Red Hat OpenShift Serverless 1.30

集成

将 OpenShift Serverless 与 Service Mesh 集成，并与成本 mangement 服务集成

Last Updated: 2023-09-20

Red Hat OpenShift Serverless 1.30 集成

将 OpenShift Serverless 与 Service Mesh 集成，并与成本 mangement 服务集成

法律通告

摘要

本文档提供有关如何将 Service Mesh 与 OpenShift Serverless 集成的信息。它还涵盖了使用成本管理服务，以帮助您了解和跟踪成本，并演示了如何在无服务器应用程序中使用 NVIDIA GPU 资源。

目录

第 1 章 将 SERVICE MESH 与 OPENSHIFT SERVERLESS 集成

OpenShift Serverless Operator 提供 Kourier 作为 Knative 的默认入口。但是，无论是否启用了 Kourier，您都可以在 OpenShift Serverless 中使用 Service Mesh。禁用 Kourier 集成后，您可以配置 Kourier ingress 不支持的额外网络和路由选项，如 mTLS 功能。

重要

OpenShift Serverless 只支持使用本指南中明确记录的 Red Hat OpenShift Service Mesh 功能，且不支持其他未记录的功能。

1.1. 先决条件

- 以下流程中的示例使用域 **example.com**。这个域的示例证书被用作为子域证书签名的证书颁发机构（CA）。
  要在部署中完成并验证这些步骤，您需要由广泛信任的公共 CA 签名的证书或您的机构提供的 CA。根据您的域、子域和 CA 调整命令示例。

- 您必须配置通配符证书，以匹配 OpenShift Container Platform 集群的域。例如，如果您的 OpenShift Container Platform 控制台地址是 **https://console-openshift-console.apps.openshift.example.com**，您必须配置通配符证书，以便域为 ***.apps.openshift.example.com**。有关配置通配符证书的更多信息，请参阅 *创建证书来加密传入的外部流量*。

- 如果要使用任何域名，包括不是默认 OpenShift Container Platform 集群域子域的域名，您必须为这些域设置域映射。如需更多信息，请参阅有关创建自定义域映射的 OpenShift Serverless 文档。

1.2. 创建证书来加密传入的外部流量

默认情况下，Service Mesh mTLS 功能只会保护 Service Mesh 本身内部的流量，在 ingress 网关和带有 sidecar 的独立 pod 间的安全。要在流向 OpenShift Container Platform 集群时对流量进行加密，您必须先生成证书，然后才能启用 OpenShift Serverless 和 Service Mesh 集成。

先决条件

- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。
- 安装了 OpenShift Serverless Operator 和 Knative Serving。
- 安装 OpenShift CLI (**oc**)。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以创建应用程序和其他工作负载。

流程

1. 创建为 Knative 服务签名的 root 证书和私钥：

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

2. 创建通配符证书：

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

3. 为通配符证书签名：

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

4. 使用通配符证书创建 secret：

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

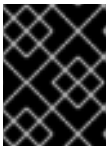
此证书由 OpenShift Serverless 与 Service Mesh 集成时创建的网关获取，以便入口网关使用此证书提供流量。

1.3. 将 SERVICE MESH 与 OPENSHIFT SERVERLESS 集成

您可以在不使用 Kourier 作为默认入口的情况下将 Service Mesh 与 OpenShift Serverless 集成。要做到这一点，在完成以下步骤前不要安装 Knative Serving 组件。在创建 **KnativeServing** 自定义资源定义 (CRD) 以将 Knative Serving 与 Service Mesh 集成时，还需要额外的步骤，这没有在一般的 Knative Serving 安装过程中覆盖。如果您要将 Service Mesh 集成为 OpenShift Serverless 安装的默认的且唯一的 ingress，这个步骤可能很有用。

先决条件

- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以创建应用程序和其他工作负载。
- 安装 Red Hat OpenShift Service Mesh Operator，并在 **istio-system** 命名空间中创建 **ServiceMeshControlPlane** 资源。如果要使用 mTLS 功能，还必须将 **ServiceMeshControlPlane** 资源的 **spec.security.dataPlane.mtls** 字段设置为 **true**。



重要

在 Service Mesh 中使用 OpenShift Serverless 只支持 Red Hat OpenShift Service Mesh 2.0.5 或更高版本。

- 安装 OpenShift Serverless Operator。
- 安装 OpenShift CLI (**oc**) 。

流程

1. 将您要与 Service Mesh 集成的命名空间作为成员添加到 **ServiceMeshMemberRoll** 对象中：

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

(1) 要与 Service Mesh 集成的命名空间列表。

重要

此命名空间列表必须包含 **knative-serving** 和 **knative-eventing** 命名空间。

2. 应用 **ServiceMeshMemberRoll** 资源：

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

3. 创建必要的网关，以便 Service Mesh 可以接受流量：

使用 HTTP 的 knative-local-gateway 对象示例

```
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 添加包含通配符证书的 secret 名称。

2 **knative-local-gateway** 提供 HTTP 流量。使用 HTTP 表示来自 Service Mesh 外部的流量不会加密，但使用内部主机名，如 **example.default.svc.cluster.local**。您可以通过创建另一个通配符证书和使用不同的 **protocol** spec 的额外网关来为这个路径设置加密。

使用 HTTPS 的 knative-local-gateway 对象示例

```
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. 应用 **Gateway** 资源：

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

5. 通过创建以下 **KnativeServing** 自定义资源定义 (CRD) 来安装 Knative Serving，该定义还启用了 Istio 集成：

```
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) 启用 Istio 集成。

(2) 为 Knative Serving data plane pod 启用 sidecar 注入。

6. 应用 **KnativeServing** 资源：

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

7. 通过创建以下 **KnativeEventing** 自定义资源定义(CRD)，该定义也可以启用 Istio 集成来安装 Knative Eventing：

```
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. 启用 Eventing istio 控制器为每个 InMemoryChannel 或 KafkaChannel 服务创建一个 **DestinationRule**。
2. 为 Knative Eventing pod 启用 sidecar 注入。

重要

Knative Eventing 与 Service Mesh 集成只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

1. 应用 **KnativeEventing** 资源：

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

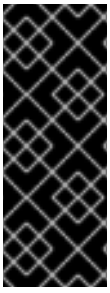
2. 通过创建以下 **KnativeKafka** 自定义资源定义(CRD)，该定义也可以启用 Istio 集成来安装 Knative Kafka：

```
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 集群 URL，例如： **my-cluster-kafka-bootstrap.kafka:9092**。

3 为 Knative Kafka pod 启用 sidecar 注入。



重要

Knative Eventing 与 Service Mesh 集成只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

1. 应用 **KnativeKafka** 资源：

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

2. 安装 **ServiceEntry** 以使 Red Hat OpenShift Service Mesh 了解 **KnativeKafka** 组件和 Apache Kafka 集群之间的通信：

```
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
```

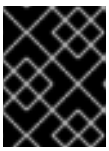
1. Apache Kafka 集群主机列表，例如： **my-cluster-kafka-bootstrap.kafka**。
2. Apache Kafka 集群监听程序端口。

注意

spec.ports 中列出的端口是 TCP 端口示例，它依赖于如何配置 Apache Kafka 集群。

3. 应用 **ServiceEntry** 资源：

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



重要

确保在 ServiceEntry 中设置地址。如果没有设置地址，无论主机是什么，ServiceEntry 中定义的端口上的所有流量都会匹配。

验证

1. 创建一个启用了 sidecar 注入并使用 pass-through 路由的 Knative Service：

```
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. 作为 Service Mesh member roll 一部分的命名空间。
2. 指示 Knative Serving 生成 OpenShift Container Platform 直通启用路由，以便您已生成的证书直接通过 ingress 网关提供。
3. 将 Service Mesh sidecar 注入 Knative 服务 pod。

2. 应用 **Service** 资源：

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

验证

- 使用 CA 信任的安全连接访问无服务器应用程序：

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

示例命令

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

输出示例

```
Hello Openshift!
```

1.4. 在使用带有 MTLS 的 SERVICE MESH 时启用 KNATIVE SERVING 指标

如果启用了 mTLS 的 Service Mesh，则默认禁用 Knative Serving 的指标，因为 Service Mesh 会防止 Prometheus 提取指标。本节介绍在使用 Service Mesh 和 mTLS 时如何启用 Knative Serving 指标。

先决条件

- 您已在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 已安装了启用了 mTLS 功能的 Red Hat OpenShift Service Mesh。
- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。
- 安装 OpenShift CLI (**oc**) 。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以创建应用程序和其他工作负载。

流程

1. 在 Knative Serving 自定义资源 (CR) 的 **observability** spec 中将 **prometheus** 指定为 **metrics.backend-destination** ：

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

此步骤可防止默认禁用指标。

2. 应用以下网络策略来允许来自 Prometheus 命名空间中的流量：

```
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. 修改并重新应用 **istio-system** 命名空间中的默认 Service Mesh control plane，使其包含以下 spec：

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

1.5. 在启用了 KOURIER 时将 SERVICE MESH 与 OPENSHIFT SERVERLESS 集成

即使已经启用了 Kourier，您也可以在 OpenShift Serverless 中使用 Service Mesh。如果您已在启用了 Kourier 的情况下安装了 Knative Serving，但决定在以后添加 Service Mesh 集成，这个过程可能会很有用。

先决条件

- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以创建应用程序和其他工作负载。
- 安装 OpenShift CLI (**oc**) 。
- 在集群上安装 OpenShift Serverless Operator 和 Knative Serving。
- 安装 Red Hat OpenShift Service Mesh。带有 Service Mesh 和 Kourier 的 OpenShift Serverless 支持与 Red Hat OpenShift Service Mesh 1.x 和 2.x 版本搭配使用。

流程

1. 将您要与 Service Mesh 集成的命名空间作为成员添加到 **ServiceMeshMemberRoll** 对象中：

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

1 要与 Service Mesh 集成的命名空间列表。

2. 应用 **ServiceMeshMemberRoll** 资源：

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

3. 创建允许 Knative 系统 Pod 到 Knative 服务流量的网络策略：

   a. 对于您要与 Service Mesh 集成的每个命名空间，创建一个 **NetworkPolicy** 资源：

```
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 添加您要与 Service Mesh 集成的命名空间。

注意

knative.openshift.io/part-of: "openshift-serverless" 标签添加到 OpenShift Serverless 1.22.0 中。如果使用 OpenShift Serverless 1.21.1 或更早版本，请将 **knative.openshift.io/part-of** 标签添加到 **knative-serving** 和 **knative-serving-ingress** 命名空间。

将标签添加到 **knative-serving** 命名空间：

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

将标签添加到 **knative-serving-ingress** 命名空间：

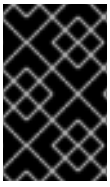
```
$ oc label namespace knative-serving-ingress knative.openshift.io/part-of=openshift-serverless
```

b. 应用 **NetworkPolicy** 资源：

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

1.6. 为 SERVICE MESH 使用 SECRET 过滤来提高 NET-ISTIO 内存用量

默认情况下，Kubernetes **client-go** 库的 informers 实施会获取特定类型的所有资源。当有很多资源可用时，这可能会导致大量资源出现大量开销，这可能会导致 Knative **net-istio** 入口控制器因为内存泄漏而在大型集群中失败。但是，一个过滤机制可用于 Knative **net-istio** ingress 控制器，它可让控制器只获取 Knative 相关的 secret。您可以通过在 **KnativeServing** 自定义资源 (CR) 中添加注解来启用此机制。



重要

如果启用 secret 过滤，则所有 secret 都需要使用 **networking.internal.knative.dev/certificate-uid: "<id>"**。否则，Knative Serving 不会检测到它们，这会导致失败。您必须标记新的和现有的 secret。

先决条件

- 在 OpenShift Container Platform 上具有集群管理员权限，或者对 Red Hat OpenShift Service on AWS 或 OpenShift Dedicated 有集群或专用管理员权限。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以创建应用程序和其他工作负载。
- 安装 Red Hat OpenShift Service Mesh。带有 Service Mesh 的 OpenShift Serverless 仅支持与 Red Hat OpenShift Service Mesh 2.0.5 或更高版本搭配使用。
- 安装 OpenShift Serverless Operator 和 Knative Serving。
- 安装 OpenShift CLI (**oc**) 。

流程

- 将 **serverless.openshift.io/enable-secret-informer-filtering** 注解添加到 **KnativeServing** CR：

  KnativeServing CR 示例

```
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 添加此注解会将 enviroment 变量 **ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID=true** 注入到 **net-istio** 控制器 pod。

注意

如果您通过覆盖部署设置了不同的值，则忽略此注解。

第 2 章 将 SERVERLESS 与成本管理服务集成

Cost management 是一种 OpenShift Container Platform 服务，可让您更好地了解和跟踪云和容器的成本。它基于开源 Koku 项目。

2.1. 先决条件

- 有集群管理员权限。
- 您已设置成本管理，并添加了 OpenShift Container Platform 源。

2.2. 使用标签进行成本管理查询

标签（label）（在成本管理中也称为 *tag* ）可用于节点、命名空间或 pod。每个标签都是键和值对。您可以使用多个标签的组合来生成报告。您可以使用红帽混合控制台访问成本的相关报告。

标签从节点继承到命名空间，并从命名空间继承到 pod。但是，如果标签已在资源中已存在，则标签不会被覆盖。例如，Knative 服务具有默认的 **app=<revision_name>** 标签：

Knative 服务默认标签示例

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

如果您为命名空间定义标签，如 **app=my-domain**，在查询使用 **app=my-domain** 标签的应用程序时，成本管理服务不会考虑带有 **app=<revision_name>** 标签的 Knative 服务的成本。具有此标签的 Knative 服务的成本必须在 **app=<revision_name>** 标签下查询。

2.3. 其他资源

- 为您的源配置标签
- 使用 Cost Explorer 来视觉化您的成本

第 3 章 使用无服务器应用程序的 NVIDIA GPU 资源

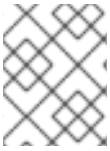
NVIDIA 支持在 OpenShift Container Platform 上使用 GPU 资源。如需有关在 OpenShift Container Platform 中设置 GPU 资源的更多信息，请参阅 OpenShift 上的 GPU Operator。

3.1. 为服务指定 GPU 要求

为 OpenShift Container Platform 集群启用 GPU 资源后，您可以使用 Knative (**kn**) CLI 为 Knative 服务指定 GPU 要求。

先决条件

- 在集群中安装了 OpenShift Serverless Operator、Knative Serving 和 Knative Eventing。
- 已安装 Knative (**kn**) CLI。
- 为 OpenShift Container Platform 集群启用 GPU 资源。
- 您已创建了一个项目，或者具有适当的角色和权限访问项目，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。



注意

在 OpenShift Container Platform 或 OpenShift Dedicated 上的 IBM zSystems 和 IBM Power 不支持使用 NVIDIA GPU 资源。

流程

1. 创建 Knative 服务并使用 **--limit nvidia.com/gpu=1** 标志将 GPU 资源要求限制设置为 **1**：

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

   GPU 资源要求限制为 **1** 表示该服务有 1 个专用的 GPU 资源。服务不共享 GPU 资源。所有需要 GPU 资源的其他服务都必须等待 GPU 资源不再被使用为止。

   限值为 1 个 GPU 意味着超过使用 1 个 GPU 资源的应用程序会受到限制。如果服务请求超过 1 个 GPU 资源，它将部署到可以满足 GPU 资源要求的节点。

2. 可选。对于现有服务，您可以使用 **--limit nvidia.com/gpu=3** 标志将 GPU 资源要求限制改为 **3**：

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

3.2. OPENSHIFT CONTAINER PLATFORM 的其他资源

- 为扩展资源设置资源配额