Red Hat OpenShift Serverless 1.33

Serving

开始使用 Knative Serving 并配置服务

Last Updated: 2024-12-18

Red Hat OpenShift Serverless 1.33 Serving

开始使用 Knative Serving 并配置服务

法律通告

摘要

本文档提供有关开始使用 Knative Serving 的信息。它演示了如何配置应用程序，并涵盖自动扩展、流量分割和外部和入口路由等功能。

目录

第 1 章 KNATIVE SERVING 入门

1.1. 创建无服务器应用程序

无服务器应用程序已创建并部署为 Kubernetes 服务，由路由和配置定义，并包含在 YAML 文件中。要使用 OpenShift Serverless 部署无服务器应用程序，您必须创建一个 Knative **Service** 对象。

Knative Service 对象 YAML 文件示例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: showcase 1
  namespace: default 2
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/showcase 3
          env:
            - name: GREET 4
              value: Ciao
```

1. 应用程序的名称。
2. 应用程序使用的命名空间。
3. 应用程序的镜像。
4. 示例应用程序输出的环境变量。

使用以下任一方法创建一个无服务器应用程序：

- 从 OpenShift Container Platform web 控制台创建 Knative 服务。
  对于 OpenShift Container Platform，请参阅使用 Developer 视角创建应用程序。
- 使用 Knative (**kn**) CLI 创建 Knative 服务。
- 使用 **oc** CLI 创建并应用 Knative **Service** 对象作为 YAML 文件。

1.1.1. 使用 Knative CLI 创建无服务器应用程序

通过使用 Knative (**kn**) CLI 创建无服务器应用程序，通过直接修改 YAML 文件来提供更精简且直观的用户界面。您可以使用 **kn service create** 命令创建基本无服务器应用程序。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 已安装 Knative (**kn**) CLI。
- 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。

流程

- 创建 Knative 服务：

```
$ kn service create <service-name> --image <image> --tag <tag-value>
```

其中：

- **--image** 是应用的镜像的 URI。
- **--tag** 是一个可选标志，可用于向利用服务创建的初始修订版本添加标签。

示例命令

```
$ kn service create showcase \
    --image quay.io/openshift-knative/showcase
```

输出示例

```
Creating service 'showcase' in namespace 'default':

  0.271s The Route is still working to reflect the latest desired specification.
  0.580s Configuration "showcase" is waiting for a Revision to become ready.
  3.857s ...
  3.861s Ingress has not yet been reconciled.
  4.270s Ready to serve.

Service 'showcase' created with latest revision 'showcase-00001' and URL:
http://showcase-default.apps-crc.testing
```

1.1.2. 使用 YAML 创建无服务器应用程序

使用 YAML 文件创建 Knative 资源使用声明性 API，它允许您以声明性的方式描述应用程序，并以可重复的方式描述应用程序。要使用 YAML 创建无服务器应用程序，您必须创建一个 YAML 文件来定义 Knative **Service** 对象，然后使用 **oc apply** 来应用它。

创建服务并部署应用程序后，Knative 会为应用程序的这个版本创建一个不可变的修订版本。Knative 还将执行网络操作，为您的应用程序创建路由、入口、服务和负载平衡器，并根据流量自动扩展或缩减 pod。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 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。
- 安装 OpenShift CLI (**oc**) 。

流程

1. 创建包含以下示例代码的 YAML 文件：

```
apiVersion: serving.knative.dev/v1
kind: Service
```

```
metadata:
  name: showcase
  namespace: default
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/showcase
          env:
            - name: GREET
              value: Bonjour
```

2. 导航到包含 YAML 文件的目录，并通过应用 YAML 文件来部署应用程序：

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

如果您不想在 OpenShift Container Platform web 控制台中切换到 **Developer** 视角，或使用 Knative (**kn**) CLI 或 YAML 文件，您可以使用 OpenShift Container Platform Web 控制台的 **Administator** 视角创建 Knative 组件。

1.1.3. 使用管理员视角创建无服务器应用程序

无服务器应用程序已创建并部署为 Kubernetes 服务，由路由和配置定义，并包含在 YAML 文件中。要使用 OpenShift Serverless 部署无服务器应用程序，您必须创建一个 Knative **Service** 对象。

Knative Service 对象 YAML 文件示例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: showcase 1
  namespace: default 2
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/showcase 3
          env:
            - name: GREET 4
              value: Ciao
```

1. 应用程序的名称。
2. 应用程序使用的命名空间。
3. 应用程序的镜像。
4. 示例应用程序输出的环境变量。

创建服务并部署应用程序后，Knative 会为应用程序的这个版本创建一个不可变的修订版本。Knative 还将执行网络操作，为您的应用程序创建路由、入口、服务和负载平衡器，并根据流量自动扩展或缩减 pod。

先决条件

要使用 **管理员** 视角创建无服务器应用程序，请确定您已完成了以下步骤。

- 安装了 OpenShift Serverless Operator 和 Knative Serving。
- 您已登录到 Web 控制台，且处于 **Administrator** 视角。

流程

1. 进入 **Serverless → Serving** 页面。
2. 在 **Create** 列表中，选择 **Service**。
3. 手动输入 YAML 或 JSON 定义，或者将文件拖放到编辑器中。
4. 点 **Create**。

1.1.4. 使用离线模式创建服务

您可以在离线模式下执行 **kn service** 命令，以便集群中不会发生任何更改，而是在本地机器上创建服务描述符文件。创建描述符文件后，您可以在向集群传播更改前修改该文件。

重要

Knative CLI 的离线模式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。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 已安装 Knative (**kn**) CLI。

流程

1. 在离线模式下，创建一个本地 Knative 服务描述符文件：

```
$ kn service create showcase \
   --image quay.io/openshift-knative/showcase \
   --target ./ \
   --namespace test
```

输出示例

```
Service 'showcase' created in namespace 'test'.
```

- **--target ./** 标志启用脱机模式，并将 **./** 指定为用于存储新目录树的目录。
  如果您没有指定现有目录，但使用文件名，如 **--target my-service.yaml**，则不会创建目录树。相反，当前目录中只创建服务描述符 **my-service.yaml** 文件。

  文件名可以具有 **.yaml**、**.yml** 或 **.json** 扩展名。选择 **.json** 以 JSON 格式创建服务描述符文件。

- **namespace test** 选项将新服务放在 **test** 命名空间中。
  如果不使用 **--namespace**，且您登录到 OpenShift Container Platform 集群，则会在当前命名空间中创建描述符文件。否则，描述符文件会在 **default** 命名空间中创建。

2. 检查创建的目录结构：

```
$ tree ./
```

输出示例

```
./
└── test
    └── ksvc
        └── showcase.yaml

2 directories, 1 file
```

- 使用 **--target** 指定的当前 **./** 目录包含新的 **test/** 目录，它在指定的命名空间后命名。
- **test/** 目录包含 **ksvc**，它在资源类型后命名。
- **ksvc** 目录包含描述符文件 **showcase.yaml**，根据指定的服务名称命名。

3. 检查生成的服务描述符文件：

```
$ cat test/ksvc/showcase.yaml
```

输出示例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  creationTimestamp: null
  name: showcase
  namespace: test
spec:
  template:
    metadata:
      annotations:
        client.knative.dev/user-image: quay.io/openshift-knative/showcase
      creationTimestamp: null
    spec:
      containers:
      - image: quay.io/openshift-knative/showcase
        name: ""
        resources: {}
status: {}
```

4. 列出新服务的信息：

```
$ kn service describe showcase --target ./ --namespace test
```

输出示例

```
Name:       showcase
Namespace:  test
Age:
URL:

Revisions:

Conditions:
  OK TYPE    AGE REASON
```

- **--target ./** 选项指定包含命名空间子目录的目录结构的根目录。
  另外，您可以使用 **--target** 选项直接指定 YAML 或 JSON 文件名。可接受的文件扩展包括 **.yaml**、**.yml** 和 **.json**。

- **--namespace** 选项指定命名空间，与 **kn** 通信包含所需服务描述符文件的子目录。
  如果不使用 **--namespace**，并且您登录到 OpenShift Container Platform 集群，**kn** 会在以当前命名空间命名的子目录中搜索该服务。否则，**kn** 在 **default/** 子目录中搜索。

5. 使用服务描述符文件在集群中创建服务：

```
$ kn service create -f test/ksvc/showcase.yaml
```

输出示例

```
Creating service 'showcase' in namespace 'test':

  0.058s The Route is still working to reflect the latest desired specification.
  0.098s ...
  0.168s Configuration "showcase" is waiting for a Revision to become ready.
 23.377s ...
 23.419s Ingress has not yet been reconciled.
 23.534s Waiting for load balancer to be ready
 23.723s Ready to serve.

Service 'showcase' created to latest revision 'showcase-00001' is available at URL:
http://showcase-test.apps.example.com
```

1.1.5. 验证无服务器应用程序的部署

要验证您的无服务器应用程序是否已成功部署，您必须获取 Knative 创建的应用程序的 URL，然后向该 URL 发送请求并检查其输出。OpenShift Serverless 支持 HTTP 和 HTTPS URL，但 **oc get ksvc** 的输出始终使用 **http://** 格式打印 URL。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 已安装 **oc** CLI。
- 您已创建了 Knative 服务。

先决条件

- 安装 OpenShift CLI (**oc**) 。

流程

1. 查找应用程序 URL:

```
$ oc get ksvc <service_name>
```

输出示例

```
NAME URL LATESTCREATED LATESTREADY READY REASON
showcase http://showcase-default.example.com showcase-00001 showcase-00001 True
```

2. 向集群发出请求并观察其输出。

HTTP 请求示例（使用 HTTPie 工具）

```
$ http showcase-default.example.com
```

HTTPS 请求示例

```
$ https showcase-default.example.com
```

输出示例

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Quarkus/2.13.7.Final-redhat-00003 Java/17.0.7
X-Config: {"sink":"http://localhost:31111","greet":"Ciao","delay":0}
X-Version: v0.7.0-4-g23d460f
content-length: 49

{
   "artifact": "knative-showcase",
   "greeting": "Ciao"
}
```

3. 可选。如果您的系统中没有安装 HTTPie 工具，您可以使用 curl 工具：

HTTPS 请求示例

```
$ curl http://showcase-default.example.com
```

输出示例

```
{"artifact":"knative-showcase","greeting":"Ciao"}
```

4. 可选。如果您在证书链中收到与自签名证书相关的错误，您可以在 HTTPie 命令中添加 **--verify=no** 标志来忽略错误：

```
$ https --verify=no showcase-default.example.com
```

输出示例

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Quarkus/2.13.7.Final-redhat-00003 Java/17.0.7
X-Config: {"sink":"http://localhost:31111","greet":"Ciao","delay":0}
X-Version: v0.7.0-4-g23d460f
content-length: 49

{
    "artifact": "knative-showcase",
    "greeting": "Ciao"
}
```

重要

在生产部署中不能使用自签名证书。这个方法仅用于测试目的。

5. 可选。如果 OpenShift Container Platform 集群配置有证书颁发机构 (CA) 签名但尚未为您的系统配置全局证书，您可以使用 **curl** 命令指定此证书。证书的路径可使用 **--cacert** 标志传递给 curl 命令：

```
$ curl https://showcase-default.example.com --cacert <file>
```

输出示例

```
{"artifact":"knative-showcase","greeting":"Ciao"}
```

1.1.6. 其他资源

- Knative Serving CLI 命令
- 为 Knative 服务配置 JSON Web 令牌身份验证

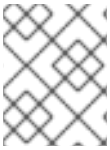
第 2 章 自动缩放

2.1. 自动缩放

Knative Serving 为应用程序提供自动扩展功能（或 *autoscaling*），以满足传入的需求。例如，如果应用程序没有流量，并且启用了缩减到零，Knative Serving 将应用程序缩减为零个副本。如果缩减到零，则应用程序会缩减到为集群中的应用程序配置的最小副本数。如果应用流量增加，也可以向上扩展副本来满足需求。

Knative 服务的自动扩展设置可以是由集群管理员配置的全局设置（或集群管理员为 Red Hat OpenShift Service on AWS 和 OpenShift Dedicated 的专用管理员）或为单个服务配置的每个修订设置。

您可以使用 OpenShift Container Platform Web 控制台修改服务的每个修订设置，方法是修改服务的 YAML 文件，或使用 Knative (**kn**) CLI 修改服务。



注意

您为服务设置的任何限制或目标均是针对应用程序的单个实例来衡量。例如，将 **target** 注解设置为 **50** 可将自动扩展器配置为缩放应用程序，以便每个修订一次处理 50 个请求。

2.2. 扩展范围

缩放范围决定了可在任意给定时间为应用程序服务的最小和最大副本数。您可以为应用设置规模绑定，以帮助防止冷启动和控制计算成本。

2.2.1. 最小扩展范围

为应用程序提供服务的最小副本数量由 **min-scale** 注解决定。如果没有启用缩减为零，则 **min-scale** 值默认为 **1**。

如果满足以下条件，**min-scale** 值默认为 **0** 个副本：

- 不设置 **min-scale** 注解
- 启用扩展到零
- 使用类 **KPA**

带有 min-scale 注解的 service spec 示例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: showcase
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "0"
...
```

2.2.1.1. 使用 Knative CLI 设置 min-scale 注解

使用 Knative (**kn**) CLI 设置 **min-scale** 注解，比直接修改 YAML 文件提供了一个更加精简且直观的用户界面。您可以使用带有 **--scale-min** 标志的 **kn service** 命令为服务创建或修改 **min-scale** 值。

先决条件

- 在集群中安装了 Knative Serving。
- 已安装 Knative (**kn**) CLI。

流程

- 使用 **--scale-min** 标志设置服务的最小副本数：

```
$ kn service create <service_name> --image <image_uri> --scale-min <integer>
```

示例命令

```
$ kn service create showcase --image quay.io/openshift-knative/showcase --scale-min 2
```

2.2.2. 最大扩展范围

可提供应用程序的副本数量由 **max-scale** 注解决定。如果没有设置 **max-scale** 注解，则创建的副本数没有上限。

带有 max-scale 注解的 service spec 示例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: showcase
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/max-scale: "10"
...
```

2.2.2.1. 使用 Knative CLI 设置 max-scale 注解

使用 Knative (**kn**) CLI 设置 **max-scale** 注解，比直接修改 YAML 文件提供了一个更精简且直观的用户界面。您可以使用带有 **--scale-max** 标志的 **kn service** 命令为服务创建或修改 **max-scale** 值。

先决条件

- 在集群中安装了 Knative Serving。
- 已安装 Knative (**kn**) CLI。

流程

- 使用 **--scale-max** 标志设置服务的最大副本数：

```
$ kn service create <service_name> --image <image_uri> --scale-max <integer>
```

示例命令

```
$ kn service create showcase --image quay.io/openshift-knative/showcase --scale-max 10
```

2.3. 并发

并发请求数决定了应用程序的每个副本可在任意给定时间处理的并发请求数。并发可以配置为*软限制*或*硬限制*：

- 软限制是目标请求限制，而不是严格实施的绑定。例如，如果流量突发，可以超过软限制目标。
- 硬限制是严格实施的上限请求限制。如果并发达到硬限制，则请求将被缓冲，必须等到有足够的可用容量来执行请求。

重要

只有在应用程序中明确用例时才建议使用硬限制配置。指定较少的硬限制可能会对应用程序的吞吐量和延迟造成负面影响，并可能导致冷启动。

添加软目标和硬限制意味着自动扩展以并发请求的软目标数为目标，但为请求的最大数量施加硬限制值。

如果硬限制值小于软限制值，则软限制值将降级，因为不需要将目标设定为多于实际处理的请求数。

2.3.1. 配置软并发目标

软限制是目标请求限制，而不是严格实施的绑定。例如，如果流量突发，可以超过软限制目标。您可以通过在 spec 中设置 **autoscaling.knative.dev/target** 注解，或者使用带有正确标记的 **kn service** 命令为 Knative 服务指定软并发目标。

流程

- 可选：在 **Service** 自定义资源的 spec 中为您的 Knative 服务设置 **autoscaling.knative.dev/target** 注解：

  服务规格示例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: showcase
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/target: "200"
```

- 可选： 使用 **kn service** 命令指定 **--concurrency-target** 标志：

```
$ kn service create <service_name> --image <image_uri> --concurrency-target <integer>
```

创建服务的示例，并发目标为 50 请求

```
$ kn service create showcase --image quay.io/openshift-knative/showcase --concurrency-target 50
```

2.3.2. 配置硬并发限制

硬并发限制是严格强制执行上限的上限。如果并发达到硬限制，则请求将被缓冲，必须等到有足够的可用容量来执行请求。您可以通过修改 **containerConcurrency** spec 或使用带有正确标记的 **kn service** 命令为 Knative 服务指定硬并发限制。

流程

- 可选：在 **Service** 自定义资源的 spec 中为您的 Knative 服务设置 **containerConcurrency** spec：

  服务规格示例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: showcase
  namespace: default
spec:
  template:
    spec:
      containerConcurrency: 50
```

  默认值为 **0**，这意味着允许同时访问服务的一个副本的请求数量没有限制。

  大于 **0** 的值指定允许一次传输到服务的一个副本的请求的确切数量。这个示例将启用 50 个请求的硬并发限制。

- 可选： 使用 **kn service** 命令指定 **--concurrency-limit** 标志：

```
$ kn service create <service_name> --image <image_uri> --concurrency-limit <integer>
```

  创建服务且并发限制为 50 个请求的命令示例

```
$ kn service create showcase --image quay.io/openshift-knative/showcase --concurrency-limit 50
```

2.3.3. 并发目标使用率

此值指定自动扩展实际的目标并发限制的百分比。这也称为指定运行副本的*热性（hotness）*，允许自动扩展在达到定义的硬限制前进行扩展。

例如，如果 **containerConcurrency** 值设置为 10，并且 **target-utilization-percentage** 值设置为 70%，则自动扩展会在所有现有副本的平均并发请求数量达到 7 时创建一个新的副本。编号为 7 到 10 的请求仍然会被发送到现有的副本，但达到 **containerConcurrency** 值后会启动额外的副本。

使用 target-utilization-percentage 注解配置的服务示例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: showcase
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/target-utilization-percentage: "70"
...
```

2.4. SCALE-TO-ZERO

Knative Serving 为应用程序提供自动扩展功能（或 *autoscaling*），以满足传入的需求。

2.4.1. 启用 scale-to-zero

您可以使用 **enable-scale-to-zero** spec，为集群中的应用程序全局启用或禁用 scale-to-zero。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 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。
- 使用默认的 Knative Pod Autoscaler。如果使用 Kubernetes Horizontal Pod Autoscaler，则缩减为零功能将不可用。

流程

- 在 **KnativeServing** 自定义资源 (CR) 中修改 **enable-scale-to-zero** spec：

  KnativeServing CR 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    autoscaler:
      enable-scale-to-zero: "false" 1
```

1 **enable-scale-to-zero** spec 可以是 **"true"** 或 **"false"**。如果设置为 true，则会启用 scale-to-zero。如果设置为 false，应用程序将缩减至配置的*最小扩展绑定*。默认值为 **"true"**。

2.4.2. 配置 scale-to-zero 宽限期

Knative Serving 为应用程序提供自动缩放为零个 pod。您可以使用 **scale-to-zero-grace-period** spec 定义上限，Knative 在删除应用程序的最后一个副本前等待 scale-to-zero machinery 原位。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 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。
- 使用默认的 Knative Pod Autoscaler。如果使用 Kubernetes Horizontal Pod Autoscaler，则缩减为零功能将不可用。

流程

- 在 **KnativeServing** 自定义资源 (CR) 中修改 **scale-to-zero-grace-period** spec：

  KnativeServing CR 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    autoscaler:
      scale-to-zero-grace-period: "30s" 1
```

1 宽限期（以秒为单位）。默认值为 30 秒。

第 3 章 配置 OPENSHIFT SERVERLESS 应用程序

3.1. 对 SERVING 的多容器支持

您可以使用单个 Knative 服务部署多容器 pod。这个方法可用于将应用程序职责划分为较小的特殊部分。

3.1.1. 配置多容器服务

默认启用多容器支持。您可以通过指定服务中的多个容器来创建多容器 pod。

流程

1. 修改您的服务使其包含其他容器。只有一个容器可以处理请求，因此为一个容器指定一个 **端口**。以下是有两个容器的示例配置：

多容器配置

```
apiVersion: serving.knative.dev/v1
kind: Service
...
spec:
  template:
    spec:
      containers:
        - name: first-container (1)
          image: gcr.io/knative-samples/helloworld-go
          ports:
            - containerPort: 8080 (2)
        - name: second-container (3)
          image: gcr.io/knative-samples/helloworld-java
```

(1) 第一个容器配置。

(2) 第一个容器的端口规格。

(3) 第二个容器配置。

3.2. EMPTYDIR 卷

emptyDir 卷是创建 pod 时创建的空卷，用来提供临时工作磁盘空间。当为其创建 pod 被删除时，**emptyDir** 卷会被删除。

3.2.1. 配置 EmptyDir 扩展

kubernetes.podspec-volumes-emptydir 扩展控制 **emptyDir** 卷是否与 Knative Serving 搭配使用。要使用 **emptyDir** 卷启用，您必须修改 **KnativeServing** 自定义资源 (CR) 使其包含以下 YAML：

KnativeServing CR 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
```

```
metadata:
  name: knative-serving
spec:
  config:
    features:
      kubernetes.podspec-volumes-emptydir: enabled
...
```

3.3. SERVING 的持久性卷声明

有些无服务器应用程序需要持久性数据存储。通过配置不同的卷类型，您可以为 Knative 服务提供数据存储。服务支持挂载卷类型，如 **secret**、**configMap**、**投射** 和 **emptyDir**。

您可以为 Knative 服务配置持久性卷声明(PVC)。持久性卷类型作为插件实现。要确定是否有可用的持久性卷类型，您可以检查集群中的可用或已安装的存储类。支持持久性卷，但需要启用功能标记。

警告

挂载大型卷可能会导致应用程序开始出现显著延迟。

3.3.1. 启用 PVC 支持

流程

1. 要启用 Knative Serving 使用 PVC 并写入它们，请修改 **KnativeServing** 自定义资源 (CR) 使其包含以下 YAML：

   启用具有写入访问的 PVC

   ```
   ...
   spec:
     config:
       features:
         "kubernetes.podspec-persistent-volume-claim": enabled
         "kubernetes.podspec-persistent-volume-write": enabled
   ...
   ```

   - **kubernetes.podspec-persistent-volume-claim** 扩展控制持久性卷 (PV) 是否可以用于 Knative Serving。
   - **kubernetes.podspec-persistent-volume-write** 扩展控制 Knative Serving 是否使用写入访问权限。

2. 要声明 PV，请修改您的服务使其包含 PV 配置。例如，您可能具有以下配置的持久性卷声明：

注意

使用支持您要请求的访问模式的存储类。例如，您可以将 **ocs-storagecluster-cephfs** 存储类用于 **ReadWriteMany** 访问模式。

ocs-storagecluster-cephfs 存储类被支持，并来自 Red Hat OpenShift Data Foundation。

PersistentVolumeClaim 配置

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pv-claim
  namespace: my-ns
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ocs-storagecluster-cephfs
  resources:
    requests:
      storage: 1Gi
```

在这种情况下，若要声明具有写访问权限的 PV，请修改服务，如下所示：

Knative 服务 PVC 配置

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  namespace: my-ns
...
spec:
 template:
   spec:
     containers:
         ...
         volumeMounts: (1)
           - mountPath: /data
             name: mydata
             readOnly: false
     volumes:
       - name: mydata
         persistentVolumeClaim: (2)
           claimName: example-pv-claim
           readOnly: false (3)
```

1. 卷挂载规格。
2. 持久性卷声明规格。
3. 启用只读访问的标记。

注意

要在 Knative 服务中成功使用持久性存储，您需要额外的配置，如 Knative 容器用户的用户权限。

3.3.2. OpenShift Container Platform 的其他资源

- 了解持久性存储

3.4. INIT 容器

Init 容器是 pod 中应用程序容器之前运行的专用容器。它们通常用于为应用程序实施初始化逻辑，其中可能包括运行设置脚本或下载所需的配置。您可以通过修改 **KnativeServing** 自定义资源 (CR) 来启用 init 容器用于 Knative 服务。

注意

Init 容器可能会导致应用程序的启动时间较长，应该谨慎地用于无服务器应用程序，这应该经常被扩展或缩减。

3.4.1. 启用 init 容器

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 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。

流程

- 通过在 **KnativeServing** CR 中添加 **kubernetes.podspec-init-containers** 标记来启用 init 容器的使用：

  KnativeServing CR 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    features:
      kubernetes.podspec-init-containers: enabled
...
```

3.5. 将镜像标签解析到摘要

如果 Knative Serving 控制器可以访问容器 registry，Knative Serving 会在创建服务的修订时将镜像标签解析为摘要。这被称为 *tag-to-digest 解析*，有助于为部署提供一致性。

3.5.1. tag-to-digest 解析

要让控制器访问 OpenShift Container Platform 上的容器 registry，您必须创建一个 secret，然后配置控制器自定义证书。您可以通过修改 **KnativeServing** 自定义资源 (CR) 中的 **controller-custom-certs** spec 来配置控制器自定义证书。secret 必须位于与 **KnativeServing** CR 相同的命名空间中。

如果 **KnativeServing** CR 中不包含 secret，此设置默认为使用公钥基础架构 (PKI) 。在使用 PKI 时，集群范围的证书会使用 **config-service-sa** 配置映射自动注入到 Knative Serving 控制器。OpenShift Serverless Operator 使用集群范围证书填充 **config-service-sa** 配置映射，并将配置映射作为卷挂载到控制器。

3.5.1.1. 使用 secret 配置 tag-to-digest 解析

如果 **controller-custom-certs** spec 使用 **Secret** 类型，secret 将被挂载为 secret 卷。Knative 组件直接使用 secret，假设 secret 具有所需的证书。

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。
- 您已在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。

流程

1. 创建 secret：

   示例命令

   ```
   $ oc -n knative-serving create secret generic custom-secret --from-file=<secret_name>.crt=
   <path_to_certificate>
   ```

2. 配置 **KnativeServing** 自定义资源 (CR) 中的 **controller-custom-certs** 规格以使用 **Secret** 类型：

   KnativeServing CR 示例

   ```
   apiVersion: operator.knative.dev/v1beta1
   kind: KnativeServing
   metadata:
     name: knative-serving
     namespace: knative-serving
   spec:
     controller-custom-certs:
       name: custom-secret
       type: Secret
   ```

3.6. 配置 TLS 身份验证

您可以使用 *传输层安全* (TLS) 加密 Knative 流量并进行身份验证。

TLS 是 Knative Kafka 唯一支持的流量加密方法。红帽建议将 SASL 和 TLS 用于 Apache Kafka 资源的 Knative 代理。

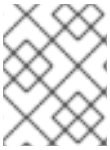
注意

如果要使用 Red Hat OpenShift Service Mesh 集成启用内部 TLS，您必须使用 mTLS 启用 Service Mesh，而不是按照以下流程所述的内部加密。

对于 OpenShift Container Platform 和 Red Hat OpenShift Service on AWS，请参阅 在使用带有 mTLS 的 Service Mesh 时启用 Knative Serving 指标 的文档。

3.6.1. 为内部流量启用 TLS 身份验证

OpenShift Serverless 默认支持 TLS 边缘终止，以便最终用户的 HTTPS 流量加密。但是，OpenShift 路由后面的内部流量使用普通数据转发到应用。通过为内部流量启用 TLS，组件间发送的流量会进行加密，从而使此流量更加安全。



注意

如果要使用 Red Hat OpenShift Service Mesh 集成启用内部 TLS，您必须使用 mTLS 启用 Service Mesh，而不是按照以下流程所述的内部加密。

重要

内部 TLS 加密支持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。

先决条件

- 安装了 OpenShift Serverless Operator 和 Knative Serving。
- 已安装 OpenShift (**oc**) CLI。

流程

1. 创建或更新 **KnativeServing** 资源，并确保它在 spec 中包含 **internal-encryption: "true"** 字段：

```
...
spec:
  config:
    network:
      internal-encryption: "true"
...
```

2. 重启 **knative-serving** 命名空间中的 activator pod 来加载证书：

```
$ oc delete pod -n knative-serving --selector app=activator
```

其他资源

- 为 Apache Kafka 的 Knative 代理配置 TLS 身份验证
- 为 Apache Kafka 的频道配置 TLS 身份验证

- 在使用带有 mTLS 的 Service Mesh 时启用 Knative Serving 指标

3.7. 配置 KOURIER

Kourier 是 Knative Serving 的轻量级 Kubernetes 原生 Ingress。Kourier 充当 Knative 的网关，将 HTTP 流量路由到 Knative 服务。

3.7.1. 为 Kourier getaways 自定义 kourier-bootstrap

Kourier 中的 Envoy 代理组件为 Knative 服务处理入站和出站 HTTP 流量。默认情况下，Kourier 在 **knative-serving-ingress** 命名空间中的 **kourier-bootstrap** 配置映射中包含一个 Envoy bootstrap 配置。您可以更改此配置。

先决条件

- 安装了 OpenShift Serverless Operator 和 Knative Serving。
- 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。

流程

- 通过更改 **KnativeServing** 自定义资源(CR)中的 **spec.ingress.kourier.bootstrap-configmap** 字段指定自定义 bootstrap 配置映射：

  KnativeServing CR 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  config:
    network:
      ingress-class: kourier.ingress.networking.knative.dev
  ingress:
    kourier:
      bootstrap-configmap: my-configmap
      enabled: true
# ...
```

3.8. 限制网络策略

3.8.1. 具有限制性网络策略的集群

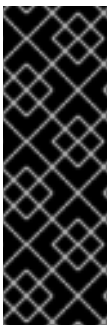
如果您使用多个用户可访问的集群，您的集群可能会使用网络策略来控制哪些 pod、服务和命名空间可以通过网络相互通信。如果您的集群使用限制性网络策略，Knative 系统 Pod 可能无法访问 Knative 应用程序。例如，如果您的命名空间具有以下网络策略（拒绝所有请求），Knative 系统 pod 无法访问您的 Knative 应用程序：

拒绝对命名空间的所有请求的 NetworkPolicy 对象示例

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
  namespace: example-namespace
spec:
  podSelector:
  ingress: []
```

3.8.2. 在具有限制性网络策略的集群中启用与 Knative 应用程序通信

要允许从 Knative 系统 pod 访问应用程序，您必须为每个 Knative 系统命名空间添加标签，然后在应用程序命名空间中创建一个 **NetworkPolicy** 对象，以便为具有此标签的其他命名空间访问命名空间。



重要

拒绝对集群中非原生服务的请求的网络策略仍阻止访问这些服务。但是，通过允许从 Knative 系统命名空间访问 Knative 应用程序，您可以从集群中的所有命名空间中访问 Knative 应用程序。

如果您不想允许从集群中的所有命名空间中访问 Knative 应用程序，您可能需要为 Knative 服务使用 *JSON Web Token 身份验证* 。Knative 服务的 JSON Web 令牌身份验证需要 Service Mesh。

先决条件

- 安装 OpenShift CLI (**oc**) 。
- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。

流程

1. 将 **knative.openshift.io/system-namespace=true** 标签添加到需要访问应用程序的每个 Knative 系统命名空间：
   a. 标记 **knative-serving** 命名空间：

   ```
   $ oc label namespace knative-serving knative.openshift.io/system-namespace=true
   ```

   b. 标记 **knative-serving-ingress** 命名空间：

   ```
   $ oc label namespace knative-serving-ingress knative.openshift.io/system-
   namespace=true
   ```

   c. 标记 **knative-eventing** 命名空间：

   ```
   $ oc label namespace knative-eventing knative.openshift.io/system-namespace=true
   ```

   d. 标记 **knative-kafka** 命名空间：

   ```
   $ oc label namespace knative-kafka knative.openshift.io/system-namespace=true
   ```

2. 在应用程序命名空间中创建一个 **NetworkPolicy** 对象，允许从带有 **knative.openshift.io/system-namespace** 标签的命名空间访问：

NetworkPolicy 对象示例

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: <network_policy_name> 1
  namespace: <namespace> 2
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          knative.openshift.io/system-namespace: "true"
  podSelector: {}
  policyTypes:
  - Ingress
```

1 为您的网络策略提供名称。

2 应用程序所在的命名空间。

第 4 章 调试无服务器应用程序

您可以使用各种方法对无服务器应用程序进行故障排除。

4.1. 检查终端输出

您可以检查您的部署命令输出，以查看部署是否成功。如果部署过程终止，您应该在输出中看到一条错误消息，该消息描述了部署失败的原因。这种故障很可能是因为清单配置错误或无效的命令。

流程

- 在您部署和管理应用程序的客户端上打开命令输出。以下示例是在 **oc apply** 命令失败后可能会看到的错误：

```
Error from server (InternalError): error when applying patch:
{"metadata":{"annotations":{"kubectl.kubernetes.io/last-applied-configuration":"
{\"apiVersion\":\"serving.knative.dev/v1\",\"kind\":\"Route\",\"metadata\":{\"annotations\":
{},\"name\":\"route-example\",\"namespace\":\"default\"},\"spec\":{\"traffic\":
[{\"configurationName\":\"configuration-example\",\"percent\":50}]}}\n"}},"spec":{"traffic":
[{"configurationName":"configuration-example","percent":50}]}}
to:
&{0xc421d98240 0xc421e77490 default route-example STDIN 0xc421db0488 264682 false}
for: "STDIN": Internal error occurred: admission webhook "webhook.knative.dev" denied the
request: mutation failed: The route must have traffic percent sum equal to 100.
ERROR: Non-zero return code '1' from command: Process exited with status 1
```

此输出显示，您必须将路由流量百分比配置为等于 100。

4.2. 检查 POD 状态

您可能需要检查 **Pod** 对象的状态来识别无服务器应用程序的问题。

流程

1. 运行以下命令，列出部署的所有 pod：

```
$ oc get pods
```

输出示例

```
NAME                                             READY     STATUS             RESTARTS   AGE
configuration-example-00001-deployment-659747ff99-9bvr4   2/2       Running            0
3h
configuration-example-00002-deployment-5f475b7849-gxcht   1/2       CrashLoopBackOff   2
36s
```

在输出中，您可以看到所选数据的所有 pod 都关于其状态。

2. 运行以下命令，查看 pod 状态的详细信息：

输出示例

```
$ oc get pod <pod_name> --output yaml
```

在输出中，**conditions** 和 **containerStatuses** 字段可能对调试特别有用。

4.3. 检查修订状态

您可能需要检查修订的状态来识别无服务器应用程序的问题。

流程

1. 如果使用 **Configuration** 对象配置路由，请运行以下命令来获取为部署创建的 **Revision** 对象的名称：

```
$ oc get configuration <configuration_name> --output jsonpath="
{.status.latestCreatedRevisionName}"
```

您可以在 **Route.yaml** 文件中找到配置名称，该文件通过定义 OpenShift **Route** 资源来指定路由设置。

如果直接使用修订配置路由，请在 **Route.yaml** 文件中查找修订名称。

2. 运行以下命令查询修订版本的状态：

```
$ oc get revision <revision-name> --output yaml
```

就绪的修订版本应具有 **reason: ServiceReady,status: "True"**，并在其状态中 **type: Ready** 条件。如果存在这些条件，您可能需要检查 pod 状态或 Istio 路由。否则，资源状态包含错误消息。

4.3.1. 其他资源

- 路由配置

4.4. 检查 INGRESS 状态

您可能需要检查 Ingress 的状态来识别无服务器应用程序的问题。

流程

- 运行以下命令，检查 Ingress 的 IP 地址：

```
$ oc get svc -n istio-system istio-ingressgateway
```

istio-ingressgateway 服务是 Knative 使用的 **LoadBalancer** 服务。

如果没有外部 IP 地址，请运行以下命令：

```
$ oc describe svc istio-ingressgateway -n istio-system
```

此命令打印未调配 IP 地址的原因。最有可能是因为配额问题造成的。

4.5. 检查路由状态

在某些情况下，**Route** 对象存在问题。您可以使用 OpenShift CLI (**oc**)检查其状态。

流程

- 运行以下命令，查看您部署应用程序的 **Route** 对象的状态：

```
$ oc get route <route_name> --output yaml
```

将 **<route_name** > 替换为您的 **Route** 对象的名称。

status 对象中的 **conditions** 对象指出故障时的原因。

4.6. 检查 INGRESS 和 ISTIO 路由

有时，当 Istio 用作 Ingress 层时，Ingress 和 Istio 路由会有问题。您可以使用 OpenShift CLI (**oc**)查看它们的详情。

流程

1. 运行以下命令列出所有 Ingress 资源及其对应的标签：

```
$ oc get ingresses.networking.internal.knative.dev -o=custom-
columns='NAME:.metadata.name,LABELS:.metadata.labels'
```

输出示例

```
NAME           LABELS
helloworld-go   map[serving.knative.dev/route:helloworld-go
serving.knative.dev/routeNamespace:default serving.knative.dev/service:helloworld-go]
```

在这个输出中，label **serving.knative.dev/route** 和 **service.knative.dev/routeNamespace** 表示 Ingress 资源所在的 **Route**。您的 **Route** 和 Ingress 应该被列出。

如果您的 Ingress 不存在，路由控制器会假定 **Route** 或 **Service** 对象的目标 **Revision** 对象未就绪。继续其他调试程序，以诊断 **修订** 就绪状态。

2. 如果列出了 Ingress，请运行以下命令检查为路由创建的 **ClusterIngress** 对象：

```
$ oc get ingresses.networking.internal.knative.dev <ingress_name> --output yaml
```

在输出的 status 部分中，如果 **type=Ready** 条件的状态为 **True**，则 Ingress 可以正常工作。否则，输出中会包含错误消息。

3. 如果 Ingress 的状态为 **Ready**，则有一个对应的 **VirtualService** 对象。运行以下命令，验证 **VirtualService** 对象的配置：

```
$ oc get virtualservice -l networking.internal.knative.dev/ingress=<ingress_name> -n
<ingress_namespace> --output yaml
```

VirtualService 对象中的网络配置必须与 **Ingress** 和 **Route** 对象匹配。因为 **VirtualService** 对象没有公开 **Status** 字段，您可能需要等待其设置传播。

4.6.1. 其他资源

- maistra Service Mesh 文档

第 5 章 KOURIER 和 ISTIO INGRESSES

OpenShift Serverless 支持以下两个入口解决方案：

- Kourier
- Istio 使用 Red Hat OpenShift Service Mesh

默认为 Kourier。

5.1. KOURIER 和 ISTIO INGRESS 解决方案

5.1.1. Kourier

Kourier 是 OpenShift Serverless 的默认入口解决方案。它有以下属性：

- 它基于 envoy 代理。
- 它很简单且轻量级。
- 它提供 Serverless 需要提供其一组功能的基本路由功能。
- 它支持基本可观察性和指标。
- 它支持 Knative Service 路由的基本 TLS 终止。
- 它仅提供有限的配置和扩展选项。

5.1.2. 使用 OpenShift Service Mesh 的 Istio

使用 Istio 作为 OpenShift Serverless 的 ingress 解决方案启用一个额外的功能集，它基于 Red Hat OpenShift Service Mesh 提供的内容：

- 所有连接间的原生 mTLS
- Serverless 组件是服务网格的一部分
- 额外的可观察性和指标
- 授权和身份验证支持
- 自定义规则和配置，由 Red Hat OpenShift Service Mesh 支持

但是，额外的功能会带来更高的开销和资源消耗。详情请参阅 Red Hat OpenShift Service Mesh 文档。

有关 Istio 要求和安装说明，请参阅 Serverless 文档中的"集成 Service Mesh with OpenShift Serverless"部分。

5.1.3. 流量配置和路由

无论您使用 Kourier 或 Istio，Knative Service 的流量都会通过 **net-kourier-controller** 或 **net-istio-controller** 在 **knative-serving** 命名空间中进行配置。

控制器读取 **KnativeService** 及其子自定义资源来配置入口解决方案。两个入口解决方案都提供一个作为流量路径一部分的入口网关 pod。两个入口解决方案都基于 Envoy。默认情况下，Serverless 对每个 **KnativeService** 对象有两个路由：

- 由 OpenShift 路由器转发的 **cluster-external 路由**，如 **myapp-namespace.example.com**。
- 包含集群域的 **cluster-local 路由**，如 **myapp.namespace.svc.cluster.local**。这个域可以用来从 Knative 或其他用户工作负载调用 Knative 服务。

ingress 网关可以在服务模式或代理模式中转发请求：

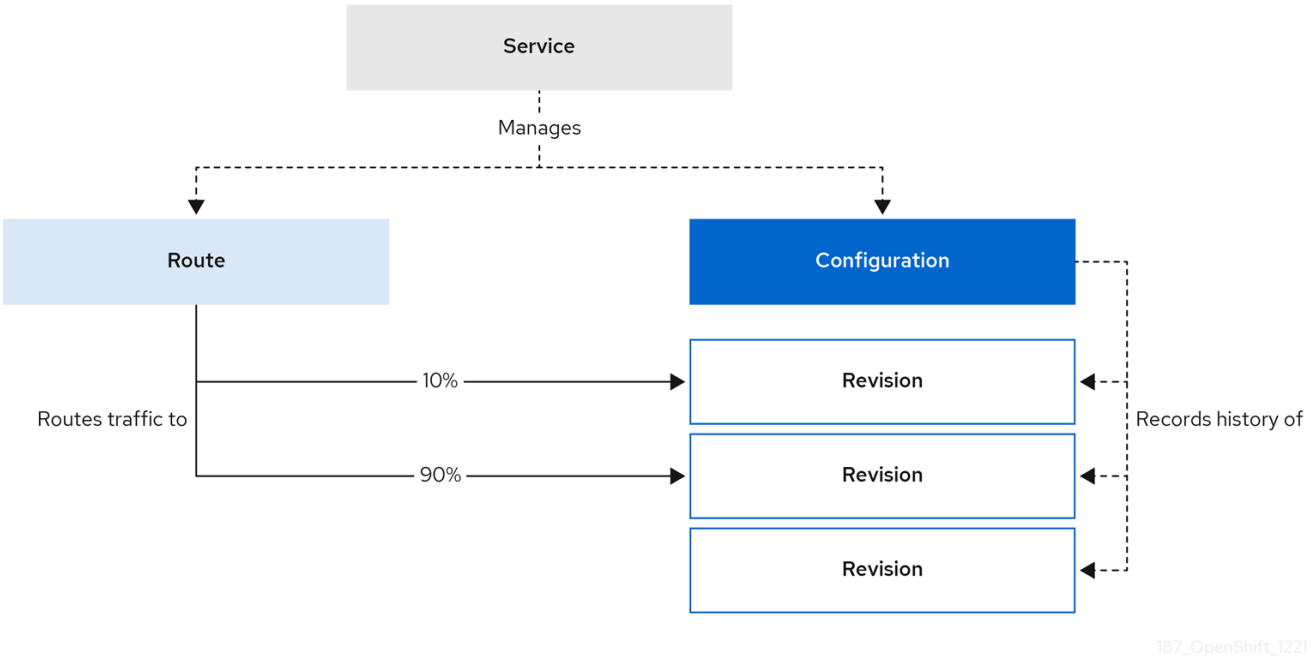
- 在服务模式中，请求直接进入 Knative 服务的 Queue-Proxy sidecar 容器。
- 在代理模式中，请求首先通过 **knative-serving** 命名空间中的 Activator 组件。

选择模式取决于 Knative、Knative 服务和当前流量的配置。例如，如果 Knative Service 缩减为零，则请求将发送到 Activator 组件，该组件充当缓冲区，直到新的 Knative 服务 pod 启动为止。

第 6 章 流量分割

6.1. 流量分割概述

在 Knative 应用程序中，可以通过创建流量分割来管理流量。流量分割被配置为由 Knative 服务管理的路由的一部分。



配置路由允许将请求发送到服务的不同修订版本。此路由由 **Service** 对象的 **traffic** spec 决定。

traffic 规格声明由一个或多个修订版本组成，每个修订版本负责处理整个流量的一部分。路由到每个修订版本的流量百分比必须添加到 100%，由 Knative 验证确保。

traffic 规格中指定的修订版本可以是固定的、名为修订的修订版本，或者可以指向"latest"修订，该修订跟踪服务所有修订版本列表的头。"latest" 修订版本是一个浮动引用类型，它在创建了新修订版本时更新。每个修订版本都可以附加标签，为该修订版本创建一个额外访问 URL。

traffic 规格可通过以下方法修改：

- 直接编辑 **Service** 对象的 YAML。
- 使用 Knative (**kn**) CLI **--traffic** 标志。
- 使用 OpenShift Container Platform Web 控制台。

当您创建 Knative 服务时，它没有任何默认 **traffic** spec 设置。

6.2. TRAFFIC 规格示例

以下示例显示了一个 **traffic** 规格，其中 100% 的流量路由到该服务的最新修订版本。在 **status** 下，您可以看到 **latestRevision** 解析为的最新修订版本的名称：

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
```

```
  name: example-service
  namespace: default
spec:
...
  traffic:
  - latestRevision: true
    percent: 100
status:
  ...
  traffic:
  - percent: 100
    revisionName: example-service
```

以下示例显示了一个 **traffic** 规格，其中 100% 的流量路由到当前标记为 **current** 修订版本，并且该修订版本的名称指定为 **example-service**。标记为 **latest** 的修订版本会保持可用，即使没有流量路由到它：

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
...
  traffic:
  - tag: current
    revisionName: example-service
    percent: 100
  - tag: latest
    latestRevision: true
    percent: 0
```

以下示例演示了如何扩展 **traffic** 规格中的修订版本列表，以便在多个修订版本间分割流量。这个示例将 50% 的流量发送到标记为 **current** 修订版本，50% 的流量发送到标记为 **candidate** 的修订版本。标记为 **latest** 的修订版本会保持可用，即使没有流量路由到它：

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
...
  traffic:
  - tag: current
    revisionName: example-service-1
    percent: 50
  - tag: candidate
    revisionName: example-service-2
    percent: 50
  - tag: latest
    latestRevision: true
    percent: 0
```

6.3. 使用 KNATIVE CLI 进行流量分割

使用 Knative (**kn**) CLI 创建流量分割功能，通过直接修改 YAML 文件，提供更精简且直观的用户界面。您可以使用 **kn service update** 命令在服务修订版本间分割流量。

6.3.1. 使用 Knative CLI 创建流量分割

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 已安装 Knative (**kn**) CLI。
- 您已创建了 Knative 服务。

流程

- 使用带有标准 **kn service update** 命令的 **--traffic** 标签指定服务修订版本以及您要路由到它的流量百分比：

  示例命令

  ```
  $ kn service update <service_name> --traffic <revision>=<percentage>
  ```

  其中：

  - **<service_name>** 是您要为其配置流量路由的 Knative 服务的名称。
  - **<revision>** 是您要配置为接收流量百分比的修订版本。您可以使用 **--tag** 标志指定修订版本的名称，或指定分配给修订版本的标签。
  - **<percentage>** 是您要发送到指定修订版本的流量百分比。

- 可选： **--traffic** 标志可在一个命令中多次指定。例如，如果您有一个标记为 **@latest** 的修订版本以及名为 **stable** 的修订版本，您可以指定您要分割到每个修订版本的流量百分比：

  示例命令

  ```
  $ kn service update showcase --traffic @latest=20,stable=80
  ```

  如果您有多个修订版本，且没有指定应分割到最后一个修订版本的流量百分比，**--traffic** 标志可以自动计算此设置。例如，如果您有一个第三个版本名为 **example**，则使用以下命令：

  示例命令

  ```
  $ kn service update showcase --traffic @latest=10,stable=60
  ```

  剩余的 30% 的流量被分成 **example** 修订，即使未指定。

6.4. 用于流量分割的 CLI 标志

Knative (**kn**) CLI 支持作为 **kn service update** 命令的一部分对服务的流量块进行流量操作。

6.4.1. Knative CLI 流量分割标志

下表显示流量分割标志、值格式和标志执行的操作汇总。Repetition 列表示在 **kn service update** 命令中是否允许重复标志的特定值。

标记	值	操作	重复
--traffic	**RevisionName=Percent**	为 **RevisionName** 提供 **Percent** 的流量	是
--traffic	**Tag=Percent**	为带有 **Tag**的修订版本提供 **Percent** 的流量	是
--traffic	**@latest=Percent**	为最新可用的修订版本提供 **Percent** 的流量	否
--tag	**RevisionName=Tag**	为 **RevisionName** 提供 **Tag**	是
--tag	**@latest=Tag**	为最新可用的修订版本提供 **Tag**	否
--untag	**Tag**	从修订中删除 **Tag**	是

6.4.1.1. 多个标志和顺序优先级

所有流量相关标志均可使用单一 **kn service update** 命令指定。**kn** 定义这些标志的优先级。不考虑使用命令时指定的标志顺序。

通过 **kn** 评估标志时，标志的优先级如下：

1. **--untag**：带有此标志的所有引用修订版本均将从流量块中移除。
2. **--tag**：修订版本将按照流量块中的指定进行标记。
3. **--traffic**：为引用的修订版本分配一部分流量分割。

您可以将标签添加到修订版本，然后根据您设置的标签来分割流量。

6.4.1.2. 修订版本的自定义 URL

使用 **kn service update** 命令为服务分配 **--tag** 标志，可为在更新服务时创建的修订版本创建一个自定义 URL。自定义 URL 遵循 **https://<tag>-<service_name>-<namespace>.<domain>** 或 **http://<tag>-<service_name>-<namespace>.<domain>**。

--tag 和 **--untag** 标志使用以下语法：

- 需要一个值。
- 在服务的流量块中表示唯一标签。
- 在一个命令中可多次指定.

6.4.1.2.1. 示例：将标签分配给修订版本

以下示例将标签 **latest** 分配给名为 **example-revision** 的修订版本：

```
$ kn service update <service_name> --tag @latest=example-tag
```

6.4.1.2.2. 示例：从修订中删除标签

您可以使用 **--untag** 标志来删除自定义 URL。

注意

如果修订版本删除了其标签，并分配了流量的 0%，则修订版本将完全从流量块中删除。

以下命令从名为 **example-revision** 的修订版本中删除所有标签：

```
$ kn service update <service_name> --untag example-tag
```

6.5. 在修订版本间分割流量

创建无服务器应用程序后，应用程序会在 OpenShift Container Platform Web 控制台中的 **Developer 视角** 的 **Topology 视图**中显示。应用程序修订版本由节点表示，Knative 服务由节点的四边形表示。

代码或服务配置中的任何新更改都会创建一个新修订版本，也就是给定时间点上代码的快照。对于服务，您可以根据需要通过分割服务修订版本并将其路由到不同的修订版本来管理服务间的流量。

6.5.1. 使用 OpenShift Container Platform Web 控制台管理修订版本之间的流量

先决条件

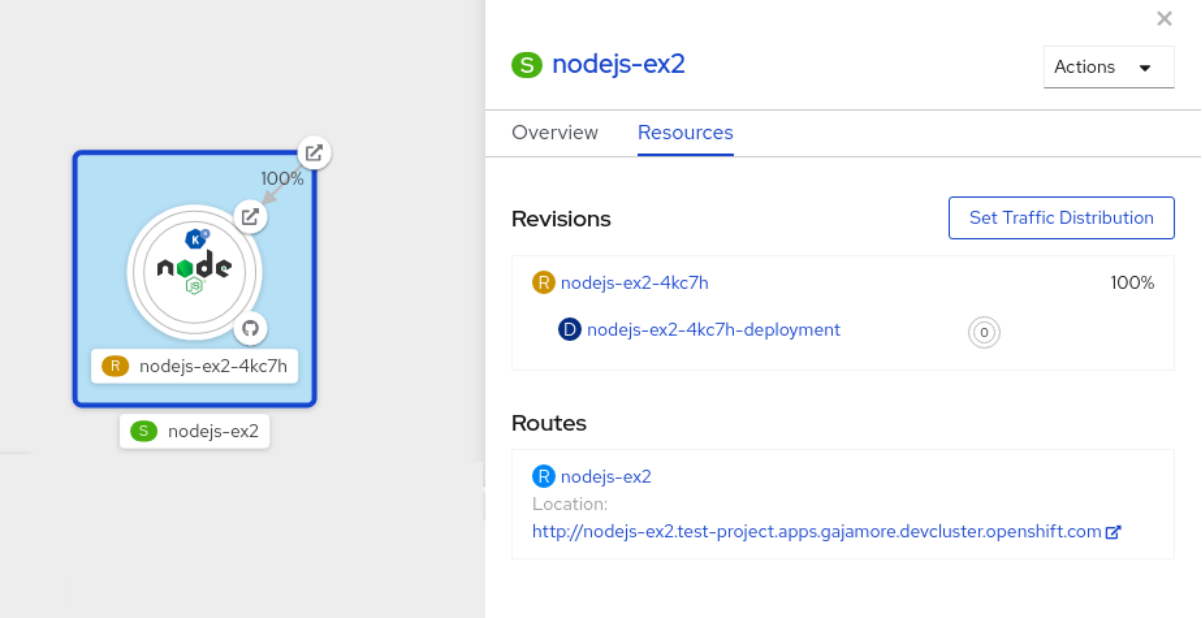
- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 已登陆到 OpenShift Container Platform Web 控制台。

流程

要在 **Topology** 视图中的多个应用程序修订版本间分割流量：

1. 点 Knative 服务在侧面面板中查看其概述信息。
2. 点 **Resources** 选项卡，查看服务的 **Revisions** 和 **Routes** 列表。

图 6.1. 无服务器应用程序



3. 点侧边面板顶部的由 S 图标代表的服务，查看服务详情概述。

4. 点 **YAML** 选项卡，在 YAML 编辑器中修改服务配置，然后点 **Save**。例如，将 **timeoutseconds** 从 300 改为 301。这个配置更改会触发新修订版本。在 **Topology** 视图中会显示最新的修订，服务 **Resources** 选项卡现在会显示两个修订版本。

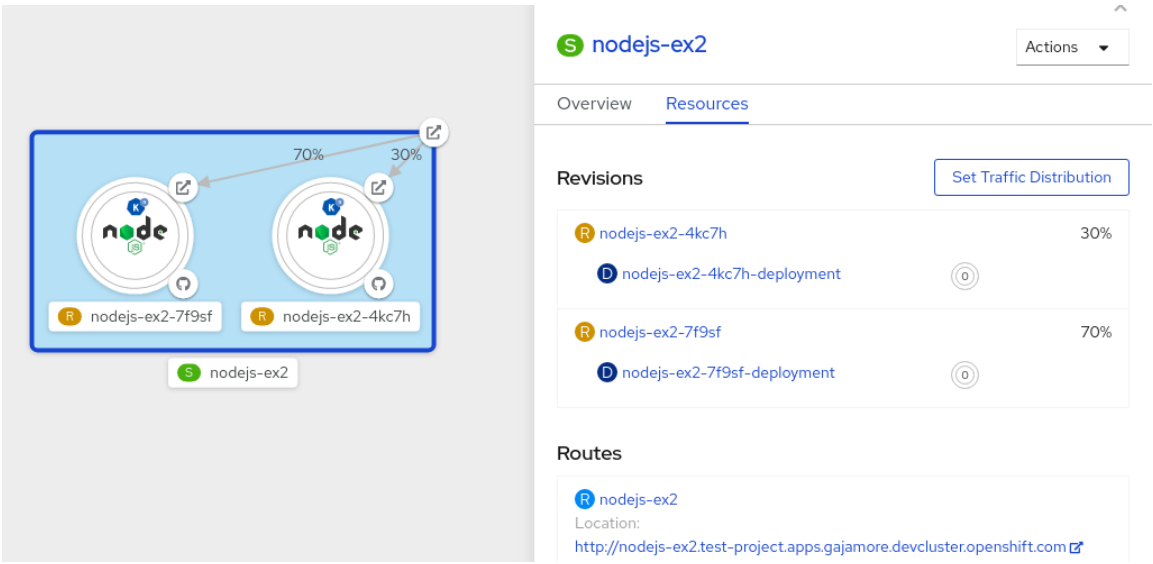
5. 在 **Resources** 选项卡中，点 **Set Traffic Distribution** 查看流量分布对话框：

   a. 在 **Splits** 字段中为两个修订版本添加流量百分比。

   b. 添加标签以便为这两个修订版本创建自定义 URL。

   c. 点 **Save** 查看两个节点，分别代表 Topology 视图中的两个修订版本。

   图 6.2. 无服务器应用程序修订



6.6. 使用蓝绿策略重新路由流量

您可以使用蓝绿部署策略，安全地将流量从应用的生产版本重新路由到新版本。

6.6.1. 使用蓝绿部署策略路由和管理流量

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 安装 OpenShift CLI (**oc**) 。

流程

1. 创建并部署应用程序作为 Knative 服务。
2. 通过查看以下命令的输出，查找部署服务时创建的第一个修订版本的名称：

```
$ oc get ksvc <service_name> -o=jsonpath='{.status.latestCreatedRevisionName}'
```

示例命令

```
$ oc get ksvc showcase -o=jsonpath='{.status.latestCreatedRevisionName}'
```

输出示例

```
$ showcase-00001
```

3. 在服务 **spec** 中添加以下 YAML 以将入站流量发送到修订版本：

```
...
spec:
  traffic:
    - revisionName: <first_revision_name>
      percent: 100 # All traffic goes to this revision
...
```

4. 验证您可以在 URL 输出中运行以下命令来查看您的应用程序：

```
$ oc get ksvc <service_name>
```

5. 通过修改服务的 **template** 规格中至少有一个字段来部署应用程序的第二个修订版本。例如，您可以修改服务的 **image** 或 **env** 环境变量。您可以通过应用服务 YAML 文件重新部署服务，如果安装了 Knative (**kn**) CLI，也可以使用 **kn service update** 命令。
6. 运行以下命令，查找您在重新部署服务时创建的第二个最新的修订版本的名称：

```
$ oc get ksvc <service_name> -o=jsonpath='{.status.latestCreatedRevisionName}'
```

此时，服务的第一个和第二个修订版本都已部署并运行。

7. 更新您的现有服务，以便为第二个修订版本创建新的测试端点，同时仍然将所有其他流量发送到第一个修订版本：

使用测试端点更新的服务 spec 示例

```
...
```

```
spec:
  traffic:
    - revisionName: <first_revision_name>
      percent: 100 # All traffic is still being routed to the first revision
    - revisionName: <second_revision_name>
      percent: 0 # No traffic is routed to the second revision
      tag: v2 # A named route
...
```

在通过重新应用 YAML 资源重新部署此服务后，应用的第二个修订现已被暂存。没有流量路由到主 URL 的第二个修订版本，Knative 会创建一个名为 **v2** 的新服务来测试新部署的修订版本。

8. 运行以下命令，获取第二个修订版本的新服务的 URL：

```
$ oc get ksvc <service_name> --output jsonpath="{.status.traffic[*].url}"
```

在将任何流量路由到之前，您可以使用此 URL 验证新版本的应用运行正常。

9. 再次更新您的现有服务，以便 50% 的流量发送到第一个修订版本，50% 发送到第二个修订版本：

更新的服务 spec 在修订版本间分割流量 50/50 的示例

```
...
spec:
  traffic:
    - revisionName: <first_revision_name>
      percent: 50
    - revisionName: <second_revision_name>
      percent: 50
      tag: v2
...
```

10. 当您准备好将所有流量路由到应用程序的新版本时，请再次更新该服务，将 100% 的流量发送到第二个修订版本：

更新的服务 spec 将所有流量发送到第二个修订版本的示例

```
...
spec:
  traffic:
    - revisionName: <first_revision_name>
      percent: 0
    - revisionName: <second_revision_name>
      percent: 100
      tag: v2
...
```

提示

如果您不计划回滚修订版本，您可以删除第一个修订版本，而不是将其设置为流量的 0%。然后，不可路由的修订版本对象会被垃圾回收。

11. 访问第一个修订版本的 URL，以验证没有更多流量发送到应用程序的旧版本。

第 7 章 EXTERNAL 和 INGRESS 路由

7.1. 路由概述

Knative 利用 OpenShift Container Platform TLS 终止来为 Knative 服务提供路由。创建 Knative 服务时，会自动为该服务创建一个 OpenShift Container Platform 路由。此路由由 OpenShift Serverless Operator 管理。OpenShift Container Platform 路由通过与 OpenShift Container Platform 集群相同的域公开 Knative 服务。

您可以禁用 OpenShift Container Platform 路由的 Operator 控制，以便您可以配置 Knative 路由来直接使用 TLS 证书。

Knative 路由也可以与 OpenShift Container Platform 路由一起使用，以提供额外的精细路由功能，如流量分割。

7.1.1. OpenShift Container Platform 的其他资源

- 特定于路由的注解

7.2. 自定义标签和注解

OpenShift Container Platform 路由支持使用自定义标签和注解，您可以通过修改 Knative 服务的**元数据规格来配置这些标签和注解**。自定义标签和注解从服务传播到 Knative 路由，然后传播到 Knative ingress，最后传播到 OpenShift Container Platform 路由。

7.2.1. 为 OpenShift Container Platform 路由自定义标签和注解

先决条件

- 您必须已在 OpenShift Container Platform 集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 安装 OpenShift CLI (**oc**) 。

流程

1. 创建包含您要传播到 OpenShift Container Platform 路由的标签或注解的 Knative 服务：
   - 使用 YAML 创建服务：

     使用 YAML 创建的服务示例

     ```
     apiVersion: serving.knative.dev/v1
     kind: Service
     metadata:
       name: <service_name>
       labels:
         <label_name>: <label_value>
       annotations:
         <annotation_name>: <annotation_value>
     ...
     ```

   - 要使用 Knative (**kn**) CLI 创建服务，请输入：

使用 kn 命令创建的服务示例

```
$ kn service create <service_name> \
  --image=<image> \
  --annotation <annotation_name>=<annotation_value> \
  --label <label_value>=<label_value>
```

2. 通过检查以下命令的输出来验证 OpenShift Container Platform 路由是否已使用您添加的注解或标签创建：

验证命令示例

```
$ oc get routes.route.openshift.io \
    -l serving.knative.openshift.io/ingressName=<service_name> \ 1
    -l serving.knative.openshift.io/ingressNamespace=<service_namespace> \ 2
    -n knative-serving-ingress -o yaml \
       | grep -e "<label_name>: \"<label_value>\""  -e "<annotation_name>:
<annotation_value>" 3
```

1. 使用服务的名称。
2. 使用创建服务的命名空间。
3. 将您的值用于标签和注解名称和值。

7.3. 为 KNATIVE 服务配置路由

如果要将 Knative 服务配置为在 OpenShift Container Platform 上使用 TLS 证书，则必须禁用 OpenShift Serverless Operator 为服务自动创建路由，而是手动为服务创建路由。

注意

完成以下步骤时，不会创建 **knative-serving-ingress** 命名空间中的默认 OpenShift Container Platform 路由。但是，应用程序的 Knative 路由仍然在此命名空间中创建。

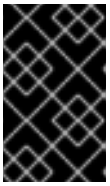
7.3.1. 为 Knative 服务配置 OpenShift Container Platform 路由

先决条件

- OpenShift Serverless Operator 和 Knative Serving 组件必须安装在 OpenShift Container Platform 集群中。
- 安装 OpenShift CLI (**oc**) 。

流程

1. 创建包含 **service.knative.openshift.io/disableRoute=true** 注解的 Knative 服务：



重要

service.knative.openshift.io/disableRoute=true 注解指示 OpenShift Serverless 不自动为您创建路由。但是，该服务仍然会显示 URL 并达到 **Ready** 状态。除非使用与 URL 中主机名相同的主机名创建自己的路由，此 URL 才能在外部工作。

a. 创建 Knative **Service** 资源：

资源示例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    serving.knative.openshift.io/disableRoute: "true"
spec:
  template:
    spec:
      containers:
      - image: <image>
...
```

b. 应用 **Service** 资源：

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

c. 可选。使用 **kn service create** 命令创建 Knative 服务：

kn 命令示例

```
$ kn service create <service_name> \
  --image=gcr.io/knative-samples/helloworld-go \
  --annotation serving.knative.openshift.io/disableRoute=true
```

2. 验证没有为服务创建 OpenShift Container Platform 路由：

示例命令

```
$ $ oc get routes.route.openshift.io \
  -l serving.knative.openshift.io/ingressName=$KSERVICE_NAME \
  -l serving.knative.openshift.io/ingressNamespace=$KSERVICE_NAMESPACE \
  -n knative-serving-ingress
```

您将看到以下输出：

```
No resources found in knative-serving-ingress namespace.
```

3. 在 **knative-serving-ingress** 命名空间中创建 **Route** 资源：

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
```

```
  annotations:
    haproxy.router.openshift.io/timeout: 600s 1
  name: <route_name> 2
  namespace: knative-serving-ingress 3
spec:
  host: <service_host> 4
  port:
    targetPort: http2
  to:
    kind: Service
    name: kourier
    weight: 100
  tls:
    insecureEdgeTerminationPolicy: Allow
    termination: edge 5
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE----
  wildcardPolicy: None
```

1. OpenShift Container Platform 路由的超时值。您必须设置与 **max-revision-timeout-seconds** 设置相同的值（默认为 **600s**）。
2. OpenShift Container Platform 路由的名称。
3. OpenShift Container Platform 路由的命名空间。这必须是 **knative-serving-ingress**。
4. 用于外部访问的主机名。您可以将其设置为 **<service_name>-<service_namespace>.<domain>**。
5. 您要使用的证书。目前，只支持 边缘（**edge**） 终止。

4. 应用 **Route** 资源：

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

7.4. 外部路由的 URL 方案

用于增强安全性，外部路由的 URL 方案默认为 HTTPS。这个方案由 **KnativeServing** 自定义资源 (CR) spec 中的 **default-external-scheme** 键决定。

7.4.1. 为外部路由设置 URL 方案

默认规格

```
...
spec:
  config:
    network:
      default-external-scheme: "https"
...
```

您可以通过修改 **default-external-scheme** 键来覆盖默认的 spec 以使用 HTTP：

HTTP 覆盖规格

```
...
spec:
  config:
    network:
      default-external-scheme: "http"
...
```

7.5. 集群本地可用性

默认情况下，Knative 服务会发布到一个公共 IP 地址。被发布到一个公共 IP 地址意味着 Knative 服务是公共应用程序，并有一个公开访问的 URL。

可以从集群以外访问公开的 URL。但是，开发人员可能需要构建后端服务，这些服务只能从集群内部访问（称为 *私有服务*）。开发人员可以使用 **networking.knative.dev/visibility=cluster-local** 标签标记集群中的各个服务，使其私有。

重要

对于 OpenShift Serverless 1.15.0 及更新的版本，**service.knative.dev/visibility** 标签不再可用。您必须更新现有服务来改用 **networking.knative.dev/visibility** 标签。

7.5.1. 将集群可用性设置为集群本地

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 您已创建了 Knative 服务。

流程

- 通过添加 **networking.knative.dev/visibility=cluster-local** 标签来设置服务的可见性：

```
$ oc label ksvc <service_name> networking.knative.dev/visibility=cluster-local
```

验证

- 输入以下命令并查看输出结果，检查您的服务的 URL 是否现在格式为 **http://<service_name>.<namespace>.svc.cluster.local**：

```
$ oc get ksvc
```

输出示例

```
NAME          URL                                        LATESTCREATED
LATESTREADY      READY   REASON
hello         http://hello.default.svc.cluster.local     hello-tx2g7      hello-
tx2g7      True
```

7.5.2. 为集群本地服务启用 TLS 身份验证

对于集群本地服务，使用 Kourier 本地网关 **kourier-internal**。如果要针对 Kourier 本地网关使用 TLS 流量，则必须在本地网关中配置您自己的服务器证书。

先决条件

- 安装了 OpenShift Serverless Operator 和 Knative Serving。
- 有管理员权限。
- 已安装 OpenShift (**oc**) CLI。

流程

1. 在 **knative-serving-ingress** 命名空间中部署服务器证书：

```
$ export san="knative"
```

注意

需要主题备用名称(SAN)验证，以便这些证书能够向 **<app_name>.<namespace>.svc.cluster.local** 提供请求。

2. 生成 root 密钥和证书：

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

3. 生成使用 SAN 验证的服务器密钥：

```
$ openssl req -out tls.csr -newkey rsa:2048 -nodes -keyout tls.key \
  -subj "/CN=Example/O=Example" \
  -addext "subjectAltName = DNS:$san"
```

4. 创建服务器证书：

```
$ openssl x509 -req -extfile <(printf "subjectAltName=DNS:$san") \
  -days 365 -in tls.csr \
  -CA ca.crt -CAkey ca.key -CAcreateserial -out tls.crt
```

5. 为 Kourier 本地网关配置 secret：

a. 从前面的步骤创建的证书，在 **knative-serving-ingress** 命名空间中部署 secret：

```
$ oc create -n knative-serving-ingress secret tls server-certs \
    --key=tls.key \
    --cert=tls.crt --dry-run=client -o yaml | oc apply -f -
```

b. 更新 **KnativeServing** 自定义资源 (CR) spec，以使用 Kourier 网关创建的 secret：

KnativeServing CR 示例

```
...
spec:
  config:
    kourier:
      cluster-cert-secret: server-certs
...
```

Kourier 控制器在不重启该服务的情况下设置证书，因此您不需要重启 pod。

您可以通过端口 **443** 访问 Kourier 内部服务，方法是从客户端挂载并使用 **ca.crt**。

7.6. KOURIER 网关服务类型

Kourier 网关默认作为 **ClusterIP** 服务类型公开。此服务类型由 **KnativeServing** 自定义资源 (CR) 中的 **service-type** ingress spec 决定。

默认规格

```
...
spec:
  ingress:
    kourier:
      service-type: ClusterIP
...
```

7.6.1. 设置 Kourier 网关服务类型

您可以通过修改 **service-type** spec 来覆盖默认服务类型来使用负载均衡器服务类型：

LoadBalancer 覆盖规格

```
...
spec:
  ingress:
    kourier:
      service-type: LoadBalancer
...
```

7.7. 使用 HTTP2 和 GRPC

OpenShift Serverless 只支持不安全或边缘终端路由。不安全或边缘终端路由不支持 OpenShift Container Platform 中的 HTTP2。这些路由也不支持 gRPC，因为 gRPC 由 HTTP2 传输。如果您在应用程序中使用

这些协议，则必须使用入口（ingress）网关直接调用应用程序。要做到这一点，您必须找到 ingress 网关的公共地址以及应用程序的特定主机。

7.7.1. 使用 HTTP2 和 gRPC 与无服务器应用程序交互

重要

此方法适用于 OpenShift Container Platform 4.10 及更新的版本。有关旧版本，请参阅以下部分。

先决条件

- 在集群上安装 OpenShift Serverless Operator 和 Knative Serving。
- 安装 OpenShift CLI (**oc**) 。
- 创建 Knative 服务。
- 升级 OpenShift Container Platform 4.10 或更高版本。
- 在 OpenShift Ingress 控制器中启用 HTTP/2。

流程

1. 将 **serverless.openshift.io/default-enable-http2=true** 注解添加到 **KnativeServing** 自定义资源中：

```
$ oc annotate knativeserving <your_knative_CR> -n knative-serving
serverless.openshift.io/default-enable-http2=true
```

2. 添加注解后，您可以验证 Kourier 服务的 **appProtocol** 值是否为 **h2c** ：

```
$ oc get svc -n knative-serving-ingress kourier -o jsonpath="{.spec.ports[0].appProtocol}"
```

输出示例

```
h2c
```

3. 现在，您可以对外部流量使用 HTTP/2 协议的 gRPC 框架，例如：

```
import "google.golang.org/grpc"

grpc.Dial(
  YOUR_URL, 1
  grpc.WithTransportCredentials(insecure.NewCredentials())), 2
)
```

1 您的 **ksvc** URL。

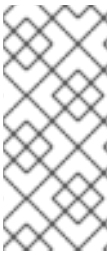
2 您的证书。

其他资源

- 启用 HTTP/2 入口连接

7.8. 使用带有 OPENSHIFT INGRESS 分片的 SERVING

您可以将 Knative Serving 与 OpenShift ingress 分片搭配使用，以根据域分割入口流量。这可让您更有效地管理和将网络流量路由到集群的不同部分。



注意

即使使用 OpenShift ingress 分片，OpenShift Serverless 流量仍然通过单个 Knative Ingress 网关和 **knative-serving** 项目中的 activator 组件路由。

有关隔离网络流量的更多信息，请参阅使用 Service Mesh 与 OpenShift Serverless 隔离网络流量。

先决条件

- 安装了 OpenShift Serverless Operator 和 Knative Serving。
- 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。

7.8.1. 配置 OpenShift ingress 分片

在配置 Knative Serving 前，您必须配置 OpenShift ingress 分片。

流程

- 使用 **IngressController** CR 中的标签选择器配置 OpenShift Serverless，以匹配具有不同域的特定入口分片：

  IngressController CR 示例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: ingress-dev 1
  namespace: openshift-ingress-operator
spec:
  routeSelector:
    matchLabels:
      router: dev 2
  domain: "dev.serverless.cluster.example.com" 3
  # ...
---
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: ingress-prod 4
  namespace: openshift-ingress-operator
spec:
  routeSelector:
    matchLabels:
```

```
    router: prod 5
domain: "prod.serverless.cluster.example.com" 6
# ...
```

1 第一个入口分片的名称。

2 与 **ingress-dev** 分片匹配的标签选择器。

3 **ingress-dev** 分片的自定义域。

4 第二个入口分片的名称。

5 与 **ingress-prod** 分片匹配的标签选择器。

6 **ingress-prod** 分片的自定义域。

7.8.2. 在 KnativeServing CR 中配置自定义域

配置 OpenShift ingress 分片后，您必须配置 Knative Serving 以匹配它们。

流程

- 在 **KnativeServing** CR 中，通过添加 **spec.config.domain** 字段将 Serving 配置为使用与入口分片相同的域和标签：

  KnativeServing CR 示例

```
spec:
  config:
    domain: 1
      dev.serverless.cluster.example.com: |
        selector:
          router: dev
      prod.serverless.cluster.example.com: |
        selector:
          router: prod
# ...
```

1 这些值需要与 ingress 分片配置中的值匹配。

7.8.3. 以 Knative Service 中的特定入口分片为目标

配置入口分片和 Knative Serving 后，您可以使用标签将 Knative Service 资源中的特定入口分片作为目标。

流程

- 在 **Service** CR 中，添加与特定分片匹配的标签选择器：

  Service CR 示例

```
apiVersion: serving.knative.dev/v1
```

```
kind: Service
metadata:
  name: hello-dev
  labels:
    router: dev (1)
spec:
  template:
    spec:
      containers:
      - image: docker.io/openshift/hello-openshift
---
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: hello-prod
  labels:
    router: prod (2)
spec:
  template:
    spec:
      containers:
      - image: docker.io/openshift/hello-openshift
  # ...
```

(1) (2) 标签必须与 **KnativeServing** CR 中的配置匹配。

7.8.4. 使用 OpenShift ingress 分片配置验证 Serving

配置入口分片、Knative Serving 和服务后，您可以验证您的服务是否使用正确的路由和所选入口分片。

流程

1. 运行以下命令输出集群中服务的信息：

```
$ oc get ksvc
```

输出示例

```
NAME        URL                                             LATESTCREATED
LATESTREADY       READY   REASON
hello-dev   https://hello-dev-default.dev.serverless.cluster.example.com    hello-dev-00001
hello-dev-00001   True
hello-prod  https://hello-prod-default.prod.serverless.cluster.example.com  hello-prod-00001
hello-prod-00001  True
```

2. 运行以下命令，验证您的服务是否使用正确的路由和所选入口分片：

```
$ oc get route -n knative-serving-ingress -o jsonpath='{range .items[*]}{@.metadata.name}{"
"}{@.spec.host}{" "}{@.status.ingress[*].routerName}{"\n"}{end}'
```

输出示例

```
route-19e6628b-77af-4da0-9b4c-1224934b2250-323461616533 hello-prod-
default.prod.serverless.cluster.example.com ingress-prod
route-cb5085d9-b7da-4741-9a56-96c88c6adaaa-373065343266 hello-dev-
default.dev.serverless.cluster.example.com ingress-dev
```

第 8 章 HTTP 配置

8.1. 全局 HTTPS 重定向

HTTPS 重定向为传入的 HTTP 请求提供重定向。这些重定向的 HTTP 请求会被加密。您可以通过为 **KnativeServing** 自定义资源 (CR) 配置 **httpProtocol** spec，为集群中的所有服务启用 HTTPS 重定向。

8.1.1. HTTPS 重定向全局设置

启用 HTTPS 重定向的 KnativeServing CR 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    network:
      httpProtocol: "redirected"
...
```

8.2. 每个服务的 HTTPS 重定向

您可以通过配置 **networking.knative.dev/http-option** 注解来为服务启用或禁用 HTTPS 重定向。

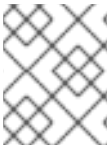
8.2.1. 为服务重定向 HTTPS

以下示例演示了如何在 Knative **Service** YAML 对象中使用此注解：

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example
  namespace: default
  annotations:
    networking.knative.dev/http-protocol: "redirected"
spec:
  ...
```

8.3. 对 HTTP/1 的完整 DUPLEX 支持

您可以通过配置 **features.knative.dev/http-full-duplex** 注解来启用对服务的 HTTP/1 完整的 duplex 支持。



注意

在启用前验证您的 HTTP 客户端，因为较早的版本客户端可能不提供对 HTTP/1 完整双工支持。

以下示例演示了如何在修订 spec 级别的 Knative **Service** YAML 对象中使用此注解：

对 HTTP/1 提供完整双工支持的 KnativeServing CR 示例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
  template:
    spec:
      annotations:
        features.knative.dev/http-full-duplex: "Enabled"
...
```

第 9 章 配置对 KNATIVE 服务的访问

9.1. 为 KNATIVE 服务配置 JSON WEB 令牌身份验证

OpenShift Serverless 当前没有用户定义的授权功能。要为部署添加用户定义的授权，您必须将 OpenShift Serverless 与 Red Hat OpenShift Service Mesh 集成，然后为 Knative 服务配置 JSON Web Token (JWT) 身份验证和 sidecar 注入。

9.2. 在 SERVICE MESH 2.X 中使用 JSON WEB 令牌身份验证

您可以使用 Service Mesh 2.x 和 OpenShift Serverless 在 Knative 服务中使用 JSON Web Token (JWT) 身份验证。要做到这一点，您必须在作为 **ServiceMeshMemberRoll** 对象成员的应用程序命名空间中创建身份验证请求和策略。您还必须为该服务启用 sidecar 注入。

9.2.1. 为 Service Mesh 2.x 和 OpenShift Serverless 配置 JSON Web 令牌身份验证

重要

在启用了 Kourier 时，不支持在系统命名空间中向 pod 添加 sidecar 注入，如 **knative-serving** 和 **knative-serving-ingress**。

对于 OpenShift Container Platform，如果您需要这些命名空间中的 pod 进行 sidecar 注入，请参阅 OpenShift Serverless 文档来 *原生将 Service Mesh 与 OpenShift Serverless 集成*。

先决条件

- 您已在集群中安装了 OpenShift Serverless Operator、Knative Serving 和 Red Hat OpenShift Service Mesh。
- 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，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 在您的服务中添加 **sidecar.istio.io/inject="true"** 注解：

   服务示例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: <service_name>
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true" 1
        sidecar.istio.io/rewriteAppHTTPProbers: "true" 2
...
```

1. 添加 **sidecar.istio.io/inject="true"** 注解。
2. 您必须在 Knative 服务中将注解 **sidecar.istio.io/rewriteAppHTTPProbers: "true"** 设置为 OpenShift Serverless 版本 1.14.0 或更新的版本，然后使用 HTTP 探测作为 Knative 服务的就绪度探测。

2. 应用 **Service** 资源：

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

3. 在 **ServiceMeshMemberRoll** 对象的每个无服务器应用程序命名空间中创建一个 **RequestAuthentication** 资源：

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: <namespace>
spec:
  jwtRules:
  - issuer: testing@secure.istio.io
    jwksUri: https://raw.githubusercontent.com/istio/istio/release-
1.8/security/tools/jwt/samples/jwks.json
```

4. 应用 **RequestAuthentication** 资源：

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

5. 通过创建以下 **AuthorizationPolicy** 资源，允许从 **ServiceMeshMemberRoll** 对象中的每个无服务器应用程序命名空间的系统 pod 访问 **RequestAuthenticaton** 资源：

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allowlist-by-paths
  namespace: <namespace>
spec:
  action: ALLOW
  rules:
  - to:
    - operation:
        paths:
        - /metrics 1
        - /healthz 2
```

1. 由系统 pod 收集指标的应用程序上的路径。
2. 系统 pod 探测到应用程序的路径。

6. 应用 **AuthorizationPolicy** 资源：

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

7. 对于作为 **ServiceMeshMemberRoll** 对象中成员的每个无服务器应用程序命名空间，请创建以下 **AuthorizationPolicy** 资源：

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: require-jwt
  namespace: <namespace>
spec:
  action: ALLOW
  rules:
  - from:
    - source:
        requestPrincipals: ["testing@secure.istio.io/testing@secure.istio.io"]
```

8. 应用 **AuthorizationPolicy** 资源：

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

验证

1. 如果您尝试使用 **curl** 请求来获取 Knative 服务 URL，则会被拒绝：

   示例命令

```
$ curl http://hello-example-1-default.apps.mycluster.example.com/
```

   输出示例

```
RBAC: access denied
```

2. 使用有效 JWT 验证请求。

   a. 获取有效的 JWT 令牌：

```
$ TOKEN=$(curl https://raw.githubusercontent.com/istio/istio/release-1.8/security/tools/jwt/samples/demo.jwt -s) && echo "$TOKEN" | cut -d '.' -f2 - | base64 --decode -
```

   b. 使用 **curl** 请求标头中的有效令牌访问该服务：

```
$ curl -H "Authorization: Bearer $TOKEN"  http://hello-example-1-default.apps.example.com
```

   现在允许请求：

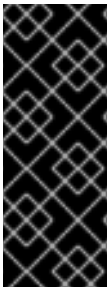
   输出示例

```
Hello OpenShift!
```

9.3. 在 SERVICE MESH 1.X 中使用 JSON WEB 令牌身份验证

您可以使用 Service Mesh 1.x 和 OpenShift Serverless 在 Knative 服务中使用 JSON Web Token (JWT) 身份验证。要做到这一点，您必须在作为 **ServiceMeshMemberRoll** 对象的成员的应用程序命名空间中创建策略。您还必须为该服务启用 sidecar 注入。

9.3.1. 为 Service Mesh 1.x 和 OpenShift Serverless 配置 JSON Web 令牌身份验证



重要

在启用了 Kourier 时，不支持在系统命名空间中向 pod 添加 sidecar 注入，如 **knative-serving** 和 **knative-serving-ingress**。

对于 OpenShift Container Platform，如果您需要这些命名空间中的 pod 进行 sidecar 注入，请参阅 OpenShift Serverless 文档来 *原生将 Service Mesh 与 OpenShift Serverless 集成*。

先决条件

- 您已在集群中安装了 OpenShift Serverless Operator、Knative Serving 和 Red Hat OpenShift Service Mesh。
- 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，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。

流程

1. 在您的服务中添加 **sidecar.istio.io/inject="true"** 注解：

   服务示例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: <service_name>
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true" 1
        sidecar.istio.io/rewriteAppHTTPProbers: "true" 2
...
```

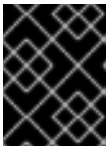
   1 添加 **sidecar.istio.io/inject="true"** 注解。

   2 您必须在 Knative 服务中将注解 **sidecar.istio.io/rewriteAppHTTPProbers: "true"** 设置为 OpenShift Serverless 版本 1.14.0 或更新的版本，然后使用 HTTP 探测作为 Knative 服务的就绪度探测。

2. 应用 **Service** 资源：

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

3. 在作为 **ServiceMeshMemberRoll** 对象的成员的无服务器应用程序命名空间中创建策略，该策略只允许具有有效 JSON Web Tokens（JWT）的请求：



重要

路径 **/metrics** 和 **/healthz** 必须包含在 **excludePaths** 中，因为它们是从 **knative-serving** 命名空间中的系统 pod 访问的。

```
apiVersion: authentication.istio.io/v1alpha1
kind: Policy
metadata:
  name: default
  namespace: <namespace>
spec:
  origins:
  - jwt:
      issuer: testing@secure.istio.io
      jwksUri: "https://raw.githubusercontent.com/istio/istio/release-
1.6/security/tools/jwt/samples/jwks.json"
      triggerRules:
      - excludedPaths:
        - prefix: /metrics 1
        - prefix: /healthz 2
  principalBinding: USE_ORIGIN
```

1. 由系统 pod 收集指标的应用程序上的路径。
2. 系统 pod 探测到应用程序的路径。

4. 应用 **Policy** 资源：

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

验证

1. 如果您尝试使用 **curl** 请求来获取 Knative 服务 URL，则会被拒绝：

```
$ curl http://hello-example-default.apps.mycluster.example.com/
```

输出示例

```
Origin authentication failed.
```

2. 使用有效 JWT 验证请求。
   a. 获取有效的 JWT 令牌：

```
$ TOKEN=$(curl https://raw.githubusercontent.com/istio/istio/release-
1.6/security/tools/jwt/samples/demo.jwt -s) && echo "$TOKEN" | cut -d '.' -f2 - | base64 --
decode -
```

   b. 使用 **curl** 请求标头中的有效令牌访问该服务：

```
$ curl http://hello-example-default.apps.mycluster.example.com/ -H "Authorization:
Bearer $TOKEN"
```

现在允许请求：

输出示例

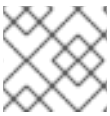
```
Hello OpenShift!
```

第 10 章 为 SERVING 配置 KUBE-RBAC-PROXY

kube-rbac-proxy 组件为 Knative Serving 提供内部身份验证和授权功能。

10.1. 为 SERVING 配置 KUBE-RBAC-PROXY 资源

您可以使用 OpenShift Serverless Operator CR 全局覆盖 **kube-rbac-proxy** 容器的资源分配。



注意

您还可以覆盖特定部署的资源分配。

以下配置设置 Knative Serving **kube-rbac-proxy** 最小值和最大 CPU 和内存分配：

KnativeServing CR 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  config:
    deployment:
      "kube-rbac-proxy-cpu-request": "10m" (1)
      "kube-rbac-proxy-memory-request": "20Mi" (2)
      "kube-rbac-proxy-cpu-limit": "100m" (3)
      "kube-rbac-proxy-memory-limit": "100Mi" (4)
```

1. 设置最小 CPU 分配。
2. 设置最小 RAM 分配。
3. 设置最大 CPU 分配。
4. 设置最大 RAM 分配。

第 11 章 为 NET-KOURIER 配置 BURST 和 QPS

每秒查询数(QPS)和 burst 值决定了对 API 服务器的请求或 API 调用的频率。

11.1. 为 NET-KOURIER 配置 BURST 和 QPS 值

每秒查询数(QPS)值决定了发送到 API 服务器的客户端请求或 API 调用的数量。

burst 值决定了客户端可以存储多少个请求进行处理。超过此缓冲区的请求将被丢弃。这可用于突发且不会及时统一其请求的控制器。

当 **net-kourier-controller** 重启时，它会解析集群中部署的所有 **入口** 资源，这会导致大量 API 调用。因此，**net-kourier-controller** 可能需要很长时间才能启动。

您可以在 KnativeServing CR 中调整 **net-kourier-controller** 的 QPS 和 burst 值：

KnativeServing CR 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  workloads:
  - name: net-kourier-controller
    env:
    - container: controller
      envVars:
      - name: KUBE_API_BURST
        value: "200" (1)
      - name: KUBE_API_QPS
        value: "200" (2)
```

1. 控制器和 API 服务器间的 QPS 速率。默认值为 200。
2. Kubelet 和 API 服务器之间通信的突发容量。默认值为 200。

第 12 章 为 KNATIVE 服务配置自定义域

12.1. 为 KNATIVE 服务配置自定义域

Knative 服务会自动根据集群配置分配默认域名。例如，**<service_name>-<namespace>.example.com**。您可以通过将您自己的自定义域名映射到 Knative 服务来自定义 Knative 服务域。

您可以通过为服务创建 **DomainMapping** 资源来完成此操作。您还可以创建多个 **DomainMapping** 资源，将多个域和子域映射到单个服务。

12.2. 自定义域映射

您可以通过将您自己的自定义域名映射到 Knative 服务来自定义 Knative 服务域。要将自定义域名映射到自定义资源（CR），您必须创建一个映射到可寻址目标 CR 的 **DomainMapping** CR，如 Knative 服务或 Knative 路由。

12.2.1. 创建自定义域映射

您可以通过将您自己的自定义域名映射到 Knative 服务来自定义 Knative 服务域。要将自定义域名映射到自定义资源（CR），您必须创建一个映射到可寻址目标 CR 的 **DomainMapping** CR，如 Knative 服务或 Knative 路由。

先决条件

- 在集群中安装了 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，以便在 OpenShift Container Platform 中创建应用程序和其他工作负载。
- 您已创建了 Knative 服务，并控制要映射到该服务的自定义域。

注意

您的自定义域必须指向 OpenShift Container Platform 集群的 IP 地址。

流程

1. 在与您要映射的目标 CR 相同的命名空间中创建一个包含 **DomainMapping** CR 的 YAML 文件：

```
apiVersion: serving.knative.dev/v1beta1
kind: DomainMapping
metadata:
 name: <domain_name> 1
 namespace: <namespace> 2
spec:
 ref:
   name: <target_name> 3
   kind: <target_type> 4
   apiVersion: serving.knative.dev/v1
```

1. 要映射到目标 CR 的自定义域名。
2. **DomainMapping** CR 和目标 CR 的命名空间。
3. 映射到自定义域的目标 CR 名称。
4. 映射到自定义域的 CR 类型。

服务域映射示例

```
apiVersion: serving.knative.dev/v1beta1
kind: DomainMapping
metadata:
 name: example.com
 namespace: default
spec:
 ref:
   name: showcase
   kind: Service
   apiVersion: serving.knative.dev/v1
```

路由域映射示例

```
apiVersion: serving.knative.dev/v1beta1
kind: DomainMapping
metadata:
 name: example.com
 namespace: default
spec:
 ref:
   name: example-route
   kind: Route
   apiVersion: serving.knative.dev/v1
```

2. 将 **DomainMapping** CR 应用为 YAML 文件：

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

12.3. 使用 KNATIVE CLI 的 KNATIVE 服务自定义域

您可以通过将您自己的自定义域名映射到 Knative 服务来自定义 Knative 服务域。您可以使用 Knative (**kn**) CLI 创建映射到可寻址目标 CR 的 **DomainMapping** 自定义资源 (CR)，如 Knative 服务或 Knative 路由。

12.3.1. 使用 Knative CLI 创建自定义域映射

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 您已创建了 Knative 服务或路由，并控制要映射到该 CR 的自定义域。

注意

您的自定义域必须指向 OpenShift Container Platform 集群的 DNS。

- 已安装 Knative (**kn**) CLI。
- 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。

流程

- 将域映射到当前命名空间中的 CR：

```
$ kn domain create <domain_mapping_name> --ref <target_name>
```

示例命令

```
$ kn domain create example.com --ref showcase
```

--ref 标志为域映射指定一个可寻址的目标 CR。

如果使用 **--ref** 标志时没有提供前缀，则会假定目标为当前命名空间中的 Knative 服务。

- 将域映射到指定命名空间中的 Knative 服务：

```
$ kn domain create <domain_mapping_name> --ref
<ksvc:service_name:service_namespace>
```

示例命令

```
$ kn domain create example.com --ref ksvc:showcase:example-namespace
```

- 将域映射到 Knative 路由：

```
$ kn domain create <domain_mapping_name> --ref <kroute:route_name>
```

示例命令

```
$ kn domain create example.com --ref kroute:example-route
```

12.4. 使用 DEVELOPER 视角的域映射

您可以通过将您自己的自定义域名映射到 Knative 服务来自定义 Knative 服务域。您可以使用 OpenShift Container Platform Web 控制台的 **Developer** 视角将 **DomainMapping** 自定义资源 (CR) 映射到 Knative 服务。

12.4.1. 使用 Developer 视角将自定义域映射到服务

先决条件

- 已登陆到 web 控制台。

- 处于 **Developer** 视角。
- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。这必须由集群管理员完成。
- 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。
- 您已创建了 Knative 服务，并控制要映射到该服务的自定义域。

注意

您的自定义域必须指向 OpenShift Container Platform 集群的 IP 地址。

流程

1. 导航到 **Topology** 页面。
2. 右键单击您要映射到某个域的服务，然后选择包含服务名称的 **Edit** 选项。例如，如果服务命名为 **showcase**，请选择 **Edit showcase** 选项。
3. 在 **Advanced options** 部分中，点 **Show advanced Routing options**。
   a. 如果要映射到该服务的域映射 CR 已存在，您可以在**域映射**列表中选择。
   b. 如果要创建新域映射 CR，在框中输入域名，然后选择 **Create** 选项。例如，如果您在 **example.com 中** 键入，则 **Create** 选项为 **Create "example.com"**。
4. 单击 **Save**，将更改保存到您的服务。

验证

1. 导航到 **Topology** 页面。
2. 单击您创建的服务。
3. 在服务信息窗口的 **Resources** 选项卡中，您可以看到您映射到**域映射**中列出的服务的域。

12.5. 使用 ADMINISTRATOR 视角的域映射

如果您不想在 OpenShift Container Platform web 控制台中切换到 **Developer** 视角，或使用 Knative (**kn**) CLI 或 YAML 文件，您可以使用 OpenShift Container Platform Web 控制台的 **Administator** 视角。

12.5.1. 使用 Administrator 视角将自定义域映射到服务

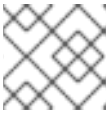
Knative 服务会自动根据集群配置分配默认域名。例如，**<service_name>-<namespace>.example.com**。您可以通过将您自己的自定义域名映射到 Knative 服务来自定义 Knative 服务域。

您可以通过为服务创建 **DomainMapping** 资源来完成此操作。您还可以创建多个 **DomainMapping** 资源，将多个域和子域映射到单个服务。

如果您在 OpenShift Container Platform 上具有集群管理员权限（或 OpenShift Dedicated 或 Red Hat OpenShift Service on AWS 的集群管理员权限），您可以使用 web 控制台中的 **Administrator** 视角创建一个 **DomainMapping** 自定义资源(CR)。

先决条件

- 已登陆到 web 控制台。
- 您处于 **Administrator** 视角。
- 已安装 OpenShift Serverless Operator。
- 已安装 Knative Serving。
- 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。
- 您已创建了 Knative 服务，并控制要映射到该服务的自定义域。



注意

您的自定义域必须指向集群的 IP 地址。

流程

1. 导航到 **CustomResourceDefinitions**，并使用搜索框查找 **DomainMapping** 自定义资源定义 (CRD)。
2. 点 **DomainMapping** CRD，然后导航到 **Instances** 选项卡。
3. 单击 **Create DomainMapping**。
4. 修改 **DomainMapping** CR 的 YAML，使其为您的实例包含以下信息：

```
apiVersion: serving.knative.dev/v1beta1
kind: DomainMapping
metadata:
 name: <domain_name> 1
 namespace: <namespace> 2
spec:
 ref:
   name: <target_name> 3
   kind: <target_type> 4
   apiVersion: serving.knative.dev/v1
```

1. 要映射到目标 CR 的自定义域名。
2. **DomainMapping** CR 和目标 CR 的命名空间。
3. 映射到自定义域的目标 CR 名称。
4. 映射到自定义域的 CR 类型。

到 Knative 服务的域映射示例

```
apiVersion: serving.knative.dev/v1beta1
kind: DomainMapping
metadata:
 name: custom-ksvc-domain.example.com
```

```
 namespace: default
spec:
 ref:
   name: showcase
   kind: Service
   apiVersion: serving.knative.dev/v1
```

验证

- 使用 **curl** 请求访问自定义域。例如：

  示例命令

  ```
  $ curl custom-ksvc-domain.example.com
  ```

  输出示例

  ```
  {"artifact":"knative-showcase","greeting":"Welcome"}
  ```

12.5.2. 使用 Administrator 视角限制密码套件

当您为 ingress 指定 **net-kourier** 并使用 **DomainMapping** 时，OpenShift 路由的 TLS 被设置为 passthrough，TLS 由 Kourier 网关处理。在这种情况下，您可能需要限制用户允许 Kourier 的 TLS 密码套件。

先决条件

- 已登陆到 web 控制台。
- 您处于 **Administrator** 视角。
- 已安装 OpenShift Serverless Operator。
- 已安装 Knative Serving。
- 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。

注意

您的自定义域必须指向集群的 IP 地址。

流程

- 在 **KnativeServing** CR 中，使用 **cipher-suites** 值来指定您要启用的密码套件：

  KnativeServing CR 示例

  ```
  spec:
    config:
      kourier:
        cipher-suites: ECDHE-ECDSA-AES128-GCM-SHA256,ECDHE-ECDSA-CHACHA20-
  POLY1305
  ```

将禁用其他密码套件。您可以使用逗号分隔多个套件。

注意

Kourier 网关的容器镜像使用 Envoy 代理镜像，默认启用的密码套件取决于 Envoy 代理的版本。

12.6. 使用 TLS 证书保护映射的服务

12.6.1. 使用 TLS 证书保护带有自定义域的服务

为 Knative 服务配置了自定义域后，您可以使用 TLS 证书来保护映射的服务。要做到这一点，您必须创建一个 Kubernetes TLS secret，然后更新 **DomainMapping** CR 以使用您创建的 TLS secret。

先决条件

- 为 Knative 服务配置了自定义域，并有一个正常工作的 **DomainMapping** CR。
- 您有来自证书授权机构供应商或自签名证书的 TLS 证书。
- 您已从证书授权中心（CA）提供商或自签名证书获取 **cert** 和 **key** 文件。
- 安装 OpenShift CLI (**oc**) 。

流程

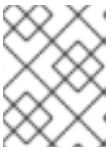
1. 创建 Kubernetes TLS secret：

```
$ oc create secret tls <tls_secret_name> --cert=<path_to_certificate_file> --key=
<path_to_key_file>
```

2. 将 **networking.internal.knative.dev/certificate-uid: <id>'** 标签添加到 Kubernetes TLS secret 中：

```
$ oc label secret <tls_secret_name> networking.internal.knative.dev/certificate-uid="<id>"
```

如果使用第三方 secret 供应商（如 cert-manager），您可以配置 secret manager 来自动标记 Kubernetes TLS secret。cert-manager 用户可以使用提供的 secret 模板自动生成带有正确标签的 secret。在本例中，secret 过滤仅基于键，但这个值可以存储有用的信息，如 secret 包含的证书 ID。



注意

Red Hat OpenShift 的 cert-manager Operator 只是一个技术预览功能。如需更多信息，请参阅 为 **Red Hat OpenShift 安装 cert-manager Operator**文档。

3. 更新 **DomainMapping** CR，以使用您创建的 TLS secret：

```
apiVersion: serving.knative.dev/v1beta1
kind: DomainMapping
metadata:
  name: <domain_name>
  namespace: <namespace>
```

```
spec:
  ref:
    name: <service_name>
    kind: Service
    apiVersion: serving.knative.dev/v1
# TLS block specifies the secret to be used
  tls:
    secretName: <tls_secret_name>
```

验证

1. 验证 **DomainMapping** CR 状态是否为 **True**，输出中的 **URL** 列显示了使用 scheme **https** 的映射域：

```
$ oc get domainmapping <domain_name>
```

输出示例

```
NAME                URL                         READY   REASON
example.com         https://example.com         True
```

2. 可选：如果服务公开，请运行以下命令验证该服务是否可用：

```
$ curl https://<domain_name>
```

如果证书是自签名的，请通过在 **curl** 命令中添加 **-k** 标志来跳过验证。

12.6.2. 使用 secret 过滤改进 net-kourier 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可能会导致大量开销。这可能会导致因为内存泄露的问题导致 Knative **net-kourier** ingress 控制器在大型集群中失败。但是，Knative **net-kourier** ingress 控制器提供了一个过滤机制，它可让控制器只获取 Knative 相关的 secret。

在 OpenShift Serverless Operator 端默认启用 secret 过滤。默认情况下，将环境变量 **ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID=true** 添加到 **net-kourier** 控制器 Pod 中。

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。
- 您创建的项目，或者具有创建应用程序和其他工作负载的角色和权限。
- 安装 OpenShift Serverless Operator 和 Knative Serving。
- 安装 OpenShift CLI (**oc**) 。

您可以使用 **KnativeServing** 自定义资源(CR)中的 workload 字段将 **ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID** 变量设为 **false** 来禁用 secret 过滤。

KnativeServing CR 示例

```
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-kourier-controller
```

第 13 章 KNATIVE SERVING 的高可用性配置

13.1. KNATIVE 服务的高可用性

高可用性 (HA) 是 Kubernetes API 的标准功能，有助于确保在出现中断时 API 保持正常运行。在 HA 部署中，如果活跃控制器崩溃或被删除，另一个控制器就可以使用。此控制器会接管处理由现在不可用的控制器提供服务的 API。

OpenShift Serverless 中的 HA 可通过领导选举机制获得，该机制会在安装 Knative Serving 和 Eventing control plane 后默认启用。在使用领导选举 HA 模式时，控制器实例在需要前应该已在集群内调度并运行。这些控制器实例争用共享资源，即领导选举锁定。在任何给定时间可以访问领导选举机制锁定资源的控制器实例被称为领导 (leader) 。

13.2. KNATIVE 部署的高可用性

默认情况下，Knative Serving activator, autoscaler ,autoscaler -hpa,controller, webhook , domain-mapping ,domainmapping- webhook,domainmapping-webhook,kourier-control, 和 kourier-gateway 组件都可用，它们各自配置为有两个副本。您可以通过修改 KnativeServing 自定义资源 (CR) 中的 spec.high-availability.replicas 值来更改这些组件的副本数。

13.2.1. 为 Knative Serving 配置高可用性副本

要为有资格的部署资源指定三个最小副本，请将自定义资源中的 spec.high-availability.replicas 的值设置为 3。

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。
- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。

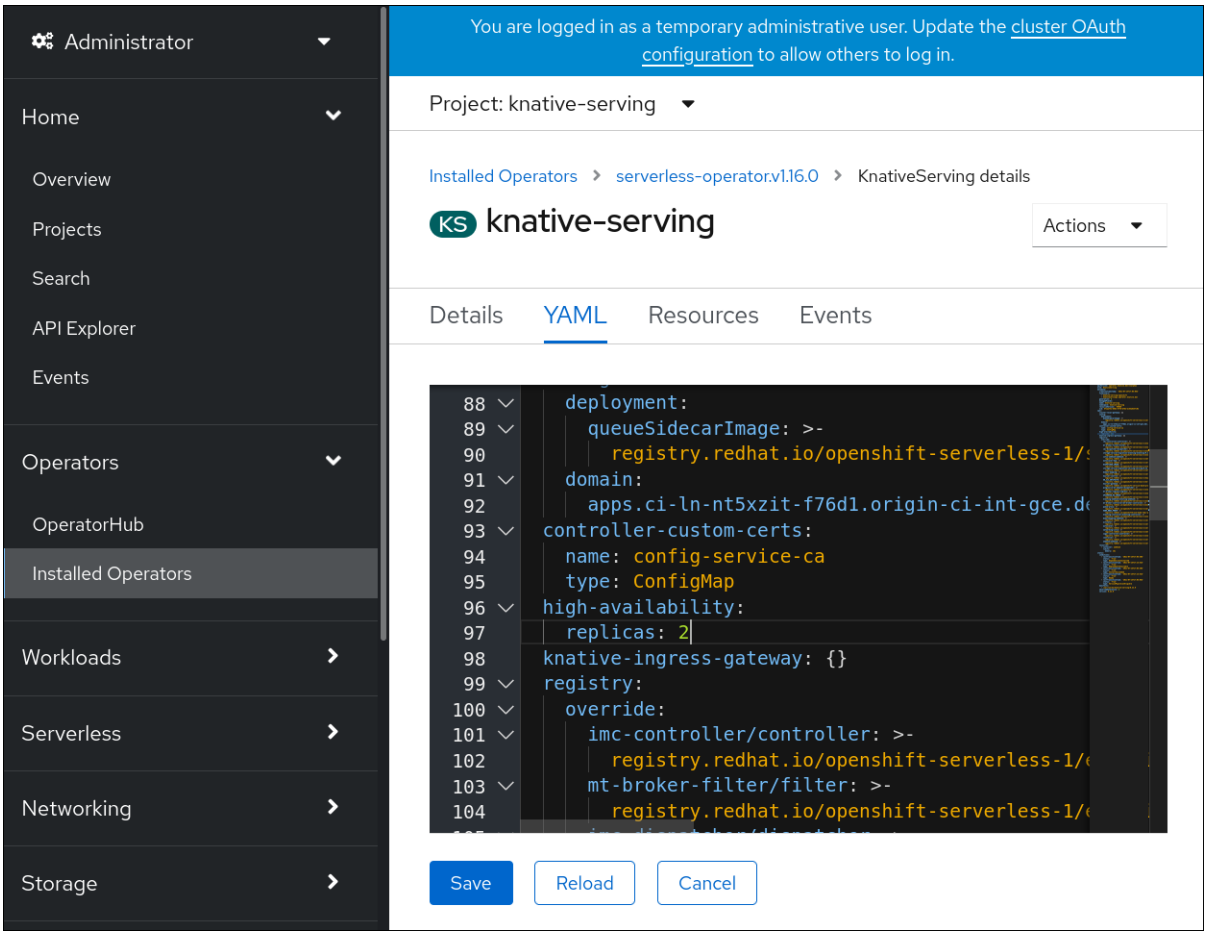
流程

1. 在 OpenShift Container Platform web 控制台的 Administrator 视角中，进入 OperatorHub → Installed Operators。

2. 选择 **knative-serving** 命名空间。

3. 点 **OpenShift Serverless Operator** 的 **Provided APIs** 列表中的 **Knative Serving** 来进入 **Knative Serving** 选项卡。

4. 点 **knative-serving**，然后使用 **knative-serving** 页面中的 **YAML** 选项卡。



5. 修改 **KnativeServing CR** 中的副本数量：

YAML 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  high-availability:
    replicas: 3
```

第 14 章 调优服务配置

14.1. 覆盖 KNATIVE SERVING 系统部署配置

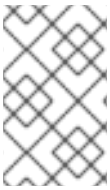
您可以通过修改 KnativeServing 自定义资源(CR)中的 workload spec 来覆盖某些特定部署的默认配置。

14.1.1. 覆盖系统部署配置

目前，支持覆盖 resources, replicas, labels, annotations, 和 nodeSelector 项的默认配置设置，以及探测的 readiness 和 liveness 字段的默认设置。

在以下示例中，KnativeServing CR 会覆盖 Webhook 部署，以便：

- net-kourier-controller 的 readiness 探测超时设置为 10 秒。
- 部署指定了 CPU 和内存资源限制。
- 部署有 3 个副本。
- 添加 example-label: label 标签。
- 添加 example-annotation: 注解。
- nodeSelector 字段被设置为选择带有 disktype: hdd 标签的节点。



注意

KnativeServing CR 标签和注解设置覆盖部署本身和生成的 Pod 的部署标签和注解。

KnativeServing CR 示例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: ks
  namespace: knative-serving
spec:
  high-availability:
    replicas: 2
  workloads:
  - name: net-kourier-controller
    readinessProbes: 1
      - container: controller
        timeoutSeconds: 10
  - name: webhook
    resources:
    - container: webhook
      requests:
        cpu: 300m
        memory: 60Mi
      limits:
        cpu: 1000m
        memory: 1000Mi
    replicas: 3
    labels:
      example-label: label
    annotations:
      example-annotation: annotation
    nodeSelector:
      disktype: hdd
```

1

您可以使用 readiness 和 liveness 探测覆盖来覆盖在 Kubernetes API 中指定的一个部署中的一个容器探测的所有字段，与探测 handler: exec, grpc, httpGet, 和 tcpSocket 相关的字段除外。

其他资源

- Kubernetes API 文档中的探测配置部分

第 15 章 配置队列代理资源

Queue Proxy 是服务中每个应用程序容器的 sidecar 容器。它改进了管理 Serverless 工作负载，确保有效的资源使用量。您可以配置 Queue Proxy。

15.1. 为 KNATIVE SERVICE 配置队列代理资源

除了在部署 configmap 中全局配置 Queue Proxy 资源请求和限值外，您还可以使用以 CPU、内存和临时存储资源类型为目标的相应注解在服务级别设置它们。

先决条件

- 在集群中必须安装 Red Hat OpenShift Pipelines。
- 已安装 OpenShift (oc) CLI。
- 已安装 Knative (kn) CLI。

流程

- 使用资源请求和限值修改服务的 configmap：

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
  template:
    metadata:
      annotations:
        queue.sidecar.serving.knative.dev/cpu-resource-request: "1"
        queue.sidecar.serving.knative.dev/cpu-resource-limit: "2"
        queue.sidecar.serving.knative.dev/memory-resource-request: "1Gi"
        queue.sidecar.serving.knative.dev/memory-resource-limit: "2Gi"
        queue.sidecar.serving.knative.dev/ephemeral-storage-resource-request: "400Mi"
        queue.sidecar.serving.knative.dev/ephemeral-storage-resource-limit: "450Mi"
```

或者，您可以使用特殊的 queue.sidecar.serving.knative.dev/resource-percentage 注解，该注解将 Queue Proxy 资源计算为应用程序容器的百分比。当从应用程序容器要求计算 CPU 和

内存资源要求且位于以下边界外时，这些值会被调整以容纳在边界内。在这种情况下，以下最小和最大界限应用于 CPU 和内存资源要求：

表 15.1. 资源要求边界

资源要求	Min	Max
CPU 请求	25m	100m
CPU 限制	40m	500m
内存请求	50Mi	200Mi
内存限制	200Mi	500Mi

注意

如果您同时使用对应的资源注解设置了百分比注解和特定资源值，则后者将具有优先权。

警告

queue.sidecar.serving.knative.dev/resource-percentage 注解现已弃用，并将在以后的发行版本中删除。