Red Hat OpenShift Serverless 1.28

Knative CLI

Knative Functions、Serving、および Eventing の CLI コマンドの概要

Last Updated: 2023-09-19

Red Hat OpenShift Serverless 1.28 Knative CLI

Knative Functions、Serving、および Eventing の CLI コマンドの概要

法律上の通知

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

このドキュメントでは、Knative Functions、Serving、Eventing で使用できる CLI コマンドの概要を説明します。また、Knative CLI の設定とプラグインの使用も説明します。

目次

第1章 KNATIVE SERVING CLI コマンド 3
1.1. KN SERVICE コマンド 3
1.2. オフラインモードの KN サービスコマンド 6
1.3. KN CONTAINER コマンド 9
1.4. KN DOMAIN コマンド 10

第2章 KNATIVE CLI の設定 13

第3章 KNATIVE CLI プラグイン 14
3.1. KN-EVENT プラグインを使用してイベントを作成する 14
3.2. KN-EVENT プラグインを使用したイベントの送信 15

第4章 KNATIVE EVENTING CLI コマンド 17
4.1. KN SOURCE コマンド 17

第5章 KNATIVE FUNCTIONS CLI コマンド 27
5.1. KN 関数コマンド 27

第1章 KNATIVE SERVING CLI コマンド

1.1. KN SERVICE コマンド

以下のコマンドを使用して Knative サービスを作成し、管理できます。

1.1.1. Knative CLI を使用したサーバーレスアプリケーションの作成

Knative (**kn**) CLI を使用してサーバーレスアプリケーションを作成すると、YAML ファイルを直接修正するよりも合理的で直感的なユーザーインターフェイスが得られます。**kn service create** コマンドを使用して、基本的なサーバーレスアプリケーションを作成できます。

前提条件

- OpenShift Serverless Operator および Knative Serving がクラスターにインストールされていること。
- Knative (**kn**) CLI をインストールしている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

- Knative サービスを作成します。

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

詳細は以下のようになります。

  - **--image** は、アプリケーションのイメージの URI です。
  - **--tag** は、サービスで作成される初期リビジョンにタグを追加するために使用できるオプションのフラグです。

コマンドの例

```
$ kn service create event-display \
    --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

出力例

```
Creating service 'event-display' in namespace 'default':

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

Service 'event-display' created with latest revision 'event-display-bxshg-1' and URL:
http://event-display-default.apps-crc.testing
```

1.1.2. Knative CLI を使用したサーバーレスアプリケーションの更新

サービスを段階的に構築する際に、コマンドラインで **kn service update** コマンドを使用し、対話式のセッションを使用できます。**kn service apply** コマンドとは対照的に、**kn service update** コマンドを使用する際は、Knative サービスの完全な設定ではなく、更新が必要な変更のみを指定する必要があります。

コマンドの例

- 新規の環境変数を追加してサービスを更新します。

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

- 新しいポートを追加してサービスを更新します。

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

- 新しい要求および制限パラメーターを追加してサービスを更新します。

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

- **latest** タグをリビジョンに割り当てます。

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

- サービスの最新の **READY** リビジョンについて、**testing** から **staging** にタグを更新します。

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

- **test** タグをトラフィックの 10% を受信するリビジョンに追加し、残りのトラフィックをサービスの最新の **READY** リビジョンに送信します。

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

1.1.3. サービス宣言の適用

kn service apply コマンドを使用して Knative サービスを宣言的に設定できます。サービスが存在しない場合は、これが作成されますが、それ以外の場合は、既存のサービスが変更されたオプションで更新されます。

kn service apply コマンドは、ユーザーがターゲットの状態を宣言するために単一コマンドでサービスの状態を詳細に指定したい場合など、とくにシェルスクリプトや継続的インテグレーションパイプラインで役に立ちます。

kn service apply を使用する場合は、Knative サービスの詳細な設定を指定する必要があります。これは **kn service update** コマンドとは異なります。このコマンドでは、更新する必要のあるオプションを指定するだけで済みます。

コマンドの例

- サービスを作成します。

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

- 環境変数をサービスに追加します。

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

- JSON または YAML ファイルからサービス宣言を読み取ります。

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

1.1.4. Knative CLI を使用したサーバーレスアプリケーションの記述

kn service describe コマンドを使用して Knative サービスを記述できます。

コマンドの例

- サービスを記述します。

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

--verbose フラグは任意ですが、さらに詳細な説明を提供するために追加できます。通常の出力と詳細の出力の違いについては、以下の例に示されます。

--verbose フラグを使用しない出力例

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

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

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

--verbose フラグを使用する出力例

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

Revisions:
  100%  @latest (hello-00001) [1] (3m)
        Image:  docker.io/openshift/hello-openshift (pinned to aaea76)
```

```
    Env:    RESPONSE=Hello Serverless!

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

- サービスを YAML 形式で記述します。

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

- サービスを JSON 形式で記述します。

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

- サービス URL のみを出力します。

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

1.2. オフラインモードの KN サービスコマンド

1.2.1. Knative CLI オフラインモードについて

kn service コマンドを実行すると、変更が即座にクラスターに伝播されます。ただし、別の方法として、オフラインモードで **kn service** コマンドを実行できます。オフラインモードでサービスを作成すると、クラスター上で変更は発生せず、代わりにサービス記述子ファイルがローカルマシンに作成されます。

重要

Knative CLI のオフラインモードはテクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat の実稼働環境におけるサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

記述子ファイルの作成後、手動で変更し、バージョン管理システムで追跡できます。記述子ファイルで **kn service create -f**、**kn service apply -f** または **oc apply -f** コマンドを使用して変更をクラスターに伝播することもできます。

オフラインモードには、いくつかの用途があります。

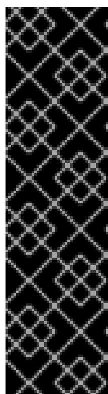
- 記述子ファイルを使用してクラスターで変更する前に、記述子ファイルを手動で変更できます。
- バージョン管理システムでは、サービスの記述子ファイルをローカルで追跡できます。これにより、記述子ファイルを再利用できます。たとえば、継続的インテグレーション (CI) パイプライン、開発環境またはデモなどで、ターゲットクラスター以外の配置が可能になります。

- 作成した記述子ファイルを検証して Knative サービスについて確認できます。特に、生成されるサービスが **kn** コマンドに渡されるさまざまな引数によってどのように影響するかを確認できます。

オフラインモードには、高速で、クラスターへの接続を必要としないという利点があります。ただし、オフラインモードではサーバー側の検証がありません。したがって、サービス名が一意であることや、指定のイメージをプルできることなどを確認できません。

1.2.2. オフラインモードを使用したサービスの作成

オフラインモードで **kn service** コマンドを実行すると、クラスター上で変更は発生せず、代わりにサービス記述子ファイルがローカルマシンに作成されます。記述子ファイルを作成した後、クラスターに変更を伝播する前にファイルを変更することができます。



重要

Knative CLI のオフラインモードはテクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat の実稼働環境におけるサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

前提条件

- OpenShift Serverless Operator および Knative Serving がクラスターにインストールされていること。
- Knative (**kn**) CLI をインストールしている。

手順

1. オフラインモードでは、ローカルの Knative サービス記述子ファイルを作成します。

```
$ kn service create event-display \
    --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest \
    --target ./ \
    --namespace test
```

出力例

```
Service 'event-display' created in namespace 'test'.
```

- **--target ./** フラグはオフラインモードを有効にし、**./** を新しいディレクトリーツリーを保存するディレクトリーとして指定します。
既存のディレクトリーを指定せずに、**--target my-service.yaml** などのファイル名を使用すると、ディレクトリーツリーは作成されません。代わりに、サービス記述子ファイル **my-service.yaml** のみが現在のディレクトリーに作成されます。

  ファイル名には、**.yaml**、**.yml** または **.json** 拡張子を使用できます。**.json** を選択すると、JSON 形式でサービス記述子ファイルが作成されます。

- **--namespace test** オプションは、新規サービスを **テスト** namespace に配置します。
--namespace を使用せずに、OpenShift Container Platform クラスターにログインしている場合には、記述子ファイルが現在の namespace に作成されます。それ以外の場合は、記述子ファイルが **default** の namespace に作成されます。

2. 作成したディレクトリー構造を確認します。

```
$ tree ./
```

出力例

```
./
└── test
    └── ksvc
        └── event-display.yaml

2 directories, 1 file
```

- **--target** で指定する現在の **./** ディレクトリーには新しい **test/** ディレクトリーが含まれます。このディレクトリーの名前は、指定の namespace をもとに付けられます。
- **test/** ディレクトリーには、リソースタイプの名前が付けられた **ksvc** ディレクトリーが含まれます。
- **ksvc** ディレクトリーには、指定のサービス名に従って命名される記述子ファイル **event-display.yaml** が含まれます。

3. 生成されたサービス記述子ファイルを確認します。

```
$ cat test/ksvc/event-display.yaml
```

出力例

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

4. 新しいサービスに関する情報を一覧表示します。

```
$ kn service describe event-display --target ./ --namespace test
```

出力例

```
Name:       event-display
Namespace:  test
Age:
URL:

Revisions:

Conditions:
  OK TYPE    AGE REASON
```

- **--target ./** オプションは、namespace サブディレクトリーを含むディレクトリー構造のルートディレクトリーを指定します。
  または、**--target** オプションで YAML または JSON ファイルを直接指定できます。使用可能なファイルの拡張子は、**.yaml**、**.yml**、および **.json** です。
- **--namespace** オプションは、namespace を指定し、この namespace は必要なサービス記述子ファイルを含むサブディレクトリーの **kn** と通信します。
  --namespace を使用せず、OpenShift Container Platform クラスターにログインしている場合には、**kn** は現在の namespace をもとに名前が付けられたサブディレクトリーでサービスを検索します。それ以外の場合は、**kn** は **default**/ サブディレクトリーで検索します。

5. サービス記述子ファイルを使用してクラスターでサービスを作成します。

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

出力例

```
Creating service 'event-display' in namespace 'test':

  0.058s The Route is still working to reflect the latest desired specification.
  0.098s ...
  0.168s Configuration "event-display" 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 'event-display' created to latest revision 'event-display-00001' is available at URL:
http://event-display-test.apps.example.com
```

1.3. KN CONTAINER コマンド

以下のコマンドを使用して、Knative サービス仕様で複数のコンテナーを作成し、管理できます。

1.3.1. Knative クライアントマルチコンテナーのサポート

kn container add コマンドを使用して、YAML コンテナーの仕様を標準出力に出力できます。このコマンドは、定義を作成するために他の標準の **kn** フラグと共に使用できるため、マルチコンテナーのユースケースに役立ちます。

kn container add コマンドは、**kn service create** コマンドで使用できるコンテナー関連のすべてのフラグを受け入れます。UNIX パイプ (|) を使用して **kn container add** コマンドを連結して、一度に複数のコンテナー定義を作成することもできます。

コマンドの例

- イメージからコンテナーを追加し、標準出力に出力します。

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

コマンドの例

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

出力例

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

- 2 つの **kn container add** コマンドを連結してから、**kn service create** コマンドに渡して、2 つのコンテナーで Knative サービスを作成します。

```
$ 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 - は、**kn** が YAML ファイルの代わりにパイプ入力を読み取る特別なケースを指定します。

コマンドの例

```
$ 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 フラグは YAML ファイルへのパスを受け入れることもできます。

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

コマンドの例

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

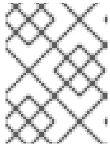
1.4. KN DOMAIN コマンド

以下のコマンドを使用して、ドメインマッピングを作成および管理できます。

1.4.1. Knative CLI を使用したカスタムドメインマッピングの作成

前提条件

- OpenShift Serverless Operator および Knative Serving がクラスターにインストールされている。
- Knative サービスまたはルートを作成し、その CR にマップするカスタムドメインを制御している。



注記

カスタムドメインは OpenShift Container Platform クラスターの DNS を参照する必要があります。

- Knative (**kn**) CLI をインストールしている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

- ドメインを現在の namespace の CR にマップします。

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

コマンドの例

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

--ref フラグは、ドメインマッピング用のアドレス指定可能なターゲット CR を指定します。

--ref フラグの使用時に接頭辞が指定されていない場合、ターゲットが現在の namespace の Knative サービスであることを前提としています。

- ドメインを指定された namespace の Knative サービスにマップします。

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

コマンドの例

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

- ドメインを Knative ルートにマップします。

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

コマンドの例

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

1.4.2. Knative CLI を使用したカスタムドメインマッピングの管理

DomainMapping カスタムリソース (CR) の作成後に、既存の CR の一覧表示、既存の CR の情報の表示、CR の更新、または Knative (**kn**) CLI を使用した CR の削除を実行できます。

前提条件

- OpenShift Serverless Operator および Knative Serving がクラスターにインストールされている。
- 1 つ以上の **DomainMapping** CR を作成している。
- Knative (**kn**) CLI ツールをインストールしている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。

手順

- 既存の **DomainMapping** CR を一覧表示します。

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

- 既存の **DomainMapping** CR の詳細を表示します。

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

- 新規ターゲットを参照するように **DomainMapping** CR を更新します。

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

- **DomainMapping** CR を削除します。

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

第2章 KNATIVE CLI の設定

config.yaml 設定ファイルを作成することで、Knative (**kn**) CLI セットアップをカスタマイズできます。**--config** フラグを使用してこの設定を指定できます。指定しない場合、設定がデフォルトの場所から選択されます。デフォルトの設定場所は XDGBaseDirectory 仕様 に準拠しており、UNIX システムと Windows システムでは異なります。

UNIX システムの場合:

- **XDG_CONFIG_HOME** 環境変数が設定されている場合、Knative (**kn**) CLI が検索するデフォルト設定の場所は **$XDG_CONFIG_HOME/kn** になります。
- **XDG_CONFIG_HOME** 環境変数が設定されていない場合、Knative (**kn**) CLI は **$HOME/.config/kn/config.yaml** のユーザーのホームディレクトリーにある設定を検索します。

Windows システムの場合、デフォルトの Knative (**kn**) CLI 設定の場所は **%APPDATA%\kn** です。

設定ファイルのサンプル

```
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**) CLI が **PATH** 環境変数でプラグインを検索するかどうかを指定します。これはブール型の設定オプションです。デフォルト値は **false** です。

(2) Knative (**kn**) CLI がプラグインを検索するディレクトリーを指定します。前述のように、デフォルトのパスはオペレーティングシステムによって異なります。これには、ユーザーに表示される任意のディレクトリーを指定できます。

(3) **sink-mappings** 仕様は、Knative (**kn**) CLI コマンドで **--sink** フラグを使用する場合に使用される Kubernetes のアドレス可能なリソースを定義します。

(4) シンクの記述に使用する接頭辞。サービスの **svc**、 **channel**、および **broker** は Knative (**kn**) CLI で事前に定義される接頭辞です。

(5) Kubernetes リソースの API グループ。

(6) Kubernetes リソースのバージョン。

(7) Kubernetes リソースタイプの複数形の名前。例: **services** または **brokers**

第3章 KNATIVE CLI プラグイン

Knative (**kn**) CLI は、プラグインの使用をサポートします。これにより、カスタムコマンドとコアディストリビューションの一部ではない他の共有コマンドを追加でき、**kn** インストールの機能の拡張を可能にします。Knative (**kn**) CLI プラグインは主な **kn** 機能として同じ方法で使用されます。

現在、Red Hat は **kn-source-kafka** プラグインと **kn-event** プラグインをサポートしています。

重要

kn-event プラグインは、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat の実稼働環境におけるサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

3.1. KN-EVENT プラグインを使用してイベントを作成する

kn event build コマンドのビルダーのようなインターフェイスを使用して、イベントをビルドできます。その後、そのイベントを後で送信するか、別のコンテキストで使用できます。

前提条件

- Knative (**kn**) CLI をインストールしている。

手順

- イベントをビルドします。

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

  ここでは、以下のようになります。

  - **--field** フラグは、データをフィールド/値のペアとしてイベントに追加します。これは複数回使用できます。
  - **--type** フラグを使用すると、イベントのタイプを指定する文字列を指定できます。
  - **--id** フラグは、イベントの ID を指定します。
  - **json** または **yaml** 引数を **--output** フラグと共に使用して、イベントの出力形式を変更できます。
    これらのフラグはすべてオプションです。

    簡単なイベントのビルド

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

    YAML 形式のビルドされたイベント

```
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
```

サンプルトランザクションイベントのビルド

```
$ 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 形式のビルドされたイベント


```
{
  "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 プラグインを使用したイベントの送信

kn event send コマンドを使用して、イベントを送信できます。イベントは、公開されているアドレス、または Kubernetes サービスや Knative サービス、ブローカー、チャネル等のクラスター内のアドレス指定可能なリソースのいずれかに送信できます。このコマンドは、**kn event build** コマンドと同じビルダーのようなインターフェイスを使用します。

前提条件

- Knative (**kn**) CLI をインストールしている。

手順

手順

- イベントの送信:

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

ここでは、以下のようになります。

  - **--field** フラグは、データをフィールド/値のペアとしてイベントに追加します。これは複数回使用できます。
  - **--type** フラグを使用すると、イベントのタイプを指定する文字列を指定できます。
  - **--id** フラグは、イベントの ID を指定します。
  - イベントを一般にアクセス可能な宛先に送信する場合は、**--to-url** フラグを使用して URL を指定します。
  - イベントをクラスター内の Kubernetes リソースに送信する場合は、**--to** フラグを使用して宛先を指定します。
    - **<Kind>:<ApiVersion>:<name>** 形式を使用して Kubernetes リソースを指定します。
  - **--namespace** フラグは namespace を指定します。省略すると、namespace は現在のコンテキストから取得されます。
    --to-url または **--to** のいずれかを使用する必要がある宛先の仕様を除き、これらのフラグはすべてオプションです。

    以下の例は、イベントを URL に送信するケースを示しています。

    コマンドの例

```
$ 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/
```

以下の例は、イベントをクラスター内のリソースに送信するケースを示しています。

コマンドの例

```
$ 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 コマンド

4.1. KN SOURCE コマンド

以下のコマンドを使用して、Knative イベントソースを一覧表示、作成、および管理できます。

4.1.1. Knative CLI の使用による利用可能なイベントソースタイプの一覧表示

kn source list-types CLI コマンドを使用して、クラスターで作成して使用できるイベントソースタイプを一覧表示できます。

前提条件

- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。
- Knative (**kn**) CLI をインストールしている。

手順

1. ターミナルに利用可能なイベントソースタイプを一覧表示します。

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

出力例

```
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. オプション: OpenShift Container Platform では、利用可能なイベントソースタイプを YAML 形式でリストすることもできます。

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

4.1.2. Knative CLI シンクフラグ

Knative (**kn**) CLI を使用してイベントソースを作成する場合、**--sink** フラグを使用して、イベントがリソースから送信されるシンクを指定できます。シンクは、他のリソースから受信イベントを受信できる、アドレス指定可能または呼び出し可能な任意のリソースです。

以下の例では、サービスの **http://event-display.svc.cluster.local** をシンクとして使用するシンクバインディングを作成します。

シンクフラグを使用したコマンドの例

```
$ 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** の **svc** は、シンクが Knative サービスであることを判別します。他のデフォルトのシンクの接頭辞には、**channel** および **broker** が含まれます。

4.1.3. Knative CLI を使用したコンテナーソースの作成および管理

kn source container コマンドを使用し、Knative (**kn**) CLI を使用してコンテナーソースを作成および管理できます。イベントソースを作成するために Knative CLI を使用すると、YAML ファイルを直接修正するよりも合理的で直感的なユーザーインターフェイスが得られます。

コンテナーソースを作成します。

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

コンテナーソースの削除

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

コンテナーソースを記述します。

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

既存のコンテナーソースを一覧表示

```
$ kn source container list
```

既存のコンテナーソースを YAML 形式で一覧表示

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

コンテナーソースを更新します。

このコマンドにより、既存のコンテナーソースのイメージ URI が更新されます。

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

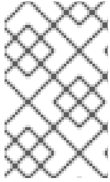
4.1.4. Knative CLI を使用した API サーバーソースの作成

kn source apiserver create コマンドを使用し、**kn** CLI を使用して API サーバーソースを作成できます。API サーバーソースを作成するために **kn** CLI を使用すると、YAML ファイルを直接修正するよりも合理的で直感的なユーザーインターフェイスが得られます。

前提条件

- OpenShift Serverless Operator および Knative Eventing がクラスターにインストールされている。

- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- Knative (**kn**) CLI をインストールしている。



手順

既存のサービスアカウントを再利用する必要がある場合には、既存の **ServiceAccount** リソースを変更して、新規リソースを作成せずに、必要なパーミッションを含めることができます。

1. イベントソースのサービスアカウント、ロールおよびロールバインディングを YAML ファイルとして作成します。

```
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 この namespace を、イベントソースのインストールに選択した namespace に変更します。

2. YAML ファイルを適用します。

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

3. イベントシンクを持つ API サーバーソースを作成します。次の例では、シンクはブローカーです。

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

4. API サーバーソースが正しく設定されていることを確認するには、受信メッセージをログにダンプする Knative サービスを作成します。

```
$ kn service create <service_name> --image quay.io/openshift-knative/knative-eventing-
sources-event-display:latest
```

5. ブローカーをイベントシンクとして使用した場合は、トリガーを作成して、**default** のブローカーからサービスへのイベントをフィルタリングします。

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

6. デフォルト namespace で Pod を起動してイベントを作成します。

```
$ oc create deployment hello-node --image quay.io/openshift-knative/knative-eventing-
sources-event-display:latest
```

7. 以下のコマンドを入力し、生成される出力を検査して、コントローラーが正しくマップされていることを確認します。

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

出力例

```
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
```

検証

メッセージダンパー機能ログを確認して、Kubernetes イベントが Knative に送信されていることを確認できます。

1. Pod を取得します。

```
$ oc get pods
```

2. Pod のメッセージダンパー機能ログを表示します。

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

出力例

```
☁  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{hello-node}",
      "kind": "Pod",
      "name": "hello-node",
      "namespace": "default",
       .....
    },
    "kind": "Event",
    "message": "Started container",
    "metadata": {
      "name": "hello-node.159d7608e3a3572c",
      "namespace": "default",
      ....
    },
    "reason": "Started",
    ...
  }
```

API サーバーソースの削除

1. トリガーを削除します。

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

2. イベントソースを削除します。

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

3. サービスアカウント、クラスターロール、およびクラスターバインディングを削除します。

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

4.1.5. Knative CLI を使用した ping ソースの作成

kn source ping create コマンドを使用し、Knative (**kn**) CLI を使用して ping ソースを作成できます。イベントソースを作成するために Knative CLI を使用すると、YAML ファイルを直接修正するよりも合理的で直感的なユーザーインターフェイスが得られます。

前提条件

- OpenShift Serverless Operator、Knative Serving、および Knative Eventing がクラスターにインストールされている。
- Knative (**kn**) CLI をインストールしている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- オプション: この手順の検証手順を使用する場合は、OpenShift CLI (**oc**) をインストールします。

手順

1. ping ソースが機能していることを確認するには、受信メッセージをサービスのログにダンプする単純な Knative サービスを作成します。

```
$ kn service create event-display \
    --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

2. 要求する必要のある ping イベントのセットごとに、PingSource をイベントコンシューマーと同じ namespace に作成します。

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

3. 以下のコマンドを入力し、出力を検査して、コントローラーが正しくマップされていることを確認します。

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

出力例

```
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
```

検証

シンク Pod のログを確認して、Kubernetes イベントが Knative イベントに送信されていることを確認できます。

デフォルトで、Knative サービスは、トラフィックが 60 秒以内に受信されない場合に Pod を終了します。本書の例では、新たに作成される Pod で各メッセージが確認されるように 2 分ごとにメッセージを送信する ping ソースを作成します。

1. 作成された新規 Pod を監視します。

```
$ watch oc get pods
```

2. Ctrl+C を使用して Pod の監視をキャンセルし、作成された Pod のログを確認します。

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

出力例

```
☁  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!"
  }
```

ping ソースの削除

- ping ソースを削除します。

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

4.1.6. Knative CLI を使用した Apache Kafka イベントソースの作成

kn source kafka create コマンドを使用し、Knative (**kn**) CLI を使用して Kafka ソースを作成できます。イベントソースを作成するために Knative CLI を使用すると、YAML ファイルを直接修正するよりも合理的で直感的なユーザーインターフェイスが得られます。

前提条件

- OpenShift Serverless Operator、Knative Eventing、Knative Serving、および **KnativeKafka** カスタムリソース (CR) がクラスターにインストールされている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- インポートする Kafka メッセージを生成する Red Hat AMQ Streams (Kafka) クラスターにアクセスできる。
- Knative (**kn**) CLI をインストールしている。
- オプション: この手順で検証ステップを使用する場合は、OpenShift CLI (**oc**) をインストールします。

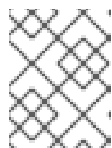
手順

1. Kafka イベントソースが機能していることを確認するには、受信メッセージをサービスのログにダンプする Knative サービスを作成します。

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

2. **KafkaSource** CR を作成します。

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



注記

このコマンドのプレースホルダー値は、ソース名、ブートストラップサーバー、およびトピックの値に置き換えます。

--servers、**--topics**、および **--consumergroup** オプションは、Kafka クラスターへの接続パラメーターを指定します。**--consumergroup** オプションは任意です。

3. オプション: 作成した **KafkaSource** CR の詳細を表示します。

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

出力例

```
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
```

検証手順

1. Kafka インスタンスをトリガーし、メッセージをトピックに送信します。

```
$ 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
```

プロンプトにメッセージを入力します。このコマンドは、以下を前提とします。

- Kafka クラスターが **kafka** namespace にインストールされている。
- **KafkaSource** オブジェクトは、**my-topic** トピックを使用するように設定されている。

2. ログを表示して、メッセージが到達していることを確認します。

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

出力例

```
☁  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 コマンド

5.1. KN 関数コマンド

5.1.1. 関数の作成

関数をビルドし、デプロイする前に、Knative (**kn**) CLI を使用して関数を作成する必要があります。コマンドラインでパス、ランタイム、テンプレート、およびイメージレジストリーをフラグとして指定するか、**-c** フラグを使用してターミナルで対話型エクスペリエンスを開始できます。

前提条件

- OpenShift Serverless Operator および Knative Serving がクラスターにインストールされている。
- Knative (**kn**) CLI をインストールしている。

手順

- 関数プロジェクトを作成します。

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

  - 受け入れられるランタイム値には、**quarkus**、**node**、**typescript**、**go**、**python**、**springboot**、および **rust** が含まれます。
  - 受け入れられるテンプレート値には、**http** と **cloudevents** が含まれます。

    コマンドの例

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

    出力例

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

  - または、カスタムテンプレートを含むリポジトリーを指定することもできます。

    コマンドの例

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

    出力例

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

5.1.2. 機能をローカルで実行する

kn func run コマンドを使用して、現在のディレクトリーまたは **--path** フラグで指定されたディレクト

リーで機能をローカルに実行できます。実行している関数が以前にビルドされたことがない場合、またはプロジェクトファイルが最後にビルドされてから変更されている場合、**kn func run** コマンドは、既定で関数を実行する前に関数をビルドします。

現在のディレクトリーで機能を実行するコマンドの例

```
$ kn func run
```

パスとして指定されたディレクトリーで機能を実行するコマンドの例

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

--build フラグを使用して、プロジェクトファイルに変更がなくても、機能を実行する前に既存のイメージを強制的に再構築することもできます。

ビルドフラグを使用した実行コマンドの例

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

ビルド フラグを false に設定すると、イメージのビルドが無効になり、以前にビルドされたイメージを使用して機能が実行されます。

ビルドフラグを使用した実行コマンドの例

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

help コマンドを使用して、**kn func run** コマンドオプションの詳細を確認できます。

help コマンドの構築

```
$ kn func help run
```

5.1.3. 関数のビルド

関数を実行する前に、関数プロジェクトをビルドする必要があります。**kn func run** コマンドを使用している場合、関数は自動的に構築されます。ただし、**kn func build** コマンドを使用すると、実行せずに関数をビルドできます。これは、上級ユーザーやデバッグシナリオに役立ちます。

kn func build は、コンピューターまたは OpenShift Container Platform クラスターでローカルに実行できる OCI コンテナーイメージを作成します。このコマンドは、関数プロジェクト名とイメージレジストリー名を使用して、関数の完全修飾イメージ名を作成します。

5.1.3.1. イメージコンテナーの種類

デフォルトでは、**kn func build** は、Red Hat Source-to-Image (S2I) テクノロジーを使用してコンテナーイメージを作成します。

Red Hat Source-to-Image (S2I) を使用したビルドコマンドの例

```
$ kn func build
```

5.1.3.2. イメージレジストリーの種類

OpenShift Container Registry は、関数イメージを保存するためのイメージレジストリーとしてデフォルトで使用されます。

OpenShift Container Registry を使用したビルドコマンドの例

```
$ kn func build
```

出力例

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

--registry フラグを使用して、OpenShift Container Registry をデフォルトのイメージレジストリーとして使用することをオーバーライドできます。

quay.io を使用するように OpenShift Container Registry をオーバーライドするビルドコマンドの例

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

出力例

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

5.1.3.3. Push フラグ

--push フラグを **kn func build** コマンドに追加して、正常にビルドされた後に関数イメージを自動的にプッシュできます。

OpenShift Container Registry を使用したビルドコマンドの例

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

5.1.3.4. Help コマンド

kn func build コマンドオプションの詳細については、help コマンドを使用できます。

help コマンドの構築

```
$ kn func help build
```

5.1.4. 関数のデプロイ

kn func deploy コマンドを使用して、関数を Knative サービスとしてクラスターにデプロイできます。ターゲット関数がすでにデプロイされている場合には、コンテナーイメージレジストリーにプッシュされている新規コンテナーイメージで更新され、Knative サービスが更新されます。

前提条件

- OpenShift Serverless Operator および Knative Serving がクラスターにインストールされている。
- Knative (**kn**) CLI をインストールしている。
- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- デプロイする関数を作成し、初期化している必要がある。

手順

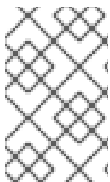
- 関数をデプロイします。

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

出力例

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

  - **namespace** が指定されていない場合には、関数は現在の namespace にデプロイされます。
  - この関数は、**パス** が指定されない限り、現在のディレクトリーからデプロイされます。
  - Knative サービス名はプロジェクト名から派生するので、以下のコマンドでは変更できません。



注記

Developer パースペクティブの **+Add** ビューで **Import from Git** または **Create Serverless Function** を使用して、Git リポジトリー URL を使用してサーバーレス関数を作成できます。

5.1.5. 既存の関数の一覧表示

kn func list を使用して既存の関数を一覧表示できます。Knative サービスとしてデプロイされた関数を一覧表示するには、**kn service list** を使用することもできます。

手順

- 既存の関数を一覧表示します。

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

出力例

```
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
```

- Knative サービスとしてデプロイされた関数を一覧表示します。

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

出力例

```
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. 関数の記述

kn func info コマンドは、関数名、イメージ、namespace、Knative サービス情報、ルート情報、イベントサブスクリプションなどのデプロイされた関数に関する情報を出力します。

手順

- 関数を説明します。

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

コマンドの例

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

出力例

```
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. テストイベントでのデプロイされた関数の呼び出し

kn func invoke CLI コマンドを使用して、ローカルまたは OpenShift Container Platform クラスター上で関数を呼び出すためのテストリクエストを送信できます。このコマンドを使用して、関数が機能し、イベントを正しく受信できることをテストできます。関数をローカルで呼び出すと、関数開発中の簡単なテストに役立ちます。クラスターで関数を呼び出すと、実稼働環境に近いテストに役立ちます。

前提条件

- OpenShift Serverless Operator および Knative Serving がクラスターにインストールされている。
- Knative (**kn**) CLI をインストールしている。

- OpenShift Container Platform でアプリケーションおよび他のワークロードを作成するために、プロジェクトを作成しているか、適切なロールおよびパーミッションを持つプロジェクトにアクセスできる。
- 呼び出す関数をすでにデプロイしている必要があります。

手順

- 関数を呼び出します。

```
$ kn func invoke
```

  - **kn func invoke** コマンドは、ローカルのコンテナーイメージが実行中の場合や、クラスターにデプロイされた関数がある場合にのみ機能します。
  - **kn func invoke** コマンドは、デフォルトでローカルディレクトリーで実行され、このディレクトリーが関数プロジェクトであると想定します。

5.1.7.1. kn func はオプションのパラメーターを呼び出します

次の **knfuncinvoke** CLI コマンドフラグを使用して、リクエストのオプションのパラメーターを指定できます。

フラグ	説明
-t、--target	呼び出された関数のターゲットインスタンスを指定します。たとえば、**local**、**remote**、**https://staging.example.com/** などです。デフォルトのターゲットは **local** です。
-f、-format	メッセージの形式を指定します (例: **cloudevent** または **http**)。
--id	リクエストの一意の文字列識別子を指定します。
-n、-namespace	クラスターの namespace を指定します。
--source	リクエストの送信者名を指定します。これは、CloudEvent **source** 属性に対応します。
--type	リクエストのタイプを指定します (例: **boson.fn**)。これは、CloudEvent **type** 属性に対応します。
--data	リクエストの内容を指定します。CloudEvent リクエストの場合、これは CloudEvent **data** 属性です。
--file	送信するデータを含むローカルファイルへのパスを指定します。
--content-type	リクエストの MIME コンテンツタイプを指定します。
-p、--path	プロジェクトディレクトリーへのパスを指定します。

フラグ	説明
-c、--confirm	すべてのオプションを対話的に確認するように要求を有効にします。
-v, --verbose	詳細出力の出力を有効にします。
-h、--help	**kn func invoke** の使用法に関する情報を出力します。

5.1.7.1.1. 主なパラメーター

次のパラメーターは、**kn func invoke** コマンドの主なプロパティーを定義します。

イベントターゲット (-t、-target)

呼び出された関数のターゲットインスタンス。ローカルにデプロイされた関数の **local** 値、リモートにデプロイされた関数の **remote** 値、または任意のエンドポイントにデプロイされた関数の URL を受け入れます。ターゲットが指定されていない場合、デフォルトで **local** になります。

イベントメッセージ形式 (-f、--format)

http や **cloudevent** などのイベントのメッセージ形式。これは、デフォルトで、関数の作成時に使用されたテンプレートの形式になります。

イベントタイプ (--type)

送信されるイベントのタイプ。各イベントプロデューサーのドキュメントで設定されている **type** パラメーターに関する情報を見つけることができます。たとえば、API サーバーソースは、生成されたイベントの **type** パラメーターを **dev.knative.apiserver.resource.update** として設定する場合があります。

イベントソース (--source)

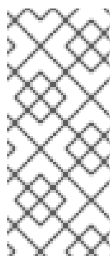
イベントを生成する一意のイベントソース。これは、https://10.96.0.1/ などのイベントソースの URI、またはイベントソースの名前である可能性があります。

イベント ID (--id)

イベントプロデューサーによって作成されるランダムな一意の ID。

イベントデータ (--data)

kn func invoke コマンドで送信されるイベントの **data** 値を指定できます。たとえば、イベントにこのデータ文字列が含まれるように、**"Hello World"** などの **--data** 値を指定できます。デフォルトでは、**kn func invoke** によって作成されたイベントにデータは含まれません。



注記

クラスターにデプロイされた関数は、**source** および **type** などのプロパティーの値を提供する既存のイベントソースからのイベントに応答できます。多くの場合、これらのイベントには、イベントのドメイン固有のコンテキストをキャプチャーする JSON 形式の **data** 値があります。本書に記載されている CLI フラグを使用して、開発者はローカルテスト用にこれらのイベントをシミュレートできます。

--file フラグを使用してイベントデータを送信し、イベントのデータを含むローカルファイルを指定することもできます。この場合は、**--content-type** を使用してコンテンツタイプを指定します。

データコンテンツタイプ (--content-type)

--data フラグを使用してイベントのデータを追加している場合は、**-content-type** フラグを使用して、イベントによって伝送されるデータのタイプを指定できます。前の例では、データはプレーン

テキストであるため、**kn func invoke --data "Hello world!" --content-type "text/plain"** を指定できます。

5.1.7.1.2. コマンドの例

これは、**kn func invoke** コマンドの一般的な呼び出しです。

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

たとえば、Hello world! イベントを送信すると、以下を行うことができます。

```
$ 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. データを使用したファイルの指定

イベントデータが含まれるディスクにファイルを指定するには、**--file** フラグおよび **--content-type** フラグを使用します。

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

たとえば、**test.json** ファイルに保存されている JSON データを送信するには、以下のコマンドを使用します。

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

5.1.7.1.2.2. 関数プロジェクトの指定

--path フラグを使用して、関数プロジェクトへのパスを指定できます。

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

たとえば、**./example/example-** function ディレクトリーにある function プロジェクトを使用するには、以下のコマンドを使用します。

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

5.1.7.1.2.3. ターゲット関数がデプロイされる場所の指定

デフォルトでは、**kn func invoke** は関数のローカルデプロイメントをターゲットにします。

```
$ kn func invoke
```

別のデプロイメントを使用するには、**--target** フラグを使用します。

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

たとえば、クラスターにデプロイされた関数を使用するには、**-target remote** フラグを使用します。

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

任意の URL にデプロイされた関数を使用するには、**-target <URL>** フラグを使用します。

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

ローカルデプロイメントを明示的にターゲットとして指定できます。この場合、関数がローカルで実行されていない場合、コマンドは失敗します。

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

5.1.8. 関数の削除

kn func delete コマンドを使用して関数を削除できます。これは、関数が不要になった場合に役立ち、クラスターのリソースを節約するのに役立ちます。

手順

- 関数を削除します。

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

  - 削除する関数の名前またはパスが指定されていない場合には、現在のディレクトリーで **func.yaml** ファイルを検索し、削除する関数を判断します。
  - namespace が指定されていない場合には、**func.yaml** の **namespace** の値にデフォルト設定されます。