Knative (kn) CLI.

Prerequisites

- The OpenShift Serverless Operator, Knative Serving and Knative Eventing are installed on the cluster.
- You have installed the Knative (kn) CLI.
- GPU resources are enabled for your OpenShift Container Platform cluster.
- You have created a project or have access to a project with the appropriate roles and permissions to create applications and other workloads in OpenShift Container Platform.

NOTE

Using NVIDIA GPU resources is not supported for IBM zSystems and IBM Power on OpenShift Container Platform or OpenShift Dedicated.

Procedure

1. Create a Knative service and set the GPU resource requirement limit to 1 by using the --limit nvidia.com/gpu=1 flag:

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

   A GPU resource requirement limit of 1 means that the service has 1 GPU resource dedicated. Services do not share GPU resources. Any other services that require GPU resources must wait until the GPU resource is no longer in use.

   A limit of 1 GPU also means that applications exceeding usage of 1 GPU resource are restricted. If a service requests more than 1 GPU resource, it is deployed on a node where the GPU resource requirements can be met.

2. Optional. For an existing service, you can change the GPU resource requirement limit to 3 by using the --limit nvidia.com/gpu=3 flag:

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

3.2. ADDITIONAL RESOURCES FOR OPENSHIFT CONTAINER PLATFORM

- Setting resource quotas for extended resources