Red Hat OpenShift Serverless 1.31

Knative CLI

Knative Functions、Serving 和 Eventing 的 CLI 命令概述

Last Updated: 2024-01-05

Red Hat OpenShift Serverless 1.31 Knative CLI

Knative Functions、Serving 和 Eventing 的 CLI 命令概述

法律通告

摘要

本文档概述了 Knative Functions、Serving 和 Eventing 的 CLI 命令。它还提供有关配置 Knative CLI 和使用插件的信息。

目录

第 1 章 KNATIVE SERVING CLI 命令

1.1. KN SERVICE 命令

您可以使用以下命令创建和管理 Knative 服务。

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 Knative CLI 更新无服务器应用程序

在以递增方式构建服务时，您可以使用 **kn service update** 命令进行命令行上的互动会话。与 **kn service apply** 命令不同，在使用 **kn service update** 命令时，只需要指定您要更新的更改，而不是指定 Knative 服务的完整配置。

示例命令

- 通过添加新环境变量来更新服务：

```
$ kn service update <service_name> --env <key>=<value>
```

- 通过添加新端口来更新服务：

```
$ kn service update <service_name> --port 80
```

- 通过添加新的请求和限制参数来更新服务：

```
$ kn service update <service_name> --request cpu=500m --limit memory=1024Mi --limit
cpu=1000m
```

- 为修订分配 **latest** 标签：

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

- 为服务的最新 **READY** 修订将标签从 **testing** 更新为 **staging**：

```
$ kn service update <service_name> --untag testing --tag @latest=staging
```

- 将 **test** 标签添加到接收 10% 流量的修订，并将其它流量发送到服务的最新 **READY** 修订：

```
$ kn service update <service_name> --tag <revision_name>=test --traffic test=10,@latest=90
```

1.1.3. 应用服务声明

您可以使用 **kn service apply** 命令声明性配置 Knative 服务。如果服务不存在，则使用已更改的选项更新现有服务。

kn service apply 命令对 shell 脚本或持续集成管道特别有用，因为用户通常希望在单个命令中完全指定服务的状态来声明目标状态。

使用 **kn service apply** 时，必须为 Knative 服务提供完整的配置。这与 **kn service update** 命令不同，它只在命令中指定您要更新的选项。

示例命令

- 创建服务：

```
$ kn service apply <service_name> --image <image>
```

- 将环境变量添加到服务：

```
$ kn service apply <service_name> --image <image> --env <key>=<value>
```

- 从 JSON 或 YAML 文件中读取服务声明：

```
$ kn service apply <service_name> -f <filename>
```

1.1.4. 使用 Knative CLI 描述无服务器应用程序

您可以使用 **kn service describe** 命令来描述 Knative 服务。

示例命令

- 描述服务：

```
$ kn service describe --verbose <service_name>
```

--verbose 标志是可选的，但可以包含它以提供更详细的描述。常规输出和详细输出之间的区别在以下示例中显示：

没有 --verbose 标记的输出示例

```
Name:       showcase
Namespace:  default
Age:        2m
URL:        http://showcase-default.apps.ocp.example.com

Revisions:
  100%  @latest (showcase-00001) [1] (2m)
        Image:  quay.io/openshift-knative/showcase (pinned to aaea76)

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                1m
  ++ ConfigurationsReady     1m
  ++ RoutesReady             1m
```

带有 --verbose 标记的输出示例

```
Name:         showcase
Namespace:    default
Annotations:  serving.knative.dev/creator=system:admin
              serving.knative.dev/lastModifier=system:admin
Age:          3m
URL:          http://showcase-default.apps.ocp.example.com
Cluster:      http://showcase.default.svc.cluster.local

Revisions:
  100%  @latest (showcase-00001) [1] (3m)
        Image:  quay.io/openshift-knative/showcase (pinned to aaea76)
        Env:    GREET=Bonjour

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                3m
  ++ ConfigurationsReady     3m
  ++ RoutesReady             3m
```

-

- 以 YAML 格式描述服务：

```
$ kn service describe <service_name> -o yaml
```

- 以 JSON 格式描述服务：

```
$ kn service describe <service_name> -o json
```

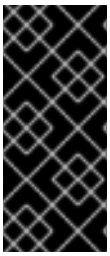
- 仅输出服务 URL：

```
$ kn service describe <service_name> -o url
```

1.2. 处于离线模式的 KN SERVICE 命令

1.2.1. 关于 Knative CLI 离线模式

执行 **kn service** 命令时，更改会立即传播到集群。但是，作为替代方案，您可以在离线模式下执行 **kn service** 命令。当您以离线模式创建服务时，集群不会发生任何更改，而是在本地计算机上创建服务描述符文件。



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。

创建描述符文件后，您可以手动修改并在版本控制系统中跟踪该文件。您还可以使用 **kn service create -f**、**kn service apply -f** 或 **oc apply -f** 命令将更改传播到集群。

离线模式有几种用途：

- 在使用描述符文件对集群进行更改之前，您可以手动修改该文件。
- 您可以在本地跟踪版本控制系统中服务的描述符文件。这可让您在目标集群以外的位置重复使用描述符文件，例如在持续集成（CI）管道、开发环境或演示中。
- 您可以检查创建的描述符文件，以了解 Knative 服务的信息。特别是，您可以看到生成的服务如何受到传递给 **kn** 命令的不同参数的影响。

离线模式有其优点：速度非常快，不需要连接到集群。但是，离线模式缺少服务器端验证。因此，您无法验证服务名称是否唯一，或者是否可以拉取指定镜像。

1.2.2. 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
```

1.3. KN 容器命令

您可以使用以下命令在 Knative 服务规格中创建和管理多个容器。

1.3.1. Knative 客户端多容器支持

您可以使用 **kn container add** 命令将 YAML 容器 spec 打印到标准输出。此命令对多容器用例很有用，因为它可以与其他标准 **kn** 标志一起使用来创建定义。

kn container add 命令接受与容器相关的所有标志，它们都支持与 **kn service create** 命令搭配使用。**kn container add** 命令也可以使用 UNIX 管道 (|) 一次创建多个容器定义来串联。

示例命令

- 从镜像添加容器并将其打印到标准输出中：

```
$ kn container add <container_name> --image <image_uri>
```

示例命令

```
$ kn container add sidecar --image docker.io/example/sidecar
```

输出示例

```
containers:
- image: docker.io/example/sidecar
  name: sidecar
  resources: {}
```

- 将两个 **kn container add** 命令链接在一起，然后将它们传递给 **kn service create** 命令创建带有两个容器的 Knative 服务：

```
$ kn container add <first_container_name> --image <image_uri> | \
kn container add <second_container_name> --image <image_uri> | \
kn service create <service_name> --image <image_uri> --extra-containers -
```

--extra-containers - 指定一个特殊情况，**kn** 读取管道输入，而不是 YAML 文件。

示例命令

```
$ kn container add sidecar --image docker.io/example/sidecar:first | \
kn container add second --image docker.io/example/sidecar:second | \
kn service create my-service --image docker.io/example/my-app:latest --extra-containers -
```

--extra-containers 标志也可以接受到 YAML 文件的路径：

```
$ kn service create <service_name> --image <image_uri> --extra-containers <filename>
```

示例命令

```
$ kn service create my-service --image docker.io/example/my-app:latest --extra-containers
my-extra-containers.yaml
```

1.4. KN 域命令

您可以使用下列命令创建和管理域映射。

1.4.1. 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
```

1.4.2. 使用 Knative CLI 管理自定义域映射

创建 **DomainMapping** 自定义资源 (CR) 后，您可以使用 Knative (**kn**) CLI 列出现有 CR、查看现有 CR 的信息、更新 CR 或删除 CR。

先决条件

- 在集群中安装了 OpenShift Serverless Operator 和 Knative Serving。
- 您至少已创建了一个 **DomainMapping** CR。
- 已安装 Knative (**kn**) CLI 工具。
- 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

- 列出现有的 **DomainMapping** CR：

```
$ kn domain list -n <domain_mapping_namespace>
```

- 查看现有 **DomainMapping** CR 的详情：

```
$ kn domain describe <domain_mapping_name>
```

- 更新 **DomainMapping** CR 以指向新目标：

```
$ kn domain update --ref <target>
```

- 删除 **DomainMapping** CR：

```
$ kn domain delete <domain_mapping_name>
```

第 2 章 配置 KNATIVE CLI

您可以通过创建 **config.yaml** 配置文件来自定义 Knative (**kn**) CLI 设置。您可以使用 **--config** 标志来提供此配置，否则会从默认位置提取配置。默认配置位置符合 XDG Base Directory 规格，对于 UNIX 系统和 Windows 系统有所不同。

对于 UNIX 系统：

- 如果设置了 **XDG_CONFIG_HOME** 环境变量，Knative (**kn**) CLI 查找的默认配置位置为 **$XDG_CONFIG_HOME/kn**。
- 如果没有设置 **XDG_CONFIG_HOME** 环境变量，Knative (**kn**) CLI 会在 **$HOME/.config/kn/config.yaml** 的用户主目录中查找配置。

对于 Windows 系统，默认的 Knative (**kn**) CLI 配置位置为 **%APPDATA%\kn**。

配置文件示例

```
plugins:
  path-lookup: true (1)
  directory: ~/.config/kn/plugins (2)
eventing:
  sink-mappings: (3)
  - prefix: svc (4)
    group: core (5)
    version: v1 (6)
    resource: services (7)
```

(1) 指定 Knative (**kn**) CLI 是否应该在 **PATH** 环境变量中查找插件。这是一个布尔值配置选项。默认值为 **false**。

(2) 指定 Knative (**kn**) CLI 查找插件的目录。默认路径取决于操作系统，如前面所述。这可以是用户可见的任何目录。

(3) **sink-mappings** spec 定义了在 Knative (**kn**) CLI 命令中使用 **--sink** 标志时使用的 Kubernetes 可寻址资源。

(4) 您用来描述接收器（sink）的前缀。**svc**（用于服务）、**channel** 和 **broker** 是 Knative (**kn**) CLI 中预定义的前缀。

(5) Kubernetes 资源的 API 组。

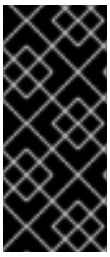
(6) Kubernetes 资源的版本。

(7) Kubernetes 资源类型的复数名称。例如，**services** 或 **brokers**。

第 3 章 KNATIVE CLI 插件

Knative (**kn**) CLI 支持使用插件，这允许您通过添加不是核心发行版本一部分的自定义命令和其他共享命令来扩展 **kn** 安装的功能。Knative (**kn**) CLI 插件的使用方式与主 **kn** 功能相同。

目前，红帽支持 **kn-source-kafka** 插件和 **kn-event** 插件。



重要

kn-event 插件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。

3.1. 使用 KN-EVENT 插件构建事件

您可以使用 **kn event build** 命令的 builder 接口来构建事件。然后，您可以稍后发送该事件或在另一个上下文中使用它。

先决条件

- 已安装 Knative (**kn**) CLI。

流程

- 构建事件：

```
$ kn event build --field <field-name>=<value> --type <type-name> --id <id> --output <format>
```

其中：

  - **--field** 标志将数据作为字段值对添加到事件中。您可以多次使用它。
  - **--type** 标志允许您指定指定事件类型的字符串。
  - **--id** 标志指定事件的 ID。
  - 您可以将 **json** 或 **yaml** 参数与 **--output** 标志一起使用，以更改事件的输出格式。所有这些标记都是可选的。

构建简单的事件

```
$ kn event build -o yaml
```

结果为 YAML 格式

```
data: {}
datacontenttype: application/json
id: 81a402a2-9c29-4c27-b8ed-246a253c9e58
source: kn-event/v0.4.0
specversion: "1.0"
time: "2021-10-15T10:42:57.713226203Z"
type: dev.knative.cli.plugin.event.generic
```

-

构建示例事务事件

```
$ kn event build \
    --field operation.type=local-wire-transfer \
    --field operation.amount=2345.40 \
    --field operation.from=87656231 \
    --field operation.to=2344121 \
    --field automated=true \
    --field signature='FGzCPLvYWdEgsdpb3qXkaVp7Da0=' \
    --type org.example.bank.bar \
    --id $(head -c 10 < /dev/urandom | base64 -w 0) \
    --output json
```

JSON 格式的结果事件

```
{
  "specversion": "1.0",
  "id": "RjtL8UH66X+UJg==",
  "source": "kn-event/v0.4.0",
  "type": "org.example.bank.bar",
  "datacontenttype": "application/json",
  "time": "2021-10-15T10:43:23.113187943Z",
  "data": {
    "automated": true,
    "operation": {
      "amount": "2345.40",
      "from": 87656231,
      "to": 2344121,
      "type": "local-wire-transfer"
    },
    "signature": "FGzCPLvYWdEgsdpb3qXkaVp7Da0="
  }
}
```

3.2. 使用 KN-EVENT 插件发送事件

您可以使用 **kn event send** 命令来发送事件。事件可以发送到公开的地址，或发送到集群中的可寻址资源，如 Kubernetes 服务，以及 Knative 服务、代理和频道。命令使用与 **kn event build** 命令相同的 builder 接口。

先决条件

- 已安装 Knative (**kn**) CLI。

流程

- 发送事件：

```
$ kn event send --field <field-name>=<value> --type <type-name> --id <id> --to-url <url> --to <cluster-resource> --namespace <namespace>
```

其中：

- **--field** 标志将数据作为字段值对添加到事件中。您可以多次使用它。
- **--type** 标志允许您指定指定事件类型的字符串。
- **--id** 标志指定事件的 ID。
- 如果您要将事件发送到公开的目的地，请使用 **--to-url** 标志指定 URL。
- 如果要将事件发送到集群内 Kubernetes 资源，请使用 **--to** 标志指定目的地。
  - 使用 **<Kind>:<ApiVersion>:<name>** 格式指定 Kubernetes 资源。
- **--namespace** 标志指定命名空间。如果省略，则会从当前上下文中获取命名空间。

所有这些标志都是可选的，除了目的地规格外，您需要使用 **--to-url** 或 **--to**。

以下示例显示向 URL 发送事件：

示例命令

```
$ kn event send \
    --field player.id=6354aa60-ddb1-452e-8c13-24893667de20 \
    --field player.game=2345 \
    --field points=456 \
    --type org.example.gaming.foo \
    --to-url http://ce-api.foo.example.com/
```

以下示例显示了将事件发送到 in-cluster 资源：

示例命令

```
$ kn event send \
    --type org.example.kn.ping \
    --id $(uuidgen) \
    --field event.type=test \
    --field event.data=98765 \
    --to Service:serving.knative.dev/v1:event-display
```

第 4 章 KNATIVE EVENTING CLI 命令

4.1. KN SOURCE 命令

您可以使用以下命令列出、创建和管理 Knative 事件源。

4.1.1. 使用 Knative CLI 列出可用事件源类型

您可以使用 **kn source list-types** CLI 命令列出集群中创建和使用的事件源类型。

先决条件

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

流程

1. 列出终端中的可用事件源类型：

```
$ kn source list-types
```

输出示例

```
TYPE              NAME                                DESCRIPTION
ApiServerSource   apiserversources.sources.knative.dev  Watch and send Kubernetes API events to a sink
PingSource        pingsources.sources.knative.dev       Periodically send ping events to a sink
SinkBinding       sinkbindings.sources.knative.dev      Binding for connecting a PodSpecable to a sink
```

2. 可选：在 OpenShift Container Platform 中，您也可以使用 YAML 格式列出可用事件源类型：

```
$ kn source list-types -o yaml
```

4.1.2. Knative CLI sink 标记

当使用 Knative (**kn**) CLI 创建事件源时，您可以使用 **--sink** 标志指定事件从该资源发送到的接收器。sink 可以是任何可寻址或可调用的资源，可以从其他资源接收传入的事件。

以下示例创建使用服务 **http://event-display.svc.cluster.local** 的接收器绑定作为接收器：

使用 sink 标记的命令示例

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
  --subject "Job:batch/v1:app=heartbeat-cron" \
  --sink http://event-display.svc.cluster.local \ 1
  --ce-override "sink=bound"
```

1 **http://event-display.svc.cluster.local** 中的 **svc** 确定接收器是一个 Knative 服务。其他默认的接收器前缀包括 **channel** 和 **broker**。

4.1.3. 使用 Knative CLI 创建和管理容器源

您可以使用 **kn source container** 命令来使用 Knative (**kn**) 创建和管理容器源。使用 Knative CLI 创建事件源提供了比直接修改 YAML 文件更精简且直观的用户界面。

创建容器源

```
$ kn source container create <container_source_name> --image <image_uri> --sink <sink>
```

删除容器源

```
$ kn source container delete <container_source_name>
```

描述容器源

```
$ kn source container describe <container_source_name>
```

列出现有容器源

```
$ kn source container list
```

以 YAML 格式列出现有容器源

```
$ kn source container list -o yaml
```

更新容器源

此命令为现有容器源更新镜像 URI：

```
$ kn source container update <container_source_name> --image <image_uri>
```

4.1.4. 使用 Knative CLI 创建 API 服务器源

您可以使用 **kn source apiserver create** 命令，使用 **kn** CLI 创建 API 服务器源。使用 **kn** CLI 创建 API 服务器源可提供比直接修改 YAML 文件更精简且直观的用户界面。

先决条件

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

流程

如果要重新使用现有服务帐户，您可以修改现有的 **ServiceAccount** 资源，使其包含所需的权限，而不是创建新资源。

1. 以 YAML 文件形式,为事件源创建服务帐户、角色和角色绑定：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: events-sa
  namespace: default 1

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: event-watcher
  namespace: default 2
rules:
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - get
      - list
      - watch

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: k8s-ra-event-watcher
  namespace: default 3
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: event-watcher
subjects:
  - kind: ServiceAccount
    name: events-sa
    namespace: default 4
```

1 2 3 4 将这个命名空间更改为已选择安装事件源的命名空间。

2. 应用 YAML 文件：

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

3. 创建具有事件 sink 的 API 服务器源。在以下示例中，sink 是一个代理：

```
$ kn source apiserver create <event_source_name> --sink broker:<broker_name> --
resource "event:v1" --service-account <service_account_name> --mode Resource
```

4. 要检查 API 服务器源是否已正确设置，请创建一个 Knative 服务，在日志中转储传入的信息：

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

5. 如果您使用代理作为事件 sink，请创建一个触发器将事件从 **default** 代理过滤到服务：

```
$ kn trigger create <trigger_name> --sink ksvc:event-display
```

6. 通过在 default 命名空间中启动 pod 来创建事件：

```
$ oc create deployment event-origin --image quay.io/openshift-knative/showcase
```

7. 通过检查以下命令生成的输出来检查是否正确映射了控制器：

```
$ kn source apiserver describe <source_name>
```

输出示例

```
Name:               mysource
Namespace:          default
Annotations:        sources.knative.dev/creator=developer,
sources.knative.dev/lastModifier=developer
Age:                3m
ServiceAccountName:  events-sa
Mode:               Resource
Sink:
  Name:       default
  Namespace:  default
  Kind:       Broker (eventing.knative.dev/v1)
Resources:
  Kind:       event (v1)
  Controller:  false
Conditions:
  OK TYPE                  AGE REASON
  ++ Ready                 3m
  ++ Deployed               3m
  ++ SinkProvided           3m
  ++ SufficientPermissions     3m
  ++ EventTypesProvided       3m
```

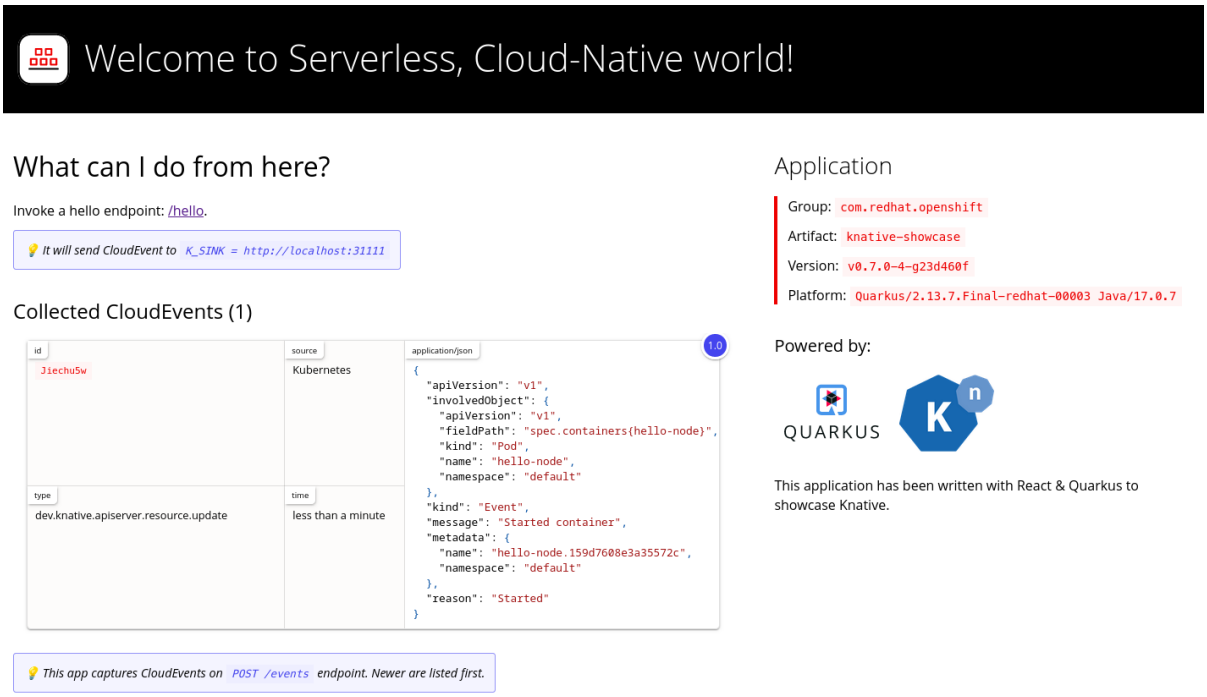
验证

要验证 Kubernetes 事件是否已发送到 Knative，请查看 event-display 日志或使用 Web 浏览器查看事件。

- 要在网页浏览器中查看事件，请使用以下命令返回的链接：

```
$ kn service describe event-display -o url
```

图 4.1. 浏览器页面示例



- 另外，要在终端中查看日志，请输入以下命令来查看 pod 的 event-display 日志：

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

输出示例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.apiserver.resource.update
  datacontenttype: application/json
  ...
Data,
  {
    "apiVersion": "v1",
    "involvedObject": {
      "apiVersion": "v1",
      "fieldPath": "spec.containers{event-origin}",
      "kind": "Pod",
      "name": "event-origin",
      "namespace": "default",
       .....
    },
    "kind": "Event",
    "message": "Started container",
    "metadata": {
      "name": "event-origin.159d7608e3a3572c",
      "namespace": "default",
      ....
    },
```

```
    "reason": "Started",
    ...
  }
```

删除 API 服务器源

1. 删除触发器：

```
$ kn trigger delete <trigger_name>
```

2. 删除事件源：

```
$ kn source apiserver delete <source_name>
```

3. 删除服务帐户、集群角色和集群绑定：

```
$ oc delete -f authentication.yaml
```

4.1.5. 使用 Knative CLI 创建 ping 源

您可以使用 **kn source ping create** 命令，通过 Knative (**kn**) CLI 创建 ping 源。使用 Knative CLI 创建事件源提供了比直接修改 YAML 文件更精简且直观的用户界面。

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。
- 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。
- 可选： 如果要使用此流程验证步骤，请安装 OpenShift CLI (**oc**) 。

流程

1. 要验证 ping 源是否可以工作，请创建一个简单的 Knative 服务，在服务日志中转储传入的信息：

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

2. 对于您要请求的每一组 ping 事件，请在与事件消费者相同的命名空间中创建一个 ping 源：

```
$ kn source ping create test-ping-source \
    --schedule "*/2 * * * *" \
    --data '{"message": "Hello world!"}' \
    --sink ksvc:event-display
```

3. 输入以下命令并检查输出，检查是否正确映射了控制器：

```
$ kn source ping describe test-ping-source
```

输出示例

```
Name:         test-ping-source
Namespace:    default
Annotations:  sources.knative.dev/creator=developer,
sources.knative.dev/lastModifier=developer
Age:          15s
Schedule:     */2 * * * *
Data:         {"message": "Hello world!"}

Sink:
  Name:       event-display
  Namespace:  default
  Resource:   Service (serving.knative.dev/v1)

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                8s
  ++ Deployed             8s
  ++ SinkProvided         15s
  ++ ValidSchedule        15s
  ++ EventTypeProvided    15s
  ++ ResourcesCorrect     15s
```

验证

您可以通过查看 sink pod 的日志来验证 Kubernetes 事件是否已发送到 Knative 事件。

默认情况下，如果在 60 秒内都没有流量，Knative 服务会终止其 Pod。本指南中演示的示例创建了一个 ping 源，每 2 分钟发送一条消息，因此每个消息都应该在新创建的 pod 中观察到。

1. 查看新创建的 pod：

```
$ watch oc get pods
```

2. 使用 Ctrl+C 取消查看 pod，然后查看所创建 pod 的日志：

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

输出示例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.sources.ping
  source: /apis/v1/namespaces/default/pingsources/test-ping-source
  id: 99e4f4f6-08ff-4bff-acf1-47f61ded68c9
  time: 2020-04-07T16:16:00.000601161Z
  datacontenttype: application/json
Data,
  {
    "message": "Hello world!"
  }
```

删除 ping 源

- 删除 ping 源：

```
$ kn delete pingsources.sources.knative.dev <ping_source_name>
```

4.1.6. 使用 Knative CLI 创建 Apache Kafka 事件源

您可以使用 **kn source kafka create** 命令，使用 Knative (**kn**) CLI 创建 Kafka 源。使用 Knative CLI 创建事件源提供了比直接修改 YAML 文件更精简且直观的用户界面。

先决条件

- OpenShift Serverless Operator、Knative Eventing、Knative Serving 和 **KnativeKafka** 自定义资源（CR）已安装在集群中。
- 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。
- 您可以访问 Red Hat AMQ Streams（Kafka）集群，该集群会生成您要导入的 Kafka 信息。
- 已安装 Knative (**kn**) CLI。
- 可选：如果您想要使用此流程中的验证步骤，已安装 OpenShift CLI (**oc**)。

流程

1. 要验证 Kafka 事件源是否可以工作，请创建一个 Knative 服务，在服务日志中转储传入的事件：

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

2. 创建 **KafkaSource** CR：

```
$ kn source kafka create <kafka_source_name> \
    --servers <cluster_kafka_bootstrap>.kafka.svc:9092 \
    --topics <topic_name> --consumergroup my-consumer-group \
    --sink event-display
```

注意

将此命令中的占位符值替换为源名称、引导服务器和主题的值。

--servers、**--topics** 和 **--consumergroup** 选项指定到 Kafka 集群的连接参数。**--consumergroup** 选项是可选的。

3. 可选：查看您创建的 **KafkaSource** CR 的详情：

```
$ kn source kafka describe <kafka_source_name>
```

输出示例

```
Name:            example-kafka-source
```

```
Namespace:         kafka
Age:               1h
BootstrapServers:  example-cluster-kafka-bootstrap.kafka.svc:9092
Topics:            example-topic
ConsumerGroup:     example-consumer-group

Sink:
  Name:       event-display
  Namespace:  default
  Resource:   Service (serving.knative.dev/v1)

Conditions:
  OK TYPE           AGE REASON
  ++ Ready          1h
  ++ Deployed       1h
  ++ SinkProvided   1h
```

验证步骤

1. 触发 Kafka 实例将信息发送到主题：

```
$ oc -n kafka run kafka-producer \
    -ti --image=quay.io/strimzi/kafka:latest-kafka-2.7.0 --rm=true \
    --restart=Never -- bin/kafka-console-producer.sh \
    --broker-list <cluster_kafka_bootstrap>:9092 --topic my-topic
```

在提示符后输入信息。这个命令假设：

- Kafka 集群安装在 **kafka** 命名空间中。
- **KafkaSource** 对象已被配置为使用 **my-topic** 主题。

2. 通过查看日志来验证消息是否显示：

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

输出示例

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.kafka.event
  source: /apis/v1/namespaces/default/kafkasources/example-kafka-source#example-topic
  subject: partition:46#0
  id: partition:46/offset:0
  time: 2021-03-10T11:21:49.4Z
Extensions,
  traceparent: 00-161ff3815727d8755848ec01c866d1cd-7ff3916c44334678-00
Data,
  Hello!
```

第 5 章 KNATIVE FUNCTIONS CLI 命令

5.1. KN 功能命令

5.1.1. 使用 Knative CLI 创建功能

您可以为命令行上的标志指定功能的路径、运行时、模板和镜像 registry，或者使用 **-c** 标志在终端中启动交互式体验。

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

- 创建功能项目：

```
$ kn func create -r <repository> -l <runtime> -t <template> <path>
```

  - 可接受的运行时值包括 **quarkus**、**node**、**typescript**、**go**、**python**、**springboot** 和 **rust**。
  - 可接受的模板值包括 **http** 和 **cloudevents**。

    示例命令

```
$ kn func create -l typescript -t cloudevents examplefunc
```

    输出示例

```
Created typescript function in /home/user/demo/examplefunc
```

  - 或者，您可以指定包含自定义模板的存储库。

    示例命令

```
$ kn func create -r https://github.com/boson-project/templates/ -l node -t hello-world examplefunc
```

    输出示例

```
Created node function in /home/user/demo/examplefunc
```

5.1.2. 在本地运行一个函数

您可以使用 **kn func run** 命令在当前目录中本地运行函数，或者在 **--path** 标志指定的目录中运行。如果您运行的函数之前没有被构建，或者项目文件自上次构建以来已修改过，**kn func run** 命令将在运行它前构建该函数。

在当前目录中运行函数的命令示例

```
$ kn func run
```

在指定为路径的目录中运行函数的示例

```
$ kn func run --path=<directory_path>
```

您也可以在运行该函数前强制重建现有镜像，即使项目文件没有更改项目文件，则使用 **--build** 标志：

使用 build 标记的 run 命令示例

```
$ kn func run --build
```

如果将 **build** 标志设置为 false，这将禁用构建镜像，并使用之前构建的镜像运行该功能：

使用 build 标记的 run 命令示例

```
$ kn func run --build=false
```

您可以使用 help 命令了解更多有关 **kn func run** 命令选项的信息：

构建 help 命令

```
$ kn func help run
```

5.1.3. 构建函数

在运行功能前，您必须构建 function 项目。如果使用 **kn func run** 命令，则该函数会自动构建。但是，您可以使用 **kn func build** 命令在不运行的情况下构建函数，这对于高级用户或调试场景非常有用。

kn func build 命令创建可在您的计算机或 OpenShift Container Platform 集群中运行的 OCI 容器镜像。此命令使用功能项目名称和镜像 registry 名称为您的功能构建完全限定镜像名称。

5.1.3.1. 镜像容器类型

默认情况下，**kn func build** 使用 Red Hat Source-to-Image (S2I) 技术创建一个容器镜像。

使用 Red Hat Source-to-Image (S2I) 的 build 命令示例.

```
$ kn func build
```

5.1.3.2. 镜像 registry 类型

OpenShift Container Registry 默认用作存储功能镜像的镜像 registry。

使用 OpenShift Container Registry 的 build 命令示例

```
$ kn func build
```

输出示例

```
Building function image
Function image has been built, image: registry.redhat.io/example/example-function:latest
```

您可以使用 **--registry** 标志覆盖使用 OpenShift Container Registry 作为默认镜像 registry：

build 命令覆盖 OpenShift Container Registry 以使用 quay.io

```
$ kn func build --registry quay.io/username
```

输出示例

```
Building function image
Function image has been built, image: quay.io/username/example-function:latest
```

5.1.3.3. push 标记

您可以将 **--push** 标志添加到 **kn func build** 命令中，以便在成功构建后自动推送功能镜像：

使用 OpenShift Container Registry 的 build 命令示例

```
$ kn func build --push
```

5.1.3.4. help 命令

您可以使用 help 命令了解更多有关 **kn func build** 命令选项的信息：

构建 help 命令

```
$ kn func help build
```

5.1.4. 部署功能

您可以使用 **kn func deploy** 命令将功能部署到集群中，作为 Knative 服务。如果已经部署了目标功能，则会使用推送到容器镜像 registry 的新容器镜像进行更新，并更新 Knative 服务。

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。
- 您必须已创建并初始化要部署的功能。

流程

- 部署功能：

```
$ kn func deploy [-n <namespace> -p <path> -i <image>]
```

输出示例

```
Function deployed at: http://func.example.com
```

- 如果没有指定 **namespace**，则该函数部署到当前命名空间中。
- 此函数从当前目录中部署，除非指定了 **path**。
- Knative 服务名称派生自项目名称，无法使用此命令进行更改。

注意

您可以使用 **Developer** 视角的 **+Add** 视图中的 **Import from Git** 或 **Create Serverless Function** 使用 Git 存储库 URL 创建无服务器功能。

5.1.5. 列出现有功能

您可以使用 **kn func list** 列出现有功能。如果要列出部署为 Knative 服务的功能，也可以使用 **kn service list**。

流程

- 列出现有功能：

```
$ kn func list [-n <namespace> -p <path>]
```

输出示例

```
NAME            NAMESPACE  RUNTIME  URL
READY
example-function  default    node     http://example-function.default.apps.ci-ln-g9f36hb-
d5d6b.origin-ci-int-aws.dev.rhcloud.com  True
```

- 列出部署为 Knative 服务的功能：

```
$ kn service list -n <namespace>
```

输出示例

```
NAME            URL                                                                LATEST
AGE   CONDITIONS   READY   REASON
example-function   http://example-function.default.apps.ci-ln-g9f36hb-d5d6b.origin-ci-int-
aws.dev.rhcloud.com   example-function-gzl4c   16m   3 OK / 3     True
```

5.1.6. 描述函数

kn func info 命令输出有关已部署功能的信息，如功能名称、镜像、命名空间、Knative 服务信息、路由信息和事件订阅。

流程

- 描述函数：

```
$ kn func info [-f <format> -n <namespace> -p <path>]
```

示例命令

```
$ kn func info -p function/example-function
```

输出示例

```
Function name:
  example-function
Function is built in image:
  docker.io/user/example-function:latest
Function is deployed as Knative Service:
  example-function
Function is deployed in namespace:
  default
Routes:
  http://example-function.default.apps.ci-ln-g9f36hb-d5d6b.origin-ci-int-aws.dev.rhcloud.com
```

5.1.7. 使用测试事件调用部署的功能

您可以使用 **kn func invoke** CLI 命令发送测试请求，在本地或 OpenShift Container Platform 集群中调用功能。您可以使用此命令测试功能是否正常工作并且能够正确接收事件。本地调用函数可用于在功能开发期间进行快速测试。在测试与生产环境更接近的测试时，在集群中调用函数非常有用。

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。
- 您必须已部署了要调用的功能。

流程

- 调用函数：

```
$ kn func invoke
```

  - **kn func invoke** 命令仅在当前运行本地容器镜像时或在集群中部署功能时才有效。
  - **kn func invoke** 命令默认在本地目录上执行，并假定此目录是一个功能项目。

5.1.7.1. kn func 调用可选参数

您可以使用以下 **kn func invoke** CL 命令标记为请求指定可选参数。

标记	描述
-t,--target	指定调用函数的目标实例，如 **local** 或 **remote** 或 **https://staging.example.com/**。默认目标是 **local**。
-f,--format	指定消息的格式，如 **cloudevent** 或 **http**。
--id	指定请求的唯一字符串标识符。
-n,--namespace	指定集群上的命名空间。
--source	指定请求的发件人名称。这与 CloudEvent **source** 属性对应。
--type	指定请求类型，例如 **boson.fn**。这与 CloudEvent **type** 属性对应。
--data	指定请求的内容。对于 CloudEvent 请求，这是 CloudEvent **data** 属性。
--file	指定包含要发送数据的本地文件的路径。
--content-type	指定请求的 MIME 内容类型。
-p,--path	指定项目目录的路径。
-c,--confirm	启用系统提示，以交互方式确认所有选项。
-v,--verbose	启用打印详细输出。
-h,--help	输出有关使用 **kn func invoke** 的信息。

5.1.7.1.1. 主要参数

以下参数定义 **kn func invoke** 命令的主要属性：

事件目标 (-t,--target)

调用函数的目标实例。接受 **local** 值用于本地部署的函数、**remote** 值用于远程部署函数，或一个 URL 用于一个任意的端点。如果没有指定目标，则默认为 **local**。

事件消息格式 (-f,--format)

事件的消息格式，如 **http** 或 **cloudevent**。默认为创建函数时使用的模板格式。

事件类型 (--type)

发送的事件类型。您可以查找有关各个事件制作者文档中设置的 **type** 参数的信息。例如，API 服务器源可能会将生成的事件的 **type** 参数设置为 **dev.knative.apiserver.resource.update**。

事件源 (--source)

生成该事件的唯一事件源。这可能是事件源的 URI，如 **https://10.96.0.1/** 或事件源的名称。

事件 ID (--id)

由事件制作者创建的随机唯一 ID。

事件数据 (--data)

允许您为 **kn func invoke** 命令发送的事件指定 **data** 值。例如，您可以指定一个 **--data** 值，如 **"Hello World"**，以便事件包含此数据字符串。默认情况下，**kn func invoke** 创建的事件中不包含任何数据。

注意

已部署到集群的功能可以对现有事件源的事件响应，该源提供属性（如 **source** 和 **type**）的值。这些事件通常具有 JSON 格式的 **data** 值，用于捕获事件的特定域上下文。通过使用本文档中介绍的 CLI 标志，开发人员可以模拟这些事件以进行本地测试。

您还可以使用 **--file** 标志发送事件数据，以提供包含事件数据的本地文件。在这种情况下，使用 **--content-type** 指定内容类型。

数据内容类型 (--content-type)

如果您使用 **--data** 标志为事件添加数据，您可以使用 **--content-type** 标志指定事件传输的数据类型。在上例中，数据是纯文本，因此您可以指定 **kn func call --data "Hello world!" --content-type "text/plain** "。

5.1.7.1.2. 示例命令

这是 **kn func invoke** 命令的一般调用：

```
$ kn func invoke --type <event_type> --source <event_source> --data <event_data> --content-type
<content_type> --id <event_ID> --format <format> --namespace <namespace>
```

例如，要发送 "Hello world!" 事件，您可以运行：

```
$ kn func invoke --type ping --source example-ping --data "Hello world!" --content-type "text/plain" --
id example-ID --format http --namespace my-ns
```

5.1.7.1.2.1. 使用数据指定文件

要指定磁盘上包含事件数据的文件，请使用 **--file** 和 **--content-type** 标志：

```
$ kn func invoke --file <path> --content-type <content-type>
```

例如，要发送存储在 **test.json** 文件中的 JSON 数据，请使用以下命令：

```
$ kn func invoke --file ./test.json --content-type application/json
```

5.1.7.1.2.2. 指定功能项目

您可以使用 **--path** 标志指定到功能项目的路径：

```
$ kn func invoke --path <path_to_function>
```

例如，要使用位于 **./example/example-function** 目录中的功能项目，请使用以下命令：

```
$ kn func invoke --path ./example/example-function
```

5.1.7.1.2.3. 指定部署目标功能的位置

默认情况下，**kn func invoke** 作为功能本地部署的目标：

```
$ kn func invoke
```

要使用不同的部署，请使用 **--target** 标志：

```
$ kn func invoke --target <target>
```

例如，要使用在集群中部署的功能，请使用 **--target remote** 标志：

```
$ kn func invoke --target remote
```

要使用在任意 URL 中部署的功能，请使用 **--target <URL>** 标志：

```
$ kn func invoke --target "https://my-event-broker.example.com"
```

您可以明确以本地部署为目标。在这种情况下，如果这个功能没有在本地运行，命令会失败：

```
$ kn func invoke --target local
```

5.1.8. 删除函数

您可以使用 **kn func delete** 命令删除功能。当不再需要某个函数时，这很有用，并有助于在集群中保存资源。

流程

- 删除函数：

```
$ kn func delete [<function_name> -n <namespace> -p <path>]
```

  - 如果没有指定要删除的功能的名称或路径，则会搜索当前目录以查找用于决定要删除的功能的 **func.yaml** 文件。
  - 如果没有指定命名空间，则默认为 **func.yaml** 文件中的 **namespace** 值。