Red Hat OpenShift Serverless 1.30

Knative CLI

Knative Functions, Serving 및 Eventing에 대한 CLI 명령 개요

Last Updated: 2023-09-20

Red Hat OpenShift Serverless 1.30 Knative CLI

Knative Functions, Serving 및 Eventing에 대한 CLI 명령 개요

법적 공지

초록

이 문서에서는 Knative Functions, Serving 및 Eventing에 사용할 수 있는 CLI 명령에 대한 개요를 제공합니다. 또한 Knative CLI 구성 및 플러그인 사용에 대한 정보도 제공합니다.

차례

1장. KNATIVE SERVING CLI 명령

1.1. KN SERVICE 명령

다음 명령을 사용하여 Knative 서비스를 생성하고 관리할 수 있습니다.

1.1.1. Knative CLI를 사용하여 서버리스 애플리케이션 생성

Knative(**kn**) CLI를 사용하여 서버리스 애플리케이션을 생성하면 YAML 파일을 직접 수정하는 것보다 간소화되고 직관적인 사용자 인터페이스가 제공됩니다. **kn service create** 명령을 사용하여 기본 서버리스 애플리케이션을 생성할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

- Knative 서비스를 생성합니다.

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

  다음과 같습니다.

  - **--image** 는 애플리케이션의 이미지 URI입니다.
  - **--tag** 는 서비스와 함께 생성된 초기 버전에 태그를 추가하는 데 사용할 수 있는 선택적 플래그입니다.

명령 예

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

출력 예

```
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. Knative CLI를 사용하여 서버리스 애플리케이션 업데이트

서비스를 단계적으로 구축할 때 명령줄에서 대화형 세션에 **kn service update** 명령을 사용할 수 있습니다. **kn service apply** 명령과 달리 **kn service update** 명령을 사용하는 경우 Knative 서비스의 전체 구성이 아닌 업데이트하려는 변경 사항만 지정해야 합니다.

명령 예

- 새 환경 변수를 추가하여 서비스를 업데이트합니다.

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

- 새 포트를 추가하여 서비스를 업데이트합니다.

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

- 새 요청 및 제한 매개변수를 추가하여 서비스를 업데이트합니다.

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

- **latest** 태그를 개정 버전에 할당합니다.

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

- 서비스의 최신 **READY** 버전에 대한 태그를 **testing**에서 **staging**으로 업데이트합니다.

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

- 트래픽의 10%를 수신하는 버전에 **test** 태그를 추가하고 나머지 트래픽을 서비스의 최신 **READY** 버전으로 전송합니다.

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

1.1.3. 서비스 선언 적용

kn service apply 명령을 사용하여 Knative 서비스를 선언적으로 구성할 수 있습니다. 서비스가 존재하지 않으면 기존 서비스가 변경된 옵션으로 업데이트됩니다.

kn service apply 명령은 사용자가 일반적으로 단일 명령으로 서비스 상태를 완전히 지정하여 대상 상태를 선언하려는 셸 스크립트 또는 지속적 통합 파이프라인에 특히 유용합니다.

kn service apply를 사용하는 경우 Knative 서비스에 대한 전체 구성을 제공해야 합니다. 이 동작은 업데이트하려는 옵션을 명령에서 지정하기만 하면 되는 **kn service update** 명령과 다릅니다.

명령 예

- 서비스를 생성합니다.

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

- 서비스에 환경 변수를 추가합니다.

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

- JSON 또는 YAML 파일에서 서비스 선언을 읽습니다.

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

1.1.4. Knative CLI를 사용하여 서버리스 애플리케이션 설명

kn service describe 명령을 사용하여 Knative 서비스를 설명할 수 있습니다.

명령 예

- 서비스를 설명합니다.

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

--verbose 플래그는 선택 사항이지만 자세한 설명을 제공하기 위해 포함할 수 있습니다. 일반 출력과 자세한 출력의 차이점은 다음 예에 표시됩니다.

--verbose 플래그를 사용하지 않는 출력 예

```
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
```

--verbose 플래그를 사용하는 출력 예

```
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
```

- YAML 형식으로 서비스를 설명합니다.

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

- JSON 형식으로 서비스를 설명합니다.

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

- 서비스 URL만 인쇄합니다.

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

1.2. KN SERVICE 명령 오프라인 모드

1.2.1. Knative CLI 오프라인 모드 정보

kn service 명령을 실행하면 변경 사항이 즉시 클러스터로 전파됩니다. 그러나 대안으로 오프라인 모드에서 **kn service** 명령을 실행할 수 있습니다. 오프라인 모드로 서비스를 생성하면 클러스터에서 변경 사항이 발생하지 않으며 대신 로컬 시스템에 서비스 설명자 파일이 생성됩니다.

중요

Knative CLI의 오프라인 모드는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

설명자 파일이 생성되면 수동으로 수정하고 버전 제어 시스템에서 추적할 수 있습니다. 설명자 파일에서 **kn service create -f**,**kn service apply -f** 또는 **oc apply -f** 명령을 사용하여 변경 사항을 클러스터에 전파할 수도 있습니다.

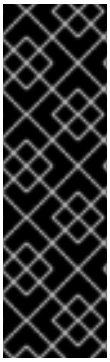
오프라인 모드에는 여러 방식이 있습니다.

- 클러스터 변경에 사용하기 전에 설명자 파일을 수동으로 수정할 수 있습니다.
- 버전 관리 시스템에서 서비스의 설명자 파일을 로컬로 추적할 수 있습니다. 이를 통해 CI(지속적 통합) 파이프라인, 개발 환경 또는 데모와 같이 대상 클러스터 이외의 위치에서 설명자 파일을 재사용할 수 있습니다.
- 생성된 설명자 파일을 검사하여 Knative 서비스에 대해 확인할 수 있습니다. 특히 결과 서비스가 **kn** 명령에 전달된 다양한 인수에 영향을 받는 방법을 확인할 수 있습니다.

오프라인 모드는 속도가 빠르고 클러스터에 연결할 필요가 없다는 장점이 있습니다. 그러나 오프라인 모드에서는 서버 측 유효성 검사가 없습니다. 따라서 서비스 이름이 고유하거나 지정된 이미지를 가져올 수 있는지 등을 확인할 수 없습니다.

1.2.2. 오프라인 모드를 사용하여 서비스 생성

클러스터에서 변경 사항이 발생하지 않도록 **kn service** 명령을 오프라인 모드에서 실행할 수 있으며 대신 로컬 시스템에 서비스 설명자 파일이 생성됩니다. 설명자 파일이 생성되면 파일을 수정한 후 클러스터의 변경 사항을 전파할 수 있습니다.



중요

Knative CLI의 오프라인 모드는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

프로세스

1. 오프라인 모드에서 로컬 Knative 서비스 설명자 파일을 생성합니다.

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

출력 예

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

- **--target ./** 플래그는 오프라인 모드를 활성화하고 새 디렉터리 트리를 저장하는 디렉토리로 **./**를 지정합니다.
  기존 디렉터리를 지정하지 않고 **--target my-service.yaml**과 같은 파일 이름을 사용하면 디렉터리 트리가 생성되지 않습니다. 대신 서비스 설명자 파일 **my-service.yaml**만 현재 디렉터리에 생성됩니다.

  파일 이름에는 **.yaml**, **.yml**, 또는 **.json** 확장을 사용할 수 있습니다. **.json**을 선택하면 JSON 형식으로 서비스 설명자 파일을 생성합니다.

- **--namespace test** 옵션은 새 서비스를 **test** 네임스페이스에 배치합니다.
  --namespace를 사용하지 않고 OpenShift Container Platform 클러스터에 로그인한 경우 현재 네임스페이스에 설명자 파일이 생성됩니다. 그렇지 않으면 설명자 파일이 **default** 네임스페이스에 생성됩니다.

2. 생성된 디렉터리 구조를 확인합니다.

```
$ tree ./
```

출력 예

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

2 directories, 1 file
```

- **--target**에서 지정된 현재 **./** 디렉터리에는 지정된 네임스페이스를 바탕으로 이름이 지정된 **test/** 디렉터리가 포함되어 있습니다.
- **test/** 디렉터리에는 리소스 유형의 이름에 따라 이름이 지정된 **ksvc** 디렉터리가 포함되어 있습니다.
- **ksvc** 디렉터리에는 지정된 서비스 이름에 따라 이름이 지정된 설명자 파일 **showcase.yaml**이 포함되어 있습니다.

3. 생성된 서비스 기술자 파일을 확인합니다.

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

출력 예

```
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. 새 서비스에 대한 정보를 나열합니다.

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

출력 예

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

```
Revisions:

Conditions:
 OK TYPE    AGE REASON
```

- **target ./** 옵션은 네임스페이스 하위 디렉터리를 포함하는 디렉터리 구조의 루트 디렉터리를 지정합니다.
  또는 **--target** 옵션을 사용하여 YAML 또는 JSON 파일 이름을 직접 지정할 수 있습니다. 허용된 파일 확장자는 **.yaml**, **.yml**, **.json**입니다.

- **--namespace** 옵션은 필요한 서비스 기술자 파일을 포함하는 하위 디렉터리를 **kn**와 통신하는 네임스페이스를 지정합니다.
  --namespace를 사용하지 않고 OpenShift Container Platform 클러스터에 로그인한 경우 **kn**은 현재 네임스페이스를 바탕으로 이름이 지정된 하위 디렉터리에서 서비스를 검색합니다. 그렇지 않으면 **kn**은 **default/** 하위 디렉터리에서 검색합니다.

5. 서비스 설명자 파일을 사용하여 클러스터에 서비스를 생성합니다.

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

출력 예

```
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 컨테이너 명령

다음 명령을 사용하여 Knative 서비스 사양에서 여러 컨테이너를 생성하고 관리할 수 있습니다.

1.3.1. Knative 클라이언트 멀티컨테이너 지원

kn container add 명령을 사용하여 YAML 컨테이너 사양을 표준 출력에 출력할 수 있습니다. 이 명령은 다른 표준 **kn** 플래그와 함께 사용하여 정의를 생성할 수 있으므로 멀티컨테이너 사용 사례에 유용합니다.

kn container add 명령은 **kn service create** 명령과 함께 사용할 수 있도록 지원되는 모든 컨테이너 관련 플래그를 허용합니다. **kn container add** 명령은 UNIX 파이프(|)를 사용하여 한 번에 여러 컨테이너 정의를 생성하여 연결할 수도 있습니다.

명령 예

- 이미지에서 컨테이너를 추가하고 표준 출력에 출력합니다.

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

명령 예

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

출력 예

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

- 두 개의 **kn 컨테이너 add** 명령을 함께 연결한 다음 **kn service create** 명령에 전달하여 두 개의 컨테이너가 있는 Knative 서비스를 생성합니다.

```
$ 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 - kn 이 YAML 파일 대신 파이프 입력을 읽는 특수 케이스를 지정합니다.

명령 예

```
$ 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 플래그는 YAML 파일의 경로도 허용할 수 있습니다.

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

명령 예

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

1.4. KN DOMAIN 명령

다음 명령을 사용하여 도메인 매핑을 생성하고 관리할 수 있습니다.

1.4.1. Knative CLI를 사용하여 사용자 정의 도메인 매핑 생성

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- Knative 서비스 또는 경로를 생성했으며 CR에 매핑할 사용자 정의 도메인을 제어할 수 있습니다.

참고

사용자 정의 도메인에서 OpenShift Container Platform 클러스터의 DNS를 가리켜야 합니다.

- Knative(**kn**) CLI가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

- 현재 네임스페이스의 CR에 도메인을 매핑합니다.

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

명령 예

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

--ref 플래그는 도메인 매핑을 위해 주소 지정 가능한 대상 CR을 지정합니다.

--ref 플래그를 사용할 때 접두사가 지정되어 있지 않은 경우 대상이 현재 네임스페이스의 Knative 서비스라고 가정합니다.

- 지정된 네임스페이스의 Knative 서비스에 도메인을 매핑합니다.

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

명령 예

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

- 도메인을 Knative 경로에 매핑합니다.

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

명령 예

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

1.4.2. Knative CLI를 사용하여 사용자 정의 도메인 매핑 관리

DomainMapping CR(사용자 정의 리소스)을 생성한 후 Knative(**kn**) CLI를 사용하여 기존 CR을 나열하거나, 기존 CR에 대한 정보를 보거나, CR을 업데이트하거나, CR을 삭제할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- 하나 이상의 **DomainMapping** CR을 생성했습니다.
- Knative (**kn**) CLI 툴을 설치했습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

프로세스

- 기존 **DomainMapping** CR을 나열합니다.

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

- 기존 **DomainMapping** CR 정보를 표시합니다.

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

- 새 대상을 참조하도록 **DomainMapping** CR을 업데이트합니다.

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

- **DomainMapping** CR을 삭제합니다.

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

2장. KNATIVE CLI 구성

config.yaml 구성 파일을 생성하여 Knative (**kn**) CLI 설정을 사용자 지정할 수 있습니다. **--config** 플래그를 사용하여 이 구성을 지정할 수 있습니다. 지정하지 않으면 기본 위치에서 구성을 가져옵니다. 기본 구성 위치는 XDG 기본 디렉터리 사양을 준수하고 UNIX 시스템 및 Windows 시스템에 따라 다릅니다.

UNIX 시스템의 경우:

- **XDG_CONFIG_HOME** 환경 변수가 설정된 경우 Knative(**kn**) CLI에서 찾는 기본 구성 위치는 **$XDG_CONFIG_HOME/kn** 입니다.
- **XDG_CONFIG_HOME** 환경 변수가 설정되지 않은 경우 Knative(**kn**) CLI는 사용자의 홈 디렉터리에서 **$HOME/.config/kn/config.yaml** 에서 구성을 찾습니다.

Windows 시스템의 경우 기본 Knative (**kn**) CLI 구성 위치는 **%APPDATA%\kn** 입니다.

설정 파일 예

```
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) Knative(**kn**) CLI에서 **PATH** 환경 변수에서 플러그인을 찾을지 여부를 지정합니다. 이는 부울 구성 옵션입니다. 기본값은 **false**입니다.

(2) Knative(**kn**) CLI에서 플러그인을 찾는 디렉터리를 지정합니다. 기본 경로는 이전에 설명한 대로 운영 체제에 따라 다릅니다. 이는 사용자에게 표시되는 모든 디렉터리일 수 있습니다.

(3) **sink-mappings** 사양은 Knative(**kn**) CLI 명령과 함께 **--sink** 플래그를 사용할 때 사용되는 Kubernetes 주소 지정 가능 리소스를 정의합니다.

(4) 싱크를 설명하는 데 사용할 접두사입니다. 서비스, **채널**, **브로커** 의 **svc** 는 Knative(**kn**) CLI의 사전 정의된 접두사입니다.

(5) Kubernetes 리소스의 API 그룹입니다.

(6) Kubernetes 리소스의 버전입니다.

(7) Kubernetes 리소스 유형의 복수형 이름입니다. 예: **services** 또는 **brokers**

3장. KNATIVE CLI 플러그인

Knative(**kn**) CLI는 플러그인 사용을 지원하므로 사용자 정의 명령 및 코어 배포에 포함되지 않은 기타 공유 명령을 추가하여 **kn** 설치 기능을 확장할 수 있습니다. Knative(**kn**) CLI 플러그인은 기본 **kn** 기능과 동일한 방식으로 사용됩니다.

현재 Red Hat은 **kn-source-kafka** 플러그인과 **kn-event** 플러그인을 지원합니다.

중요

kn-event 플러그인은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

3.1. KN-EVENT 플러그인을 사용하여 이벤트 빌드

kn event build 명령의 빌더와 같은 인터페이스를 사용하여 이벤트를 빌드할 수 있습니다. 그런 다음 나중에 해당 이벤트를 보내거나 다른 컨텍스트에서 사용할 수 있습니다.

사전 요구 사항

- Knative(**kn**) CLI가 설치되어 있습니다.

프로세스

- 이벤트를 빌드합니다.

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

  다음과 같습니다.

  - **--field** 플래그는 이벤트에 필드-값 쌍으로 데이터를 추가합니다. 여러 번 사용할 수 있습니다.
  - **--type** 플래그를 사용하면 이벤트 유형을 지정하는 문자열을 지정할 수 있습니다.
  - **--id** 플래그는 이벤트의 ID를 지정합니다.
  - **json** 또는 **yaml** 인수를 **--output** 플래그와 함께 사용하여 이벤트의 출력 형식을 변경할 수 있습니다.
    이러한 플래그는 모두 선택 사항입니다.

    간단한 이벤트 만들기

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

    YAML 형식의 결과 이벤트

```
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
```

샘플 트랜잭션 이벤트 빌드

```
$ 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 형식의 결과 이벤트

```
{
  "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. KN-EVENT 플러그인을 사용하여 이벤트 전송

kn event send 명령을 사용하여 이벤트를 보낼 수 있습니다. 이벤트는 공개적으로 사용 가능한 주소로 보내거나 Kubernetes 서비스와 Knative 서비스, 브로커 및 채널과 같은 클러스터 내부의 주소 지정 가능한 리소스로 보낼 수 있습니다. 명령은 **kn event build** 명령과 동일한 빌더와 같은 인터페이스를 사용합니다.

사전 요구 사항

- Knative(**kn**) CLI가 설치되어 있습니다.

프로세스

- 이벤트를 보냅니다.
  -

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

다음과 같습니다.

- **--field** 플래그는 이벤트에 필드-값 쌍으로 데이터를 추가합니다. 여러 번 사용할 수 있습니다.
- **--type** 플래그를 사용하면 이벤트 유형을 지정하는 문자열을 지정할 수 있습니다.
- **--id** 플래그는 이벤트의 ID를 지정합니다.
- 공개적으로 액세스 가능한 대상으로 이벤트를 보내는 경우 **--to-url** 플래그를 사용하여 URL을 지정합니다.
- 이벤트를 클러스터 내 Kubernetes 리소스로 보내는 경우 **--to** 플래그를 사용하여 대상을 지정합니다.
  - < **Kind>:<ApiVersion>:<name** > 형식을 사용하여 Kubernetes 리소스를 지정합니다.
- **--namespace** 플래그는 네임스페이스를 지정합니다. 생략하면 네임스페이스가 현재 컨텍스트에서 가져옵니다.
  이러한 플래그는 모두 대상 사양을 제외하고 선택 사항입니다. 이 플래그는 **--to-url** 또는 **--to** 중 하나를 사용해야 합니다.

  다음 예제에서는 이벤트를 URL로 전송하는 방법을 보여줍니다.

  명령 예

```
$ 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/
```

  다음 예제에서는 클러스터 내 리소스로 이벤트를 전송하는 방법을 보여줍니다.

  명령 예

```
$ 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
```

4장. KNATIVE EVENTING CLI 명령

4.1. KN SOURCE 명령

다음 명령을 사용하여 Knative 이벤트 소스를 나열, 생성 및 관리할 수 있습니다.

4.1.1. Knative CLI를 사용하여 사용 가능한 이벤트 소스 유형 나열

kn source list-types CLI 명령을 사용하여 클러스터에서 생성하고 사용할 수 있는 이벤트 소스 유형을 나열할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

프로세스

1. 터미널에서 사용 가능한 이벤트 소스 유형을 나열합니다.

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

출력 예

```
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. 선택 사항: OpenShift Container Platform에서 사용 가능한 이벤트 소스 유형을 YAML 형식으로 나열할 수도 있습니다.

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

4.1.2. Knative CLI sink 플래그

Knative(**kn**) CLI를 사용하여 이벤트 소스를 생성할 때 **--sink** 플래그를 사용하여 해당 리소스에서 이벤트가 전송되는 싱크를 지정할 수 있습니다. 싱크는 다른 리소스에서 들어오는 이벤트를 수신할 수 있는 주소 지정 가능 또는 호출 가능 리소스일 수 있습니다.

다음 예제에서는 싱크로 서비스 **http://event-display.svc.cluster.local** 를 사용하는 싱크 바인딩을 생성합니다.

sink 플래그를 사용하는 명령의 예

```
$ 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** 의 **svc** 는 싱크가 Knative 서비스인지 확인합니다. 기타 기본 싱크 접두사에는 **channel**, 및 **broker**가 포함됩니다.

4.1.3. Knative CLI를 사용하여 컨테이너 소스 생성 및 관리

kn source 컨테이너 명령을 사용하여 Knative(kn) CLI를 사용하여 컨테이너 소스를 생성하고 관리할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다.

컨테이너 소스 생성

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

컨테이너 소스 삭제

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

컨테이너 소스 설명

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

기존 컨테이너 소스 나열

```
$ kn source container list
```

YAML 형식으로 기존 컨테이너 소스 나열

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

컨테이너 소스 업데이트

이 명령은 기존 컨테이너 소스의 이미지 URI를 업데이트합니다.

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

4.1.4. Knative CLI를 사용하여 API 서버 소스 생성

kn source apiserver create 명령을 사용하여 **kn** CLI를 사용하여 API 서버 소스를 생성할 수 있습니다. **kn** CLI를 사용하여 API 서버 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.

- OpenShift CLI(**oc**)가 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

프로세스

기존 서비스 계정을 다시 사용하려면 새 리소스를 생성하는 대신 필요한 권한을 포함하도록 기존 **ServiceAccount** 리소스를 수정할 수 있습니다.

1. 이벤트 소스에 대한 서비스 계정, 역할, 역할 바인딩을 YAML 파일로 만듭니다.

```
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) 이 네임스페이스를 이벤트 소스 설치를 위해 선택한 네임스페이스로 변경합니다.

2. YAML 파일을 적용합니다.

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

3. 이벤트 싱크가 있는 API 서버 소스를 생성합니다. 다음 예에서 싱크는 브로커입니다.

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

4. API 서버 소스가 올바르게 설정되었는지 확인하려면 수신되는 메시지를 로그로 덤프하는 Knative 서비스를 생성합니다.

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

5. 브로커를 이벤트 싱크로 사용한 경우 **default** 브로커의 이벤트를 서비스로 필터링하는 트리거를 생성합니다.

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

6. 기본 네임스페이스에서 Pod를 시작하여 이벤트를 생성합니다.

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

7. 생성된 출력을 다음 명령으로 검사하여 컨트롤러가 올바르게 매핑되는지 확인합니다.

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

출력 예

```
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
```

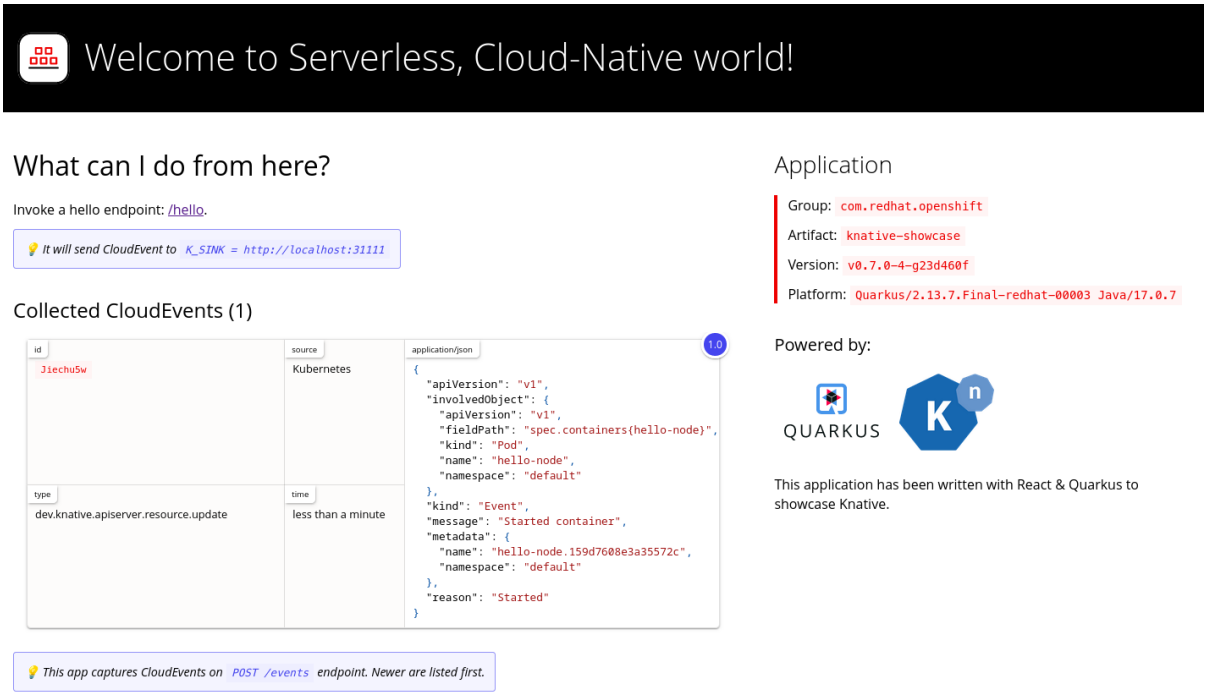
검증

Kubernetes 이벤트가 Knative로 전송되었는지 확인하려면 event-display 로그를 보거나 웹 브라우저를 사용하여 이벤트를 확인합니다.

- 웹 브라우저에서 이벤트를 보려면 다음 명령에서 반환된 링크를 엽니다.

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

그림 4.1. 브라우저 페이지 예



- 또는 터미널에서 로그를 보려면 다음 명령을 입력하여 Pod의 event-display 로그를 확인합니다.

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

출력 예

```
☁  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",
    ...
  }
```

API 서버 소스 삭제

1. 트리거를 삭제합니다.

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

2. 이벤트 소스를 삭제합니다.

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

3. 서비스 계정, 클러스터 역할, 클러스터 바인딩을 삭제합니다.

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

4.1.5. Knative CLI를 사용하여 ping 소스 생성

kn source ping create 명령을 사용하여 Knative(**kn**) CLI를 사용하여 ping 소스를 생성할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Serving 및 Knative Eventing이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 선택 사항: 이 프로세스에 확인 단계를 사용하려면 OpenShift CLI(**oc**)를 설치합니다.

프로세스

1. ping 소스가 작동하는지 확인하려면 수신 메시지를 서비스 로그에 덤프하는 간단한 Knative 서비스를 생성합니다.

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

2. 요청할 각 ping 이벤트 세트에 대해 이벤트 소비자와 동일한 네임스페이스에 ping 소스를 생성합니다.

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

3. 다음 명령을 입력하고 출력을 검사하여 컨트롤러가 올바르게 매핑되는지 확인합니다.

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

출력 예

```
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
```

검증

싱크 Pod의 로그를 보면 Kubernetes 이벤트가 Knative 이벤트 싱크로 전송되었는지 확인할 수 있습니다.

Knative 서비스는 기본적으로 60초 이내에 트래픽이 수신되지 않으면 Pod를 종료합니다. 이 가이드에 표시된 예제에서는 2분마다 메시지를 전송하는 ping 소스를 생성하므로 새로 생성된 Pod에서 각 메시지를 관찰해야 합니다.

1. 새 Pod가 생성되었는지 확인합니다.

```
$ watch oc get pods
```

2. Ctrl+C를 사용하여 Pod를 감시한 다음 생성한 Pod의 로그를 확인합니다.

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

출력 예

```
☁  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!"
  }
```

ping 소스 삭제

- ping 소스를 삭제합니다.

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

4.1.6. Knative CLI를 사용하여 Apache Kafka 이벤트 소스 생성

kn source kafka create 명령을 사용하여 Knative(**kn**) CLI를 사용하여 Kafka 소스를 생성할 수 있습니다. Knative CLI를 사용하여 이벤트 소스를 생성하면 YAML 파일을 직접 수정하는 것보다 더 효율적이고 직관적인 사용자 인터페이스가 제공됩니다.

사전 요구 사항

- OpenShift Serverless Operator, Knative Eventing, Knative Serving, **KnativeKafka** 사용자 정의 리소스(CR가 클러스터에 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 가져오려는 Kafka 메시지를 생성하는 Red Hat AMQ Streams(Kafka) 클러스터에 액세스할 수 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.
- 선택 사항: 이 절차의 확인 단계를 사용하려는 경우 OpenShift CLI(**oc**)를 설치했습니다.

프로세스

1. Kafka 이벤트 소스가 작동하는지 확인하려면 수신 이벤트를 서비스 로그에 덤프하는 Knative 서비스를 생성합니다.

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

2. **KafkaSource** CR을 생성합니다.

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

참고

이 명령의 자리 표시자 값을 소스 이름, 부트스트랩 서버 및 주제의 값으로 바꿉니다.

--servers, **--topics**, **--consumergroup** 옵션은 Kafka 클러스터에 대한 연결 매개 변수를 지정합니다. **--consumergroup** 옵션은 선택 사항입니다.

3. 선택 사항: 생성한 **KafkaSource** CR에 대한 세부 정보를 확인합니다.

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

출력 예

```
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
```

검증 단계

1. Kafka 인스턴스를 트리거하여 메시지를 항목에 보냅니다.

```
$ 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
```

프롬프트에 메시지를 입력합니다. 이 명령은 다음을 가정합니다.

- Kafka 클러스터는 **kafka** 네임스페이스에 설치됩니다.
- **my-topic** 주제를 사용하도록 **KafkaSource** 오브젝트가 구성되어 있습니다.

2. 로그를 보고 메시지가 도착했는지 확인합니다.

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

출력 예

```
☁  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!
```

5장. KNATIVE FUNCTIONS CLI 명령

5.1. KN 함수 명령

5.1.1. Knative CLI를 사용하여 함수 생성

명령줄에서 함수의 경로, 런타임, 템플릿 및 이미지 레지스트리를 지정하거나 **-c** 플래그를 사용하여 터미널에서 대화형 환경을 시작할 수 있습니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

프로세스

- 함수 프로젝트를 생성합니다.

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

  - 허용되는 런타임 값에는 **quarkus**,**node**,**typescript**,**go**,**python**,**springboot**, **rust** 가 포함됩니다.
  - 허용되는 템플릿 값에는 **http** 및 **cloudevents** 가 포함됩니다.

    명령 예

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

    출력 예

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

  - 또는 사용자 지정 템플릿이 포함된 리포지토리를 지정할 수 있습니다.

    명령 예

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

    출력 예

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

5.1.2. 로컬에서 함수 실행

kn func run 명령을 사용하여 현재 디렉터리 또는 **--path** 플래그에서 지정한 디렉터리에서 로컬로 함수를 실행할 수 있습니다. 실행 중인 함수가 이전에 빌드되지 않았거나 프로젝트 파일이 마지막으로 빌드된 이후 수정된 경우 **kn func run** 명령은 기본적으로 함수를 빌드하기 전에 함수를 빌드합니다.

현재 디렉터리에서 함수를 실행하는 명령의 예

```
$ kn func run
```

경로로 지정된 디렉터리에서 함수를 실행하는 명령의 예

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

--build 플래그를 사용하여 프로젝트 파일을 변경하지 않은 경우에도 함수를 실행하기 전에 기존 이미지를 강제로 다시 빌드할 수도 있습니다.

build 플래그를 사용한 run 명령의 예

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

빌드 플래그를 false로 설정하면 이미지 빌드가 비활성화되고 이전에 빌드된 이미지를 사용하여 함수를 실행합니다.

build 플래그를 사용한 run 명령의 예

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

help 명령을 사용하여 **kn func run** 명령 옵션에 대해 자세히 알아볼 수 있습니다.

빌드 도움말 명령

```
$ kn func help run
```

5.1.3. 함수 빌드

함수를 실행하려면 먼저 함수 프로젝트를 빌드해야 합니다. **kn func run** 명령을 사용하는 경우 함수가 자동으로 빌드됩니다. 그러나 **kn func build** 명령을 사용하여 실행하지 않고 함수를 빌드할 수 있으며 고급 사용자 또는 디버깅 시나리오에 유용할 수 있습니다.

kn func build 명령은 컴퓨터 또는 OpenShift Container Platform 클러스터에서 로컬로 실행할 수 있는 OCI 컨테이너 이미지를 생성합니다. 이 명령은 함수 프로젝트 이름과 이미지 레지스트리 이름을 사용하여 함수의 정규화된 이미지 이름을 구성합니다.

5.1.3.1. 이미지 컨테이너 유형

기본적으로 **kn func build** 는 Red Hat S2I(Source-to-Image) 기술을 사용하여 컨테이너 이미지를 생성합니다.

Red Hat S2I(Source-to-Image)를 사용하는 빌드 명령 예

```
$ kn func build
```

5.1.3.2. 이미지 레지스트리 유형

OpenShift Container Registry는 기본적으로 함수 이미지를 저장하기 위해 이미지 레지스트리로 사용됩니다.

OpenShift Container Registry를 사용하는 빌드 명령 예

```
$ kn func build
```

출력 예

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

--registry 플래그를 사용하여 OpenShift Container Registry를 기본 이미지 레지스트리로 사용하여 재정의할 수 있습니다.

quay.io를 사용하도록 OpenShift Container Registry를 재정의하는 빌드 명령의 예

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

출력 예

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

5.1.3.3. push 플래그

kn func build 명령에 **--push** 플래그를 추가하여 성공적으로 빌드된 후 함수 이미지를 자동으로 푸시할 수 있습니다.

OpenShift Container Registry를 사용하는 빌드 명령 예

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

5.1.3.4. 도움말 명령

help 명령을 사용하여 **kn func build** 명령 옵션에 대해 자세히 알아볼 수 있습니다.

빌드 도움말 명령

```
$ kn func help build
```

5.1.4. 함수 배포

kn func deploy 명령을 사용하여 Knative 서비스로 클러스터에 함수를 배포할 수 있습니다. 대상 함수가 이미 배포된 경우 컨테이너 이미지 레지스트리로 푸시된 새 컨테이너 이미지로 업데이트되고 Knative 서비스가 업데이트됩니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.

- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 배포하려는 함수를 이미 생성하고 초기화해야 합니다.

프로세스

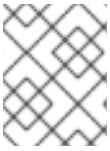
- 함수를 배포합니다.

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

출력 예

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

  - **namespace**를 지정하지 않으면 함수가 현재 네임스페이스에 배포됩니다.
  - 이 함수는 **path**를 지정하지 않는 한 현재 디렉터리에서 배포됩니다.
  - Knative 서비스 이름은 프로젝트 이름에서 파생되며 이 명령을 사용하여 변경할 수 없습니다.



참고

개발자 화면의 **+추가** 보기에서 Git **에서 가져오기 또는Serverless Function 생성 을 사용하여 Git** 리포지토리 URL로 서버리스 기능을 생성할 수 있습니다.

5.1.5. 기존 함수 나열

kn func list를 사용하여 기존 함수를 나열할 수 있습니다. Knative 서비스로 배포된 함수를 나열하려면 **kn service list**를 사용할 수도 있습니다.

프로세스

- 기존 함수를 나열합니다.

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

출력 예

```
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
```

- Knative 서비스로 배포된 함수를 나열합니다.

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

출력 예

```
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. 함수 설명

kn func info 명령은 함수 이름, 이미지, 네임스페이스, Knative 서비스 정보, 경로 정보 및 이벤트 서브스크립션과 같은 배포된 기능에 대한 정보를 출력합니다.

프로세스

- 함수를 설명합니다.

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

명령 예

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

출력 예

```
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. 테스트 이벤트와 함께 배포된 함수 호출

kn func invoke CLI 명령을 사용하여 로컬 또는 OpenShift Container Platform 클러스터에서 함수를 호출하는 테스트 요청을 보낼 수 있습니다. 이 명령을 사용하여 함수가 작동하고 이벤트를 올바르게 수신할 수 있는지 테스트할 수 있습니다. 함수를 로컬로 호출하면 함수 개발 중에 빠른 테스트에 유용합니다. 클러스터에서 함수를 호출하면 프로덕션 환경에 더 가까운 테스트를 수행하는 데 유용합니다.

사전 요구 사항

- OpenShift Serverless Operator 및 Knative Serving이 클러스터에 설치되어 있습니다.
- Knative(**kn**) CLI가 설치되어 있습니다.
- 프로젝트를 생성했거나 OpenShift Container Platform에서 애플리케이션 및 기타 워크로드를 생성하는 데 적절한 역할 및 권한이 있는 프로젝트에 액세스할 수 있습니다.
- 호출하려는 함수를 이미 배포해야 합니다.

프로세스

- 함수를 호출합니다.

```
$ kn func invoke
```

- **kn func invoke** 명령은 현재 실행 중인 로컬 컨테이너 이미지가 있거나 클러스터에 배포된 함수가 있는 경우에만 작동합니다.
- **kn func invoke** 명령은 기본적으로 로컬 디렉터리에서 실행되며 이 디렉터리는 함수 프로젝트라고 가정합니다.

5.1.7.1. kn func 호출 선택적 매개변수

다음 **kn func invoke** CLI 명령 플래그를 사용하여 요청에 대한 선택적 매개변수를 지정할 수 있습니다.

플래그	설명
-t, --target	호출된 함수의 대상 인스턴스를 지정합니다(예: **로컬** 또는 **원격** 또는 **https://staging.example.com/**). 기본 대상은 **local** 입니다.
-f, --format	메시지의 형식을 지정합니다(예: **cloudevent** 또는 **http**).
--id	요청에 대한 고유한 문자열 식별자를 지정합니다.
-n, --namespace	클러스터의 네임스페이스를 지정합니다.
--source	요청에 대한 보낸 사람 이름을 지정합니다. 이는 CloudEvent **소스** 속성에 해당합니다.
--type	요청 유형을 지정합니다(예: **boson.fn**). 이는 CloudEvent **유형** 속성에 해당합니다.
--data	요청에 대한 콘텐츠를 지정합니다. CloudEvent 요청의 경우 CloudEvent **데이터** 속성입니다.
--file	전송할 데이터를 포함하는 로컬 파일의 경로를 지정합니다.
--content-type	요청에 대한 MIME 콘텐츠 형식을 지정합니다.
-p, --path	프로젝트 디렉터리의 경로를 지정합니다.
-c, --confirm	프롬프트를 활성화하여 모든 옵션을 대화형으로 확인할 수 있습니다.
-v, --verbose	인쇄 상세 출력을 활성화합니다.
-h, --help	**kn func invoke** 를 사용하는 방법에 대한 정보를 출력합니다.

5.1.7.1.1. 기본 매개변수

다음 매개변수는 **kn func invoke** 명령의 기본 속성을 정의합니다.

이벤트 대상(-t,--target)

호출된 함수의 대상 인스턴스입니다. **로컬** 로 배포된 함수의 로컬 값, 원격으로 배포된 함수의 **원격** 값 또는 임의의 엔드포인트에 배포된 함수의 URL을 허용합니다. 대상을 지정하지 않으면 기본값은 **local** 입니다.

이벤트 메시지 형식(-f,--format)

이벤트의 메시지 형식(예: **http** 또는 **cloudevent**) 기본값은 함수를 생성할 때 사용된 템플릿의 형식입니다.

이벤트 유형(--type)

전송되는 이벤트 유형입니다. 각 이벤트 생산자의 설명서에 설정된 **type** 매개변수에 대한 정보를 찾을 수 있습니다. 예를 들어 API 서버 소스는 생성된 이벤트의 **type** 매개 변수를 **dev.knative.apiserver.resource.update** 로 설정할 수 있습니다.

이벤트 소스(--source)

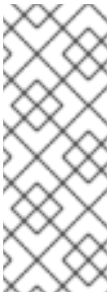
이벤트를 생성한 고유한 이벤트 소스입니다. 이벤트 소스의 URI(예: https://10.96.0.1/) 또는 이벤트 소스의 이름일 수 있습니다.https://10.96.0.1/

이벤트 ID(--id)

이벤트 프로듀서에 의해 생성되는 임의의 고유 ID입니다.

이벤트 데이터(--data)

kn func invoke 명령에서 전송한 이벤트에 대한 데이터 값을 지정할 수 있습니다. 예를 들어 이벤트에 이 데이터 문자열이 포함되도록 "Hello World" 와 같은 --data 값을 지정할 수 있습니다. 기본적으로 kn func에서 생성한 이벤트에는 데이터가 포함되지 않습니다.



참고

클러스터에 배포된 함수는 source 및 type과 같은 속성 값을 제공하는 기존 이벤트 소스의 이벤트에 응답할 수 있습니다. 이러한 이벤트에는 종종 이벤트의 도메인별 컨텍스트를 캡처하는 JSON 형식의 data 값이 있습니다. 개발자는 이 문서에 명시된 CLI 플래그를 사용하여 로컬 테스트를 위해 해당 이벤트를 시뮬레이션할 수 있습니다.

이벤트 데이터를 포함하는 로컬 파일을 제공하기 위해 --file 플래그를 사용하여 이벤트 데이터를 보낼 수도 있습니다. 이 경우 --content-type 을 사용하여 콘텐츠 유형을 지정합니다.

데이터 콘텐츠 유형(--content-type)

--data 플래그를 사용하여 이벤트에 대한 데이터를 추가하는 경우 --content-type 플래그를 사용하여 이벤트에서 전달하는 데이터 유형을 지정할 수 있습니다. 이전 예에서 데이터는 일반 텍스트이므로 kn func invoke --data "Hello world!" --content-type "text/plain" 을 지정할 수 있습니다.

5.1.7.1.2. 명령 예

이는 kn func invoke 명령의 일반적인 호출입니다.

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

예를 들어 "Hello world!" 이벤트를 보내려면 다음을 실행할 수 있습니다.

```
$ 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. 데이터로 파일 지정

이벤트 데이터가 포함된 디스크에서 파일을 지정하려면 **--file** 및 **--content-type** 플래그를 사용합니다.

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

예를 들어 **test.json** 파일에 저장된 **JSON** 데이터를 보내려면 다음 명령을 사용합니다.

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

5.1.7.1.2.2. 함수 프로젝트 지정

--path 플래그를 사용하여 함수 프로젝트의 경로를 지정할 수 있습니다.

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

예를 들어 **./example/example-function** 디렉터리에 있는 함수 프로젝트를 사용하려면 다음 명령을 사용합니다.

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

5.1.7.1.2.3. 대상 함수가 배포되는 위치 지정

기본적으로 **kn func call**은 함수의 로컬 배포를 대상으로 합니다.

```
$ kn func invoke
```

다른 배포를 사용하려면 **--target** 플래그를 사용합니다.

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

예를 들어 클러스터에 배포된 함수를 사용하려면 **--target** 원격 플래그를 사용합니다.

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

임의의 **URL**에 배포된 함수를 사용하려면 **--target <URL>** 플래그를 사용합니다.

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

로컬 배포를 명시적으로 대상으로 지정할 수 있습니다. 이 경우 함수가 로컬에서 실행되지 않으면 명령이 실패합니다.

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

5.1.8. 함수 삭제

kn func delete 명령을 사용하여 함수를 삭제할 수 있습니다. 이 기능은 더 이상 함수가 필요하지 않을 때 유용하며 클러스터에 리소스를 저장하는 데 도움이 될 수 있습니다.

프로세스

- 함수를 삭제합니다.

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

  - 삭제할 함수의 이름 또는 경로가 지정되지 않은 경우 현재 디렉터리에서 **func.yaml** 파일을 검색하고 삭제할 함수를 결정합니다.
  - 네임스페이스를 지정하지 않으면 기본값은 **func.yaml** 파일의 **namespace** 값으로 설정됩니다.