Red Hat OpenShift Serverless 1.30

Serving

Knative Serving の使用開始および設定サービス

Last Updated: 2023-09-20

Red Hat OpenShift Serverless 1.30 Serving

Knative Serving の使用開始および設定サービス

法律上の通知

概要

本書では、Knative Serving の使用を開始する方法について説明します。このドキュメントでは、アプリケーションの設定方法を示し、自動スケーリング、トラフィック分割、外部および受信ルーティングなどの機能について説明します。

目次

第1章 KNATIVE SERVING を使い始める

1.1. SERVERLESS アプリケーション

サーバーレスアプリケーションは、ルートと設定で定義され、YAML ファイルに含まれる Kubernetes サービスとして作成およびデプロイされます。OpenShift Serverless を使用してサーバーレスアプリケーションをデプロイするには、Knative **Service** オブジェクトを作成する必要があります。

Knative Service オブジェクトの YAML ファイルの例

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

1. アプリケーションの名前。
2. アプリケーションが使用する namespace。
3. アプリケーションのイメージ
4. サンプルアプリケーションで出力される環境変数

以下の方法のいずれかを使用してサーバーレスアプリケーションを作成できます。

- OpenShift Container Platform Web コンソールからの Knative サービスの作成
  OpenShift Container Platform の場合は、開発者パースペクティブを使用したアプリケーションの作成 を参照してください。
- Knative (**kn**) CLI を使用して Knative サービスを作成します。
- **oc** CLI を使用して、Knative **Service** オブジェクトを YAML ファイルとして作成し、適用します。

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。

- 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>
```

ここでは、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 showcase \
    --image quay.io/openshift-knative/showcase
```

出力例

```
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. YAML を使用したサーバーレスアプリケーションの作成

YAML ファイルを使用して Knative リソースを作成する場合、宣言的 API を使用するため、再現性の高い方法でアプリケーションを宣言的に記述することができます。YAML を使用してサーバーレスアプリケーションを作成するには、Knative **Service** を定義する YAML ファイルを作成し、**oc apply** を使用してこれを適用する必要があります。

サービスが作成され、アプリケーションがデプロイされると、Knative はこのバージョンのアプリケーションのイミュータブルなリビジョンを作成します。また、Knative はネットワークプログラミングを実行し、アプリケーションのルート、ingress、サービスおよびロードバランサーを作成し、Pod をトラフィックに基づいて自動的にスケールアップ/ダウンします。

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。

- 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。

手順

1. 以下のサンプルコードを含む YAML ファイルを作成します。

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

2. YAML ファイルが含まれるディレクトリーに移動し、YAML ファイルを適用してアプリケーションをデプロイします。

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

OpenShift Container Platform Web コンソールで **Developer** パースペクティブに切り替えたくない場合、または Knative (**kn**) CLI または YAML ファイルを使用したくない場合は、OpenShift Container PlatformWeb コンソールの **Administator** パースペクティブを使用して Knative コンポーネントを作成できます。

1.1.3. Administrator パースペクティブを使用したサーバーレスアプリケーションの作成

サーバーレスアプリケーションは、ルートと設定で定義され、YAML ファイルに含まれる Kubernetes サービスとして作成およびデプロイされます。OpenShift Serverless を使用してサーバーレスアプリケーションをデプロイするには、Knative **Service** オブジェクトを作成する必要があります。

Knative Service オブジェクトの YAML ファイルの例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: showcase (1)
  namespace: default (2)
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/showcase (3)
```

```
    env:
      - name: GREET 4
        value: Ciao
```

1. アプリケーションの名前。
2. アプリケーションが使用する namespace。
3. アプリケーションのイメージ
4. サンプルアプリケーションで出力される環境変数

サービスが作成され、アプリケーションがデプロイされると、Knative はこのバージョンのアプリケーションのイミュータブルなリビジョンを作成します。また、Knative はネットワークプログラミングを実行し、アプリケーションのルート、ingress、サービスおよびロードバランサーを作成し、Pod をトラフィックに基づいて自動的にスケールアップ/ダウンします。

前提条件

Administrator パースペクティブを使用してサーバーレスアプリケーションを作成するには、以下の手順を完了していることを確認してください。

- OpenShift Serverless Operator および Knative Serving がインストールされていること。
- Web コンソールにログインしており、**Administrator** パースペクティブを使用している。

手順

1. **Serverless → Serving** ページに移動します。
2. **Create** リストで、**Service** を選択します。
3. YAML または JSON 定義を手動で入力するか、またはファイルをエディターにドラッグし、ドロップします。
4. **Create** をクリックします。

1.1.4. 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 製品の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。
- 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 showcase \
    --image quay.io/openshift-knative/showcase \
    --target ./ \
    --namespace test
```

出力例

```
Service 'showcase' 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と、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
        └── showcase.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名前が付けられる記述子ファイル **showcase.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/showcase.yaml
```

出力例

```
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. 新しいサービスに関する情報をリスト表示します。

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

出力例

```
Name:       showcase
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、**kn** は現在の namespace にちなんで名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/showcase.yaml
```

出力例

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

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

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

1.1.5. 関連情報

- Knative Serving CLI コマンド
- Knative サービスの JSON Web Token 認証の設定

1.2. サーバーレスアプリケーションのデプロイメントの確認

サーバーレスアプリケーションが正常にデプロイされたことを確認するには、Knative によって作成されたアプリケーション URL を取得してから、その URL に要求を送信し、出力を確認する必要があります。OpenShift Serverless は HTTP および HTTPS URL の両方の使用をサポートしますが、**oc get ksvc** からの出力は常に **http://** 形式を使用して URL を出力します。

1.2.1. サーバーレスアプリケーションのデプロイメントの確認

サーバーレスアプリケーションが正常にデプロイされたことを確認するには、Knative によって作成されたアプリケーション URL を取得してから、その URL に要求を送信し、出力を確認する必要があります。OpenShift Serverless は HTTP および HTTPS URL の両方の使用をサポートしますが、**oc get ksvc** からの出力は常に **http://** 形式を使用して URL を出力します。

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。
- **oc** CLI がインストールされている。
- Knative サービスを作成している。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. アプリケーション URL を検索します。

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

出力例

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

2. クラスターに対して要求を実行し、出力を確認します。

HTTP リクエストの例 (HTTPie ツールを使用)

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

HTTPS 要求の例

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

出力例

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

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

3. オプション: システムに HTTPie ツールがインストールされていない場合は、代わりに curl ツールを使用できる可能性があります。

HTTPS 要求の例

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

出力例

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

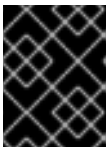
4. オプション: 証明書チェーンで自己署名証明書に関連するエラーが発生した場合は、HTTPPie コマンドに **--verify=no** フラグを追加して、エラーを無視できます。

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

出力例

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

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



重要

自己署名証明書は、実稼働デプロイメントでは使用しないでください。この方法は、テスト目的にのみ使用されます。

5. オプション:OpenShift Container Platform クラスターが認証局 (CA) で署名されているが、システムにグローバルに設定されていない証明書で設定されている場合、**curl** コマンドでこれを指定できます。証明書へのパスは、**--cacert** フラグを使用して curl コマンドに渡すことができます。

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

出力例

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

第2章 自動スケーリング

2.1. 自動スケーリング

Knative Serving は、アプリケーションが受信要求に一致するように、自動スケーリング (**autoscaling**) を提供します。たとえば、アプリケーションがトラフィックを受信せず、scale-to-zero が有効にされている場合、Knative Serving はアプリケーションをゼロレプリカにスケールダウンします。scale-to-zero が無効になっている場合、アプリケーションはクラスターのアプリケーションに設定された最小のレプリカ数にスケールダウンされます。アプリケーションへのトラフィックが増加したら、要求を満たすようにレプリカをスケールアップすることもできます。

Knative サービスの自動スケーリング設定は、クラスター管理者 (または Red Hat OpenShift Service on AWS および OpenShift Dedicated の専用管理者) によって設定されるグローバル設定、または個々のサービスに対して設定されるリビジョンごとに設定できます。

OpenShift Container Platform Web コンソールを使用して、サービスの YAML ファイルを変更するか、または Knative (**kn**) CLI を使用して、サービスのリビジョンごとの設定を変更できます。

注記

サービスに設定した制限またはターゲットは、アプリケーションの単一インスタンスに対して測定されます。たとえば、**target** アノテーションを **50** に設定することにより、各リビジョンが一度に 50 の要求を処理できるようアプリケーションをスケーリングするように Autoscaler が設定されます。

2.2. スケーリング限度

スケーリング限度は、任意の時点でアプリケーションに対応できる最小および最大のレプリカ数を決定します。アプリケーションのスケーリング限度を設定して、コールドスタートを防止したり、コンピューティングコストを制御したりできます。

2.2.1. スケーリング下限

アプリケーションにサービスを提供できるレプリカの最小数は、最小 **min-scale** のアノテーションによって決定されます。ゼロへのスケーリングが有効になっていない場合、**min-Scale** 値のデフォルトは **1** になります。

次の条件が満たされた場合、**min-scale** 値はデフォルトで **0** レプリカになります。

- **mi-scale** のアノテーションが設定されていません
- ゼロへのスケーリングが有効にされている
- **KPA** クラスが使用されている

min-scale アノテーションを使用したサービス仕様の例

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

```
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "0"
...
```

2.2.1.1. Knative CLI を使用した最小スケールアノテーションの設定

minScale アノテーションを設定するために Knative (**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提供されます。**kn service** コマンドを **--scale-min** フラグと共に使用して、サービスの **--min-scale** 値を作成または変更できます。

前提条件

- 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

- **--scale-min** フラグを使用して、サービスのレプリカの最小数を設定します。

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

コマンドの例

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

2.2.2. スケーリング上限

アプリケーションにサービスを提供できるレプリカの最大数は、**max-scale** アノテーションによって決定されます。**max-scale** アノテーションが設定されていない場合、作成されるレプリカの数に上限はありません。

max-scale アノテーションを使用したサービス仕様の例

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

2.2.2.1. Knative CLI を使用した最大スケールアノテーションの設定

Knative (**kn**) CLI を使用して **max-scale** のアノテーションを設定すると、YAML ファイルを直接変更する場合に比べ、ユーザーインターフェイスがより合理的で直感的です。**--scale-max** フラグを指定して knservice コマンドを使用すると、**kn service** の **max-scale** 値を作成または変更できます。

前提条件

- 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

- **--scale-max** フラグを使用して、サービスのレプリカの最大数を設定します。

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

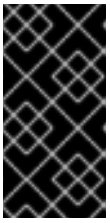
コマンドの例

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

2.3. 並行処理性

並行処理性は、特定の時点でアプリケーションの各レプリカが処理できる同時リクエストの数を決定します。並行処理性は、**ソフトリミット**または**ハードリミット**のいずれかとして設定できます。

- ソフトリミットは、厳格に強制される限度ではなく、目標となるリクエストの限度です。たとえば、トラフィックの急増が発生した場合、ソフトリミットのターゲットを超過できます。
- ハードリミットは、リクエストに対して厳密に適用される上限です。並行処理がハードリミットに達すると、それ以降のリクエストはバッファー処理され、リクエストを実行するのに十分な空き容量ができるまで待機する必要があります。



重要

ハードリミット設定の使用は、アプリケーションに明確なユースケースがある場合にのみ推奨されます。ハードリミットを低い値に指定すると、アプリケーションのスループットとレイテンシーに悪影響を与える可能性があり、コールドスタートが発生する可能性があります。

ソフトターゲットとハードリミットを追加することは、Autoscaler は同時リクエストのソフトターゲット数を目標とするが、リクエストの最大数にハードリミット値のハードリミットを課すことを意味します。

ハードリミットの値がソフトリミットの値より小さい場合、実際に処理できる数よりも多くのリクエストを目標にする必要がないため、ソフトリミットの値が低減されます。

2.3.1. ソフト並行処理ターゲットの設定

ソフトリミットは、厳格に強制される限度ではなく、目標となるリクエストの限度です。たとえば、トラフィックの急増が発生した場合、ソフトリミットのターゲットを超過できます。**autoscaling.knative.dev/target** アノテーションを仕様に設定するか、または正しいフラグを指定して **kn service** コマンドを使用して、Knative サービスにソフト並行処理ターゲットを指定できます。

手順

- オプション:**Service** カスタムリソースの仕様で Knative サービスに **autoscaling.knative.dev/target** アノテーションを設定します。

サービス仕様の例

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

- オプション:**kn service** コマンドを使用して **--concurrency-target** フラグを指定します。

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

並行処理のターゲットを 50 リクエストに設定したサービスを作成するコマンドの例

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

2.3.2. ハード並行処理リミットの設定

ハード並行処理リミットは、リクエストに対して厳密に適用される上限です。並行処理がハードリミットに達すると、それ以降のリクエストはバッファー処理され、リクエストを実行するのに十分な空き容量ができるまで待機する必要があります。**containerConcurrency** 仕様を変更するか、または正しいフラグを指定して **kn service** コマンドを使用して、Knative サービスにハード並行処理リミットを指定できます。

手順

- オプション:**Service** カスタムリソースの仕様で Knative サービスに **containerConcurrency** 仕様を設定します。

サービス仕様の例

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

デフォルト値は **0** です。これは、サービスの 1 つのレプリカに一度に流れることができる同時リクエストの数に制限がないことを意味します。

0 より大きい値は、サービスの 1 つのレプリカに一度に流れることができるリクエストの正確な数を指定します。この例では、50 リクエストのハード並行処理リミットを有効にします。

- オプション:**kn service** コマンドを使用して **--concurrency-limit** フラグを指定します。

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

並行処理のリミットを 50 リクエストに設定したサービスを作成するコマンドの例

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

2.3.3. 並行処理ターゲットの使用率

この値は、Autoscaler が実際に目標とする並行処理リミットのパーセンテージを指定します。これは、レプリカが実行する **ホット度** を指定することとも呼ばれます。これにより、Autoscaler は定義されたハードリミットに達する前にスケールアップできるようになります。

たとえば、**containerConcurrency** 値が 10 に設定され、 **target-utilization-percentage** 値が 70% に設定されている場合、既存のすべてのレプリカの同時リクエストの平均数が 7 に達すると、オートスケーラーは新しいレプリカを作成します。7 から 10 の番号が付けられたリクエストは引き続き既存のレプリカに送信されますが、**containerConcurrency** 値に達した後、必要になることを見越して追加のレプリカが開始されます。

target-utilization-percentage アノテーションを使用して設定されたサービスの例

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

2.4. SCALE-TO-ZERO

Knative Serving は、アプリケーションが受信要求に一致するように、自動スケーリング (**autoscaling**) を提供します。

2.4.1. scale-to-zero の有効化

enable-scale-to-zero 仕様を使用して、クラスター上のアプリケーションの scale-to-zero をグローバルに有効または無効にすることができます。

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。
- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。

- デフォルトの Knative Pod Autoscaler を使用している。Kubernetes Horizontal Pod Autoscaler を使用している場合は、ゼロにスケーリングすることはできません。

手順

- **KnativeServing** カスタムリソース (CR) の **enable-scale-to-zero** 仕様を変更します。

  KnativeServing CR の例

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

1 **enable-scale-to-zero** 仕様は、**true** または **false** のいずれかです。true に設定すると、scale-to-zero が有効にされます。false に設定すると、アプリケーションは設定された **スケーリング下限** にスケールダウンされます。デフォルト値は **"true"** です。

2.4.2. scale-to-zero 猶予期間の設定

Knative Serving は、アプリケーションの Pod をゼロにスケールダウンします。**scale-to-zero-grace-period** 仕様を使用して、アプリケーションの最後のレプリカが削除される前に Knative が scale-to-zero 機構が配置されるのを待機する上限時間を定義できます。

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。
- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。
- デフォルトの Knative Pod Autoscaler を使用している。Kubernetes Horizontal Pod Autoscaler を使用している場合は、ゼロにスケーリングすることはできません。

手順

- **KnativeServing** カスタムリソース CR の **scale-to-zero-grace-period** 仕様を変更します。

  KnativeServing CR の例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
```

```
config:
  autoscaler:
    scale-to-zero-grace-period: "30s" (1)
```

(1) 猶予期間 (秒単位)。デフォルト値は 30 秒です。

第3章 SERVERLESS アプリケーションの設定

3.1. KNATIVE SERVING システムのデプロイメント設定のオーバーライド

KnativeServing カスタムリソース (CR) の **deployments** 仕様を変更して、特定のデプロイメントのデフォルト設定を上書きできます。

注記

デフォルトでデプロイメントに定義されているプローブのみをオーバーライドできます。

Knative Serving デプロイメントはすべて、以下の例外を除き、デフォルトで readiness および liveness プローブを定義します。

- **net-kourier-controller** および **3scale-kourier-gateway** は readiness プローブのみを定義します。
- **net-istio-controller** および **net-istio-webhook** はプローブを定義しません。

3.1.1. システムのデプロイメント設定の上書き

現在、**resources**、**replicas**、**labels**、**annotations**、**nodeSelector** フィールド、およびプローブの **readiness** と **liveness** フィールドで、デフォルトの構成設定のオーバーライドがサポートされています。

以下の例では、**KnativeServing** CR は **webhook** デプロイメントをオーバーライドし、以下を確認します。

- **net-kourier-controller** の **readiness** プローブのタイムアウトは 10 秒に設定されています。
- デプロイメントには、CPU およびメモリーのリソース制限が指定されています。
- デプロイメントには 3 つのレプリカがあります。
- **example-label:labellabel** が追加されました。
- **example-annotation: annotation** が追加されます。
- **nodeSelector** フィールドは、**disktype: hdd** ラベルを持つノードを選択するように設定されます。

注記

KnativeServing CR ラベルおよびアノテーション設定は、デプロイメント自体と結果として生成される Pod の両方のデプロイメントのラベルおよびアノテーションを上書きします。

KnativeServing CR の例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: ks
```

```
  namespace: knative-serving
spec:
  high-availability:
    replicas: 2
  deployments:
  - name: net-kourier-controller
    readinessProbes: 1
      - container: controller
        timeoutSeconds: 10
  - name: webhook
    resources:
    - container: webhook
      requests:
        cpu: 300m
        memory: 60Mi
      limits:
        cpu: 1000m
        memory: 1000Mi
    replicas: 3
    labels:
      example-label: label
    annotations:
      example-annotation: annotation
    nodeSelector:
      disktype: hdd
```

1 **readiness** および **liveness** プローブオーバーライドを使用して、プローブハンドラーに関連するフィールド (**exec**、**grpc**、**httpGet**、および **tcpSocket**) を除き、Kubernetes API で指定されているデプロイメントのコンテナー内のプローブのすべてのフィールドをオーバーライドできます。

関連情報

- Kubernetes API ドキュメントのプローブ設定セクション

3.2. SERVING のマルチコンテナーサポート

単一の Knative サービスを使用してマルチコンテナー Pod をデプロイできます。この方法は、アプリケーションの役割を小さく特化した部分に分離する場合に便利です。

3.2.1. マルチコンテナーサービスの設定

マルチコンテナーのサポートはデフォルトで有効になっています。サービス内の複数のコンテナーを指定してマルチコンテナー Pod を作成できます。

手順

1. サービスを変更して、追加のコンテナーを追加します。リクエストを処理できるコンテナーは 1 つだけであるため、**ポート** は 1 つのコンテナーにのみ指定してください。以下は、2 つのコンテナーの設定例です。

   複数のコンテナー設定

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

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

1. 最初のコンテナー設定。
2. 最初のコンテナーのポート仕様。
3. 2 つ目のコンテナー設定。

3.3. EMPTYDIR ボリューム

emptyDir ボリュームは、Pod の作成時に作成される空のボリュームであり、一時的な作業ディスク領域を提供するために使用されます。**emptyDir** ボリュームは、それらが作成された Pod が削除されると削除されます。

3.3.1. EmptyDir 拡張機能の設定

kubernetes.podspec-volumes-emptydir の拡張は、**emptyDir** ボリュームを Knative Serving で使用できるかどうかを制御します。**emptyDir** ボリュームの使用を有効にするには、**KnativeServing** カスタムリソース (CR) を変更して以下の YAML を追加する必要があります。

KnativeServing CR の例

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

3.4. 配信のための永続ボリューム要求

一部のサーバーレスアプリケーションには、永続的なデータストレージが必要です。これを実現するために、Knative サービスの永続ボリュームクレーム (PVC) を設定できます。

3.4.1. PVC サポートの有効化

手順

1. Knative Serving が PVC を使用して書き込むことができるようにするには、**KnativeServing** カスタムリソース (CR) を変更して次の YAML を含めます。

書き込みアクセスで PVC を有効にする

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

- **kubernetes.podspec-persistent-volume-claim** 拡張機能は、永続ボリューム (PV) を Knative Serving で使用できるかどうかを制御します。
- **kubernetes.podspec-persistent-volume-write** 拡張機能は、書き込みアクセスで Knative Serving が PV を利用できるかどうかを制御します。

2. PV を要求するには、PV 設定を含めるようにサービスを変更します。たとえば、次の設定で永続的なボリュームクレームがある場合があります。

注記

要求しているアクセスモードをサポートするストレージクラスを使用してください。たとえば、**ReadWriteMany** アクセスモードの **ocs-storagecluster-cephfs** クラスを使用できます。

PersistentVolumeClaim 設定

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

この場合、書き込みアクセス権を持つ PV を要求するには、次のようにサービスを変更します。

ネイティブサービス PVC 設定

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  namespace: my-ns
...
spec:
```

```
template:
  spec:
    containers:
      ...
      volumeMounts: (1)
        - mountPath: /data
          name: mydata
          readOnly: false
    volumes:
      - name: mydata
        persistentVolumeClaim: (2)
          claimName: example-pv-claim
          readOnly: false (3)
```

(1) ボリュームマウント仕様。

(2) 永続的なボリュームクレームの仕様。

(3) 読み取り専用アクセスを有効にするフラグ。

注記

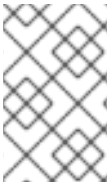
Knative サービスで永続ストレージを正常に使用するには、Knative コンテナーユーザーのユーザー権限などの追加の設定が必要です。

3.4.2. OpenShift Container Platform の関連情報

- 永続ストレージについて

3.5. INIT コンテナー

Init コンテナー は、Pod 内のアプリケーションコンテナーの前に実行される特殊なコンテナーです。これらは通常、アプリケーションの初期化ロジックを実装するために使用されます。これには、セットアップスクリプトの実行や、必要な設定のダウンロードが含まれる場合があります。**KnativeServing** カスタムリソース (CR) を変更することにより、Knative サービスの init コンテナーの使用を有効にできます。



注記

Init コンテナーを使用すると、アプリケーションの起動時間が長くなる可能性があるため、頻繁にスケールアップおよびスケールダウンすることが予想されるサーバーレスアプリケーションには注意して使用する必要があります。

3.5.1. init コンテナーの有効化

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。

- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。

手順

- **KnativeServing** CR に **kubernetes.podspec-init-containers** フラグを追加して、init コンテナーの使用を有効にします。

  KnativeServing CR の例

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

3.6. イメージタグのダイジェストへの解決

Knative Serving コントローラーがコンテナーレジストリーにアクセスできる場合、Knative Serving は、サービスのリビジョンを作成するときにイメージタグをダイジェストに解決します。これは**タグからダイジェストへの解決**と呼ばれ、デプロイメントの一貫性を提供するのに役立ちます。

3.6.1. タグからダイジェストへの解決

コントローラーに OpenShift Container Platform のコンテナーレジストリーへのアクセスを許可するには、シークレットを作成してから、コントローラーのカスタム証明書を設定する必要があります。**KnativeServing** カスタムリソース (CR) の **controller-custom-certs** 仕様を変更することにより、コントローラーカスタム証明書を設定できます。シークレットは、**KnativeServing** CR と同じ namespace に存在する必要があります。

シークレットが **KnativeServing** CR に含まれていない場合、この設定はデフォルトで公開鍵インフラストラクチャー (PKI) を使用します。PKI を使用する場合、クラスター全体の証明書は、**config-service-sa** Config Map を使用して KnativeServing コントローラーに自動的に挿入されます。OpenShift Serverless Operator は、**config-service-sa** Config Map にクラスター全体の証明書を設定し、Config Map をボリュームとしてコントローラーにマウントします。

3.6.1.1. シークレットを使用したタグからダイジェストへの解決の設定

controller-custom-certs 仕様で **Secret** タイプが使用されている場合、シークレットはシークレットボリュームとしてマウントされます。シークレットに必要な証明書があると仮定すると、ネイティブコンポーネントはシークレットを直接消費します。

前提条件

- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。

- 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。

手順

1. シークレットを作成します。

   コマンドの例

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

2. **Secret** タイプを使用するように、**KnativeServing** カスタムリソース (CR) で **controller-custom-certs** 仕様を設定します。

   KnativeServing CR の例

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

3.7. TLS 認証の設定

Transport Layer Security (TLS) を使用して、Knative トラフィックを暗号化し、認証することができます。

TLS は、Knative Kafka のトラフィック暗号化でサポートされている唯一の方法です。Red Hat は、Apache Kafka リソースの Knative ブローカーに SASL と TLS の両方を併用することを推奨します。

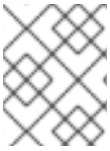
注記

Red Hat OpenShift Service Mesh 統合で内部 TLS を有効にする場合は、以下の手順で説明する内部暗号化の代わりに、mTLS で Service Mesh を有効にする必要があります。

OpenShift Container Platform および Red Hat OpenShift Service on AWS の場合は、mTLS で Service Mesh を使用する場合の Knative Serving メトリクスの有効化 ドキュメント を参照してください。

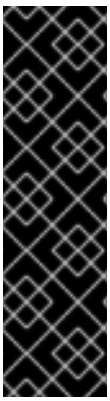
3.7.1. 内部トラフィックの TLS 認証を有効にする

OpenShift Serverless はデフォルトで TLS エッジターミネーションをサポートしているため、エンドユーザーからの HTTPS トラフィックは暗号化されます。ただし、OpenShift ルートの背後にある内部トラフィックは、プレーンデータを使用してアプリケーションに転送されます。内部トラフィックに対して TLS を有効にすることで、コンポーネント間で送信されるトラフィックが暗号化され、このトラフィックがより安全になります。



注記

Red Hat OpenShift Service Mesh 統合で内部 TLS を有効にする場合は、以下の手順で説明する内部暗号化の代わりに、mTLS で Service Mesh を有効にする必要があります。



重要

内部 TLS 暗号化のサポートは、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 製品の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 がインストールされている。
- OpenShift (**oc**) CLI がインストールされている。

手順

1. **KnativeServing** リソースを作成または更新し、仕様に **external-encryption: "true"** フィールドが含まれていることを確認します。

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

2. **knative-serving** namespace でアクティベーター Pod を再起動して、証明書を読み込みます。

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

関連情報

- Apache Kafka の Knative ブローカーの TLS 認証設定
- Apache Kafka のチャネルの TLS 認証設定
- mTLS で Service Mesh を使用する場合の Knative Serving メトリックの有効化

3.8. 制限のあるネットワークポリシー

3.8.1. 制限のあるネットワークポリシーを持つクラスター

複数のユーザーがアクセスできるクラスターを使用している場合、クラスターはネットワークポリシーを使用してネットワーク経由で相互に通信できる Pod、サービス、および namespace を制御する可能

性があります。クラスターで制限的なネットワークポリシーを使用する場合は、Knative システム Pod が Knative アプリケーションにアクセスできない可能性があります。たとえば、namespace に、すべての要求を拒否する以下のネットワークポリシーがある場合、Knative システム Pod は Knative アプリケーションにアクセスできません。

namespace へのすべての要求を拒否する NetworkPolicy オブジェクトの例

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

3.8.2. 制限のあるネットワークポリシーを持つクラスターでの Knative アプリケーションとの通信の有効化

Knative システム Pod からアプリケーションへのアクセスを許可するには、ラベルを各 Knative システム namespace に追加し、このラベルを持つ他の namespace の namespace へのアクセスを許可する アプリケーション namespace に **NetworkPolicy** オブジェクトを作成する必要があります。

重要

クラスターの非 Knative サービスへの要求を拒否するネットワークポリシーは、これらのサービスへのアクセスを防止するネットワークポリシーです。ただし、Knative システム namespace から Knative アプリケーションへのアクセスを許可することにより、クラスターのすべての namespace から Knative アプリケーションへのアクセスを許可する必要があります。

クラスターのすべての namespace から Knative アプリケーションへのアクセスを許可しない場合は、代わりに **Knative サービスの JSON Web Token 認証**を使用するようにしてください。Knative サービスの JSON Web トークン認証にはサービスメッシュが必要です。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- 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。

手順

1. アプリケーションへのアクセスを必要とする各 Knative システム namespace に **knative.openshift.io/system-namespace=true** ラベルを追加します。
   a. **knative-serving** namespace にラベルを付けます。

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

   b. **knative-serving-ingress** namespace にラベルを付けます。

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

c. **knative-eventing namespace** にラベルを付けます。

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

d. **knative-kafka namespace** にラベルを付けます。

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

2. アプリケーション namespace で **NetworkPolicy** オブジェクトを作成し、**knative.openshift.io/system-namespace** ラベルのある namespace からのアクセスを許可します。

サンプル NetworkPolicy オブジェクト

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

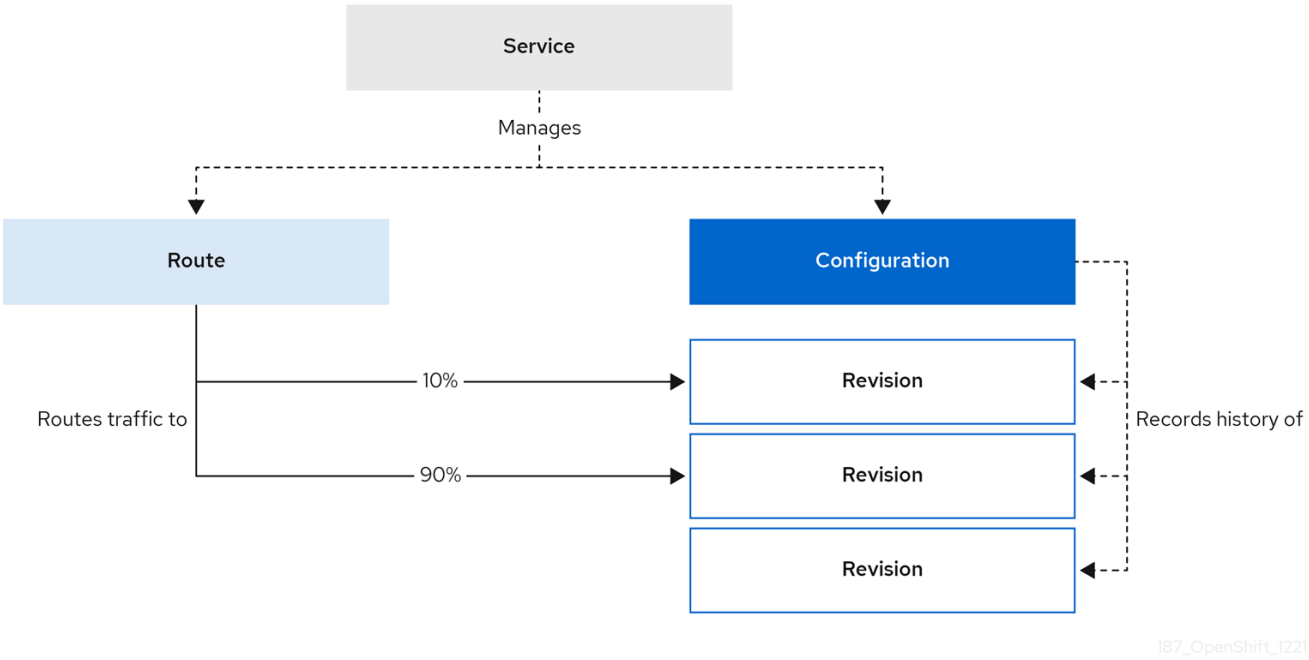
1 ネットワークポリシーの名前を指定します。

2 アプリケーションが存在する namespace。

第4章 トラフィック分割

4.1. トラフィック分割の概要

Knative アプリケーションでは、トラフィック分割を作成することでトラフィックを管理できます。トラフィック分割は、Knative サービスによって管理されるルートの一部として設定されます。



ルートを設定すると、サービスのさまざまなリビジョンにリクエストを送信できます。このルーティングは、**Service** オブジェクトの **traffic** 仕様によって決定されます。

traffic 仕様宣言は、1 つ以上のリビジョンで設定され、それぞれがトラフィック全体の一部を処理する責任があります。各リビジョンにルーティングされるトラフィックの割合は、合計で 100% になる必要があります。これは、Knative 検証によって保証されます。

traffic 仕様で指定されたリビジョンは、固定の名前付きリビジョンにすることも、サービスのすべてのリビジョンのリストの先頭を追跡する最新のリビジョンを指すこともできます。最新のリビジョンは、新しいリビジョンが作成された場合に更新される一種のフローティング参照です。各リビジョンには、そのリビジョンの追加のアクセス URL を作成するタグを付けることができます。

traffic 仕様は次の方法で変更できます。

- **Service** オブジェクトの YAML を直接編集します。
- Knative (**kn**) CLI **--traffic** フラグを使用します。
- OpenShift Container Platform Web コンソールの使用

Knative サービスの作成時に、デフォルトの **traffic** 仕様設定は含まれません。

4.2. トラフィックスペックの例

以下の例は、トラフィックの 100% がサービスの最新リビジョンにルーティングされる **traffic** 仕様を示しています。**status** では、**latestRevision** が解決する最新リビジョンの名前を確認できます。

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

以下の例は、トラフィックの 100% が **current** としてタグ付けされたリビジョンにルーティングされ、そのリビジョンの名前が **example-service** として指定される **traffic** 仕様を示しています。**latest** とタグ付けされたリビジョンは、トラフィックが宛先にルーティングされない場合でも、利用可能な状態になります。

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

以下の例は、トラフィックが複数のリビジョン間で分割されるように、**traffic** 仕様のリビジョンの一覧を拡張する方法を示しています。この例では、トラフィックの 50% を、**current** としてタグ付けされたリビジョンに送信します。また、**candidate** としてタグ付けされたリビジョンにトラフィックの 50% を送信します。**latest** とタグ付けされたリビジョンは、トラフィックが宛先にルーティングされない場合でも、利用可能な状態になります。

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

```
    percent: 50
  - tag: latest
    latestRevision: true
    percent: 0
```

4.3. KNATIVE CLI を使用したトラフィック分割

Knative (**kn**) CLI を使用してトラフィック分割を作成すると、YAML ファイルを直接変更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提供されます。**kn service update** コマンドを使用して、サービスのリビジョン間でトラフィックを分割できます。

4.3.1. KnativeCLI を使用してトラフィック分割を作成する

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。
- Knative サービスを作成している。

手順

- 標準の **kn service update** コマンドで **--traffic** タグを使用して、サービスのリビジョンとそれにルーティングするトラフィックの割合を指定します。

  コマンドの例

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

  ここでは、以下のようになります。

  - **<service_name>** は、トラフィックルーティングを設定する Knative サービスの名前です。
  - **<revision>** は、一定の割合のトラフィックを受信するように設定するリビジョンです。リビジョンの名前、または **--tag** フラグを使用してリビジョンに割り当てたタグのいずれかを指定できます。
  - **<percentage>** は、指定されたリビジョンに送信するトラフィックのパーセンテージです。

- オプション: **--traffic** フラグは、1 つのコマンドで複数回指定できます。たとえば、**@latest** というタグの付いたリビジョンと **stable** という名前のリビジョンがある場合、次のように各リビジョンに分割するトラフィックの割合を指定できます。

  コマンドの例

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

  複数のリビジョンがあり、最後のリビジョンに分割する必要があるトラフィックの割合を指定しない場合、**-traffic** フラグはこれを自動的に計算できます。たとえば、**example** という名前の 3 番目のリビジョンがあり、次のコマンドを使用する場合:

コマンドの例

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

トラフィックの残りの 30% は、指定されていなくても、**example** リビジョンに分割されます。

4.4. トラフィック分割の CLI フラグ

Knative (**kn**) CLI は **kn service update** コマンドの一環として、サービスのトラフィックブロックでのトラフィック操作をサポートします。

4.4.1. Knative CLI トラフィック分割フラグ

以下の表は、トラフィック分割フラグ、値の形式、およびフラグが実行する操作の概要を表示しています。**Repetition** 列は、フラグの特定の値が **kn service update** コマンドで許可されるかどうかを示します。

フラグ	値	操作	繰り返し
--traffic	**RevisionName=Percent**	**Percent** トラフィックを **RevisionName** に指定します。	はい
--traffic	**Tag=Percent**	**Percent** トラフィックを、**Tag** を持つリビジョンに指定します。	はい
--traffic	**@latest=Percent**	**Percent** トラフィックを準備状態にある最新のリビジョンに指定します。	いいえ
--tag	**RevisionName=Tag**	**Tag** を **RevisionName** に指定します。	はい
--tag	**@latest=Tag**	**Tag** を準備状態にある最新リビジョンに指定します。	いいえ
--untag	**Tag**	リビジョンから **Tag** を削除します。	はい

4.4.1.1. 複数のフラグおよび順序の優先順位

すべてのトラフィック関連のフラグは、単一の **kn service update** コマンドを使用して指定できます。**kn** は、これらのフラグの優先順位を定義します。コマンドの使用時に指定されるフラグの順番は考慮に入れられません。

kn で評価されるフラグの優先順位は以下のとおりです。

1. **--untag**: このフラグで参照されるすべてのリビジョンはトラフィックブロックから削除されます。
2. **--tag**: リビジョンはトラフィックブロックで指定されるようにタグ付けされます。
3. **--traffic**: 参照されるリビジョンには、分割されたトラフィックの一部が割り当てられます。

タグをリビジョンに追加してから、設定したタグに応じてトラフィックを分割することができます。

4.4.1.2. リビジョンのカスタム URL

kn service update コマンドを使用して **--tag** フラグをサービスに割り当てると、サービスの更新時に作成されるリビジョンのカスタム URL が作成されます。カスタム URL は、**https://<tag>-<service_name>-<namespace>.<domain>** パターンまたは **http://<tag>-<service_name>-<namespace>.<domain>** パターンに従います。

--tag フラグおよび **--untag** フラグは以下の構文を使用します。

- 1 つの値が必要です。
- サービスのトラフィックブロックに一意のタグを示します。
- 1 つのコマンドで複数回指定できます。

4.4.1.2.1. 例: リビジョンへのタグの割り当て

以下の例では、タグ **latest** を、**example-revision** という名前のリビジョンに割り当てます。

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

4.4.1.2.2. 例: リビジョンからのタグの削除

--untag フラグを使用して、カスタム URL を削除するタグを削除できます。

注記

リビジョンのタグが削除され、トラフィックの 0% が割り当てられる場合、リビジョンはトラフィックブロックから完全に削除されます。

以下のコマンドは、**example-revision** という名前のリビジョンからすべてのタグを削除します。

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

4.5. リビジョン間でのトラフィックの分割

サーバーレスアプリケーションの作成後、アプリケーションは OpenShift Container Platform Web コンソールの **Developer** パースペクティブの **Topology** ビューに表示されます。アプリケーションのリビジョンはノードによって表され、Knative サービスはノードの周りの四角形のマークが付けられます。

コードまたはサービス設定の新たな変更により、特定のタイミングでコードのスナップショットである新規リビジョンが作成されます。サービスの場合、必要に応じてこれを分割し、異なるリビジョンにルーティングして、サービスのリビジョン間のトラフィックを管理することができます。

4.5.1. OpenShift Container Platform Web コンソールを使用したリビジョン間のトラフィックの管理

前提条件

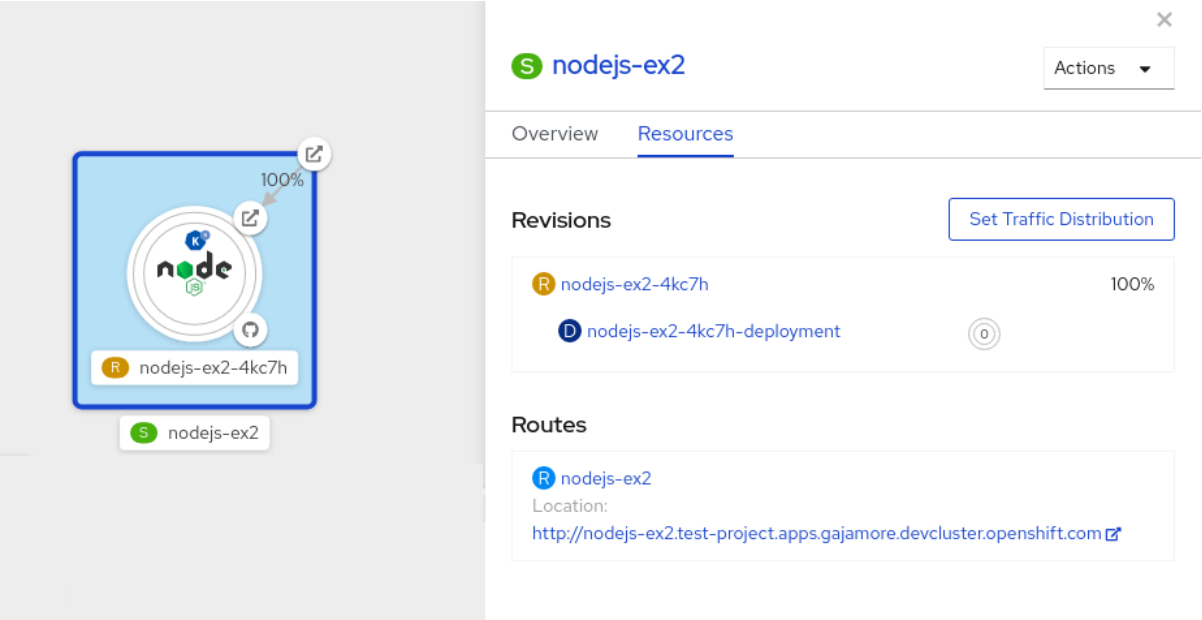
- OpenShift Serverless Operator および Knative Serving がクラスターにインストールされている。
- OpenShift Container Platform Web コンソールにログインしている。

手順

Topology ビューでアプリケーションの複数のリビジョン間でトラフィックを分割するには、以下を行います。

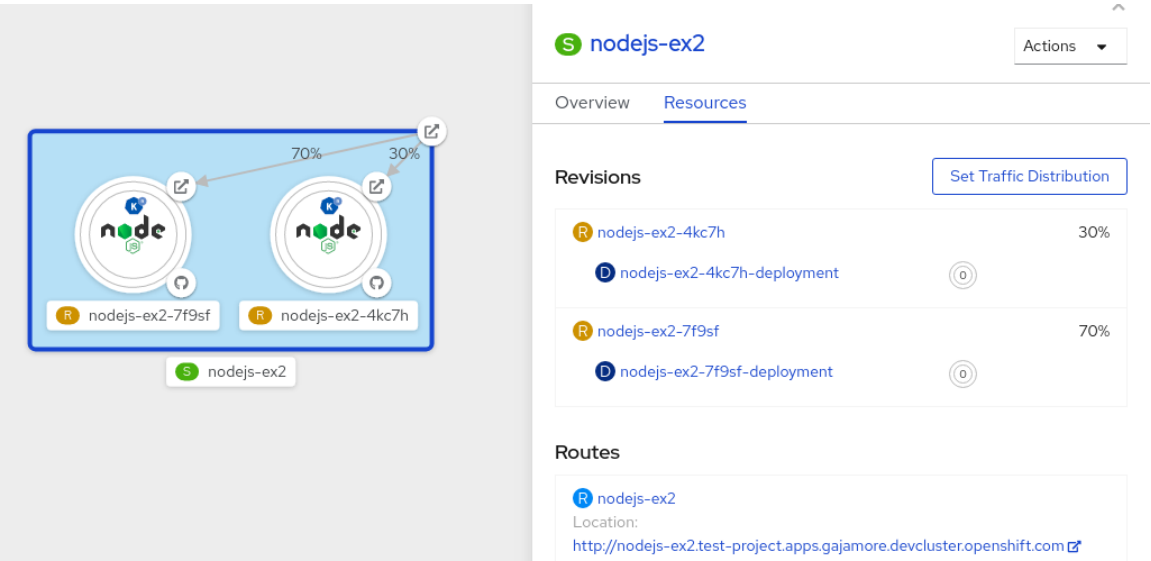
1. Knative サービスをクリックし、サイドパネルの概要を表示します。
2. **Resources** タブをクリックして、サービスの **Revisions** および **Routes** の一覧を表示します。

図4.1 Serverless アプリケーション



3. サイドパネルの上部にある **S** アイコンで示されるサービスをクリックし、サービスの詳細の概要を確認します。
4. **YAML** タブをクリックし、YAML エディターでサービス設定を変更し、**Save** をクリックします。たとえば、**timeoutseconds** を 300 から 301 に変更します。この設定の変更により、新規リビジョンがトリガーされます。**Topology** ビューでは、最新のリビジョンが表示され、サービスの **Resources** タブに 2 つのリビジョンが表示されるようになります。
5. **Resources** タブで **Set Traffic Distribution** をクリックして、トラフィック分配ダイアログボックスを表示します。
   a. **Splits** フィールドに、2 つのリビジョンのそれぞれの分割されたトラフィックパーセンテージを追加します。
   b. 2 つのリビジョンのカスタム URL を作成するタグを追加します。
   c. **Save** をクリックし、Topology ビューで 2 つのリビジョンを表す 2 つのノードを表示します。

図4.2 Serverless アプリケーションのリビジョン



4.6. ブルーグリーン戦略を使用したトラフィックの再ルーティング

Blue-green デプロイメントストラテジー を使用して、実稼働バージョンのアプリケーションから新規バージョンにトラフィックを安全に再ルーティングすることができます。

4.6.1. blue-green デプロイメントストラテジーを使用したトラフィックのルーティングおよび管理

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。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. アプリケーションを Knative サービスとして作成し、デプロイします。
2. 以下のコマンドから出力を表示して、サービスのデプロイ時に作成された最初のリビジョンの名前を検索します。

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

コマンドの例

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

出力例

```
$ showcase-00001
```

3. 以下の YAML をサービスの **spec** に追加して、受信トラフィックをリビジョンに送信します。

```
...
```

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

4. 以下のコマンドを実行して、URL の出力でアプリケーションを表示できることを確認します。

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

5. サービスの **template** 仕様の少なくとも 1 つのフィールドを変更してアプリケーションの 2 番目のリビジョンをデプロイし、これを再デプロイします。たとえば、サービスの **image** や **env** 環境変数を変更できます。サービスの再デプロイは、サービスの YAML ファイルを適用するか、Knative (**kn**) CLI をインストールしている場合は、**kn service update** コマンドを使用します。

6. 以下のコマンドを実行して、サービスを再デプロイする際に作成された 2 番目の最新のリビジョンの名前を見つけます。

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

この時点で、サービスの最初のバージョンと 2 番目のリビジョンの両方がデプロイされ、実行されます。

7. 既存のサービスを更新して、2 番目のリビジョンの新規テストエンドポイントを作成し、他のすべてのトラフィックを最初のリビジョンに送信します。

テストエンドポイントのある更新されたサービス仕様の例

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

YAML リソースを再適用してこのサービスを再デプロイすると、アプリケーションの 番目のリビジョンがステージングされます。トラフィックはメインの URL の 2 番目のリビジョンにルーティングされず、Knative は新たにデプロイされたリビジョンをテストするために **v2** という名前の新規サービスを作成します。

8. 以下のコマンドを実行して、2 番目のリビジョンの新規サービスの URL を取得します。

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

この URL を使用して、トラフィックをルーティングする前に、新しいバージョンのアプリケーションが予想通りに機能していることを検証できます。

9. 既存のサービスを再度更新して、トラフィックの 50% が最初のリビジョンに送信され、50% が 2 番目のリビジョンに送信されます。

リビジョン間でトラフィックを 50/50 に分割する更新サービス仕様の例

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

10. すべてのトラフィックを新しいバージョンのアプリケーションにルーティングできる状態になったら、再度サービスを更新して、100% のトラフィックを 2 番目のリビジョンに送信します。

すべてのトラフィックを 2 番目のリビジョンに送信する更新済みのサービス仕様の例

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

ヒント

リビジョンのロールバックを計画しない場合は、これを 0% に設定する代わりに最初のリビジョンを削除できます。その後、ルーティング不可能なリビジョンオブジェクトにはガベージコレクションが行われます。

11. 最初のリビジョンの URL にアクセスして、アプリケーションの古いバージョンに送信されていないことを確認します。

第5章 外部およびイングレスルーティング

5.1. ルーティングの概要

Knative は OpenShift Container Platform TLS 終端を使用して Knative サービスのルーティングを提供します。Knative サービスが作成されると、OpenShift Container Platform ルートがサービス用に自動的に作成されます。このルートは OpenShift Serverless Operator によって管理されます。OpenShift Container Platform ルートは、OpenShift Container Platform クラスターと同じドメインで Knative サービスを公開します。

OpenShift Container Platform ルーティングの Operator 制御を無効にすることで、Knative ルートを TLS 証明書を直接使用するように設定できます。

Knative ルートは OpenShift Container Platform ルートと共に使用し、トラフィック分割などの詳細なルーティング機能を提供します。

5.1.1. OpenShift Container Platform の関連情報

- ルート固有のアノテーション

5.2. ラベルとアノテーションのカスタマイズ

OpenShift Container Platform ルートは、Knative サービスの **metadata** 仕様を変更して設定できるカスタムラベルおよびアノテーションの使用をサポートします。カスタムラベルおよびアノテーションはサービスから Knative ルートに伝番され、次に Knative ingress に、最後に OpenShift Container Platform ルートに伝播されます。

5.2.1. OpenShift Container Platform ルートのラベルおよびアノテーションのカスタマイズ

前提条件

- OpenShift Serverless Operator および Knative Serving が OpenShift Container Platform クラスターにインストールされている必要があります。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. OpenShift Container Platform ルートに伝播するラベルまたはアノテーションが含まれる Knative サービスを作成します。
   - YAML を使用してサービスを作成するには、以下を実行します。

     YAML を使用して作成されるサービスの例

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

```
  annotations:
    <annotation_name>: <annotation_value>
...
```

- Knative (**kn**) CLI を使用してサービスを作成するには、次のように入力します。

  kn コマンドを使用して作成されるサービスの例

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

2. 以下のコマンドからの出力を検査して、OpenShift Container Platform ルートが追加したアノテーションまたはラベルで作成されていることを確認します。

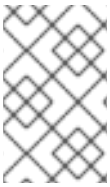
   検証のコマンドの例

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

1. サービスの名前を使用します。
2. サービスが作成された namespace を使用します。
3. ラベルおよびアノテーション名および値の値を使用します。

5.3. KNATIVE サービスのルートの設定

Knative サービスを OpenShift Container Platform で TLS 証明書を使用するように設定するには、OpenShift Serverless Operator によるサービスのルートの自動作成を無効にし、代わりにサービスのルートを手動で作成する必要があります。



注記

以下の手順を完了すると、**knative-serving-ingress** namespace のデフォルトの OpenShift Container Platform ルートは作成されません。ただし、アプリケーションの Knative ルートはこの namespace に引き続き作成されます。

5.3.1. OpenShift Container Platform ルートでの Knative サービスの設定

前提条件

- OpenShift Serverless Operator および Knative Serving コンポーネントが OpenShift Container Platform クラスターにインストールされている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **serving.knative.openshift.io/disableRoute=true** アノテーションが含まれる Knative サービスを作成します。

重要

serving.knative.openshift.io/disableRoute=true アノテーションは、OpenShift Serverless に対してルートを自動的に作成しないように指示します。ただし、サービスには URL が表示され、ステータスが **Ready** に達します。URL のホスト名と同じホスト名を使用して独自のルートを作成するまで、この URL は外部では機能しません。

   a. **Service** サービスリソースを作成します。

   リソースの例

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

   b. **Service** リソースを適用します。

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

   c. オプション:**kn service create** コマンドを使用して Knative サービスを作成します。

   kn コマンドの例

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

2. サービス用に OpenShift Container Platform ルートが作成されていないことを確認します。

   コマンドの例

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

   以下の出力が表示されるはずです。

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

3. **knative-serving-ingress** namespace で **Route** リソースを作成します。

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 600s (1)
  name: <route_name> (2)
  namespace: knative-serving-ingress (3)
spec:
  host: <service_host> (4)
  port:
    targetPort: http2
  to:
    kind: Service
    name: kourier
    weight: 100
  tls:
    insecureEdgeTerminationPolicy: Allow
    termination: edge (5)
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE----
  wildcardPolicy: None
```

(1) OpenShift Container Platform ルートのタイムアウト値。**max-revision-timeout-seconds** 設定と同じ値を設定する必要があります (デフォルトでは **600s**)。

(2) OpenShift Container Platform ルートの名前。

(3) OpenShift Container Platform ルートの namespace。これは **knative-serving-ingress** である必要があります。

(4) 外部アクセスのホスト名。これを **<service_name>-<service_namespace>.<domain>** に設定できます。

(5) 使用する証明書。現時点で、**edge** termination のみがサポートされています。

4. **Route** リソースを適用します。

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

5.4. グローバル HTTPS リダイレクト

HTTPS リダイレクトは、着信 HTTP リクエストのリダイレクトを提供します。これらのリダイレクトされた HTTP リクエストは暗号化されます。**KnativeServing** カスタムリソース (CR) の **httpProtocol** 仕様を設定して、クラスターのすべてのサービスに対して HTTPS リダイレクトを有効にできます。

5.4.1. HTTPS リダイレクトのグローバル設定

HTTPS リダイレクトを有効にする KnativeServing CR の例

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

5.5. 外部ルートの URL スキーム

セキュリティーを強化するために、外部ルートの URL スキームはデフォルトで HTTPS に設定されています。このスキームは、**KnativeServing** カスタムリソース (CR) 仕様の **default-external-scheme** キーによって決定されます。

5.5.1. 外部ルートの URL スキームの設定

デフォルト仕様

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

default-external-schemeキーを変更することにより、HTTP を使用するようにデフォルトの仕様をオーバーライドできます。

HTTP オーバーライド仕様

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

5.6. サービスごとの HTTPS リダイレクト

networking.knative.dev/http-option アノテーションを設定することにより、サービスの HTTPS リダイレクトを有効または無効にできます。

5.6.1. サービスの HTTPS のリダイレクト

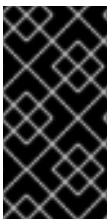
次の例は、Knative **Service** YAML オブジェクトでこのアノテーションを使用する方法を示しています。

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

5.7. クラスターローカルの可用性

デフォルトで、Knative サービスはパブリック IP アドレスに公開されます。パブリック IP アドレスに公開されているとは、Knative サービスがパブリックアプリケーションであり、一般にアクセス可能な URL があることを意味します。

一般にアクセス可能な URL は、クラスター外からアクセスできます。ただし、開発者は **プライベートサービス** と呼ばれるクラスター内からのみアクセス可能なバックエンドサービスをビルドする必要がある場合があります。開発者は、クラスター内の個々のサービスに **networking.knative.dev/visibility=cluster-local** ラベルを使用してラベル付けし、それらをプライベートにすることができます。



重要

OpenShift Serverless 1.15.0 以降のバージョンの場合には、**serving.knative.dev/visibility** ラベルは利用できなくなりました。既存のサービスを更新して、代わりに **networking.knative.dev/visibility** ラベルを使用する必要があります。

5.7.1. クラスターローカルへのクラスター可用性の設定

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 サービスを作成している。

手順

- **networking.knative.dev/visibility=cluster-local** ラベルを追加して、サービスの可視性を設定します。

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

検証

- 以下のコマンドを入力して出力を確認し、サービスの URL の形式が **http://<service_name>.<namespace>.svc.cluster.local** であることを確認します。

```
$ oc get ksvc
```

出力例

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

5.7.2. クラスターローカルサービスの TLS 認証の有効化

クラスターローカルサービスの場合、Kourier ローカルゲートウェイ **kourier-internal** が使用されます。Kourier ローカルゲートウェイに対して TLS トラフィックを使用する場合は、ローカルゲートウェイで独自のサーバー証明書を設定する必要があります。

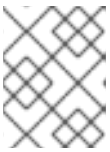
前提条件

- OpenShift Serverless Operator および Knative Serving がインストールされている。
- 管理者権限がある。
- OpenShift (**oc**) CLI がインストールされている。

手順

1. サーバー証明書を **knative-serving-ingress** namespace にデプロイします。

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



注記

これらの証明書が **<app_name>.<namespace>.svc.cluster.local** への要求を処理できるように、Subject Alternative Name (SAN) の検証が必要です。

2. ルートキーと証明書を生成します。

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

3. SAN 検証を使用するサーバーキーを生成します。

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

4. サーバー証明書を作成します。

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

5. Courier ローカルゲートウェイのシークレットを設定します。

   a. 前の手順で作成した証明書から、**knative-serving-ingress** namespace にシークレットをデプロイします。

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

   b. **KnativeServing** カスタムリソース (CR) 仕様を更新して、Kourier ゲートウェイによって作成されたシークレットを使用します。

   KnativeServing CR の例

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

Kourier コントローラーはサービスを再起動せずに証明書を設定するため、Pod を再起動する必要はありません。

クライアントから **ca.crt** をマウントして使用することにより、ポート **443** 経由で TLS を使用して Kourier 内部サービスにアクセスできます。

5.8. KOURIER GATEWAY サービスタイプ

Kourier Gateway は、デフォルトで **ClusterIP** サービスタイプとして公開されます。このサービスタイプは、**KnativeServing** カスタムリソース (CR) の **service-type** 入力仕様によって決定されます。

デフォルト仕様

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

5.8.1. Kourier Gateway サービスタイプの設定

service-type 仕様を変更することで、デフォルトのサービスタイプをオーバーライドして、代わりにロードバランサーサービスタイプを使用できます。

LoadBalancer オーバーライド仕様

```
...
```

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

5.9. HTTP2 と GRPC の使用

OpenShift Serverless はセキュアでないルートまたは edge termination ルートのみをサポートします。非セキュアなルートまたは edge termination ルートは OpenShift Container Platform で HTTP2 をサポートしません。gRPC は HTTP2 によって転送されるため、これらのルートは gRPC もサポートしません。アプリケーションでこれらのプロトコルを使用する場合は、Ingress ゲートウェイを使用してアプリケーションを直接呼び出す必要があります。これを実行するには、Ingress ゲートウェイのパブリックアドレスとアプリケーションの特定のホストを見つける必要があります。

5.9.1. HTTP2 および gRPC を使用したサーバーレスアプリケーションとの対話

重要

この方法は、OpenShift Container Platform 4.10 以降が対象です。古いバージョンについては、以下のセクションを参照してください。

前提条件

- OpenShift Serverless Operator と Knative Serving をクラスターにインストールしている。
- OpenShift CLI (**oc**) がインストールされている。
- Knative サービスを作成する。
- OpenShift Container Platform 4.10 以降をアップグレードする。
- OpenShift Ingress コントローラーで HTTP/2 を有効にする。

手順

1. **serverless.openshift.io/default-enable-http2=true** アノテーションを **KnativeServing** カスタムリソースに追加します。

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

2. アノテーションが追加されたら、Kourier サービスの **appProtocol** 値が **h2c** であることを確認できます。

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

出力例

```
h2c
```

3. 以下のように、外部トラフィックに HTTP/2 プロトコルで gRPC フレームワークを使用できるようになりました。

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

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

(1) **ksvc** URL。

(2) 証明書。

関連情報

- HTTP/2 Ingress 接続の有効化

5.9.2. OpenShift Container Platform 4.9 以前での HTTP2 および gRPC を使用したサーバーレスアプリケーションとの対話

重要

この方法は、**LoadBalancer** サービスタイプを使用して Kourier Gateway を公開する必要があります。これは、以下の YAML を **KnativeServing** カスタムリソース定義 (CRD) に追加して設定できます。

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

前提条件

- OpenShift Serverless Operator と Knative Serving をクラスターにインストールしている。
- OpenShift CLI (**oc**) がインストールされている。
- Knative サービスを作成する。

手順

1. アプリケーションホストを検索します。**サーバーレスアプリケーションのデプロイメントの確認**の説明を参照してください。
2. Ingress ゲートウェイのパブリックアドレスを見つけます。

```
$ oc -n knative-serving-ingress get svc kourier
```

出力例

```
NAME              TYPE        CLUSTER-IP     EXTERNAL-IP
PORT(S)
```

```
AGE
kourier   LoadBalancer   172.30.51.103   a83e86291bcdd11e993af02b7a65e514-
33544245.us-east-1.elb.amazonaws.com   80:31380/TCP,443:31390/TCP   67m
```

パブリックアドレスは **EXTERNAL-IP** フィールドで表示され、この場合は **a83e86291bcdd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com** になります。

3. HTTP 要求のホストヘッダーを手動でアプリケーションのホストに手動で設定しますが、Ingress ゲートウェイのパブリックアドレスに対して要求自体をダイレクトします。

```
$ curl -H "Host: hello-default.example.com" a83e86291bcdd11e993af02b7a65e514-
33544245.us-east-1.elb.amazonaws.com
```

出力例

```
Hello Serverless!
```

Ingress ゲートウェイに対して直接 gRPC 要求を行うこともできます。

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

grpc.Dial(
    "a83e86291bcdd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com:80",
    grpc.WithAuthority("hello-default.example.com:80"),
    grpc.WithInsecure(),
)
```

注記

直前の例のように、それぞれのポート (デフォルトでは 80) を両方のホストに追加します。

第6章 KNATIVE サービスへのアクセスの設定

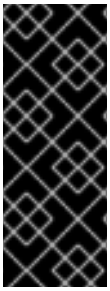
6.1. KNATIVE サービスの JSON WEB TOKEN 認証の設定

OpenShift Serverless には現在、ユーザー定義の承認機能がありません。ユーザー定義の承認をデプロイメントに追加するには、OpenShift Serverless を Red Hat OpenShift Service Mesh と統合してから、Knative サービスの JSON Web Token (JWT) 認証とサイドカーインジェクションを設定する必要があります。

6.2. SERVICE MESH 2.X での JSON WEB トークン認証の使用

Service Mesh 2.x と OpenShift Serverless を使用して、Knative サービスで JSON Web Token (JWT) 認証を使用できます。これを行うには、**ServiceMeshMemberRoll** オブジェクトのメンバーであるアプリケーション namespace に認証要求とポリシーを作成する必要があります。サービスのサイドカーインジェクションも有効にする必要があります。

6.2.1. Service Mesh 2.x および OpenShift Serverless の JSON Web トークン認証の設定



重要

knative-serving および **knative-serving-ingress** などのシステム namespace の Pod へのサイドカー挿入の追加は、Kourier が有効化されている場合はサポートされません。

OpenShift Container Platform では、これらの namespace の Pod にサイドカーの挿入が必要な場合は、**サービスメッシュと OpenShift Serverless のネイティブに統合**に関する OpenShift Serverless のドキュメントを参照してください。

前提条件

- OpenShift Serverless Operator、Knative Serving、および Red Hat OpenShift Service Mesh をクラスターにインストールしました。
- OpenShift CLI (**oc**) が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。

手順

1. **sidecar.istio.io/inject="true"** アノテーションをサービスに追加します。

   サービスの例

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: <service_name>
spec:
  template:
    metadata:
      annotations:
```

```
        sidecar.istio.io/inject: "true" 1
        sidecar.istio.io/rewriteAppHTTPProbers: "true" 2
...
```

1 **sidecar.istio.io/inject="true"** アノテーションを追加します。

2 OpenShift Serverless バージョン 1.14.0 以降では、HTTP プローブをデフォルトで Knative サービスの readiness プローブとして使用することから、Knative サービスでアノテーション **sidecar.istio.io/rewriteAppHTTPProbers: "true"** を設定する必要があります。

2. **Service** リソースを適用します。

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

3. **ServiceMeshMemberRoll** オブジェクトのメンバーである各サーバーレスアプリケーション namespace に **RequestAuthentication** リソースを作成します。

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

4. **RequestAuthentication** リソースを適用します。

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

5. 以下の **AuthorizationPolicy** リソースを作成して、**ServiceMeshMemberRoll** オブジェクトのメンバーである各サーバーレスアプリケーション namespace のシステム Pod からの **RequestAuthenticaton** リソースへのアクセスを許可します。

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

1 システム Pod でメトリクスを収集するためのアプリケーションのパス。

2 システム Pod でプローブするアプリケーションのパス。

6. **AuthorizationPolicy** リソースを適用します。

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

7. **ServiceMeshMemberRoll** オブジェクトのメンバーであるサーバーレスアプリケーション namespace ごとに、以下の **AuthorizationPolicy** リソースを作成します。

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

8. **AuthorizationPolicy** リソースを適用します。

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

検証

1. **curl** 要求を使用して Knative サービス URL を取得しようとすると、これは拒否されます。

   コマンドの例

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

   出力例

```
RBAC: access denied
```

2. 有効な JWT で要求を確認します。

   a. 有効な JWT トークンを取得します。

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

   b. **curl** 要求ヘッダーで有効なトークンを使用してサービスにアクセスします。

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

   これで要求が許可されます。

   出力例

```
Hello OpenShift!
```

6.3. SERVICE MESH 1.X での JSON WEB トークン認証の使用

Service Mesh 1.x と OpenShift Serverless を使用して、Knative サービスで JSON Web Token (JWT) 認証を使用できます。これを行うには、**ServiceMeshMemberRoll** オブジェクトのメンバーであるアプリケーション namespace にポリシーを作成する必要があります。サービスのサイドカーインジェクションも有効にする必要があります。

6.3.1. Service Mesh 1.x および OpenShift Serverless の JSON Web トークン認証の設定

重要

knative-serving および **knative-serving-ingress** などのシステム namespace の Pod へのサイドカー挿入の追加は、Kourier が有効化されている場合はサポートされません。

OpenShift Container Platform では、これらの namespace の Pod にサイドカーの挿入が必要な場合は、**サービスメッシュと OpenShift Serverless のネイティブに統合**に関する OpenShift Serverless のドキュメントを参照してください。

前提条件

- OpenShift Serverless Operator、Knative Serving、および Red Hat OpenShift Service Mesh をクラスターにインストールしました。
- OpenShift CLI (**oc**) が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。

手順

1. **sidecar.istio.io/inject="true"** アノテーションをサービスに追加します。

   サービスの例

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

1 **sidecar.istio.io/inject="true"** アノテーションを追加します。

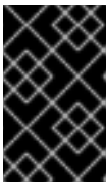
2

OpenShift Serverless バージョン 1.14.0 以降では、HTTP プローブをデフォルトで Knative サービスの readiness プローブとして使用することから、Knative サービスでアノテーショ

2. **Service** リソースを適用します。

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

3. 有効な JSON Web Tokens (JWT) の要求のみを許可する **ServiceMeshMemberRoll** オブジェクトのメンバーであるサーバーレスアプリケーション namespace でポリシーを作成します。



重要

パスの **/metrics** および **/healthz** は、**knative-serving** namespace のシステム Pod からアクセスされるため、**excludedPaths** に組み込まれる必要があります。

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

1. システム Pod でメトリクスを収集するためのアプリケーションのパス。
2. システム Pod でプローブするアプリケーションのパス。

4. **Policy** リソースを適用します。

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

検証

1. **curl** 要求を使用して Knative サービス URL を取得しようとすると、これは拒否されます。

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

出力例

```
Origin authentication failed.
```

2. 有効な JWT で要求を確認します。

   a. 有効な JWT トークンを取得します。

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

   b. **curl** 要求ヘッダーで有効なトークンを使用してサービスにアクセスします。

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

      これで要求が許可されます。

      出力例

      ```
      Hello OpenShift!
      ```

第7章 SERVING の KUBE-RBAC-PROXY の設定

kube-rbac-proxy コンポーネントは、Knative Serving の内部認証および認可機能を提供します。

7.1. SERVING の KUBE-RBAC-PROXY リソースの設定

OpenShift Serverless Operator CR を使用して、**kube-rbac-proxy** コンテナーのリソース割り当てをグローバルにオーバーライドできます。

> You can also override resource allocation for a specific deployment.

次の設定では、Knative Serving **kube-rbac-proxy** の最小および最大の CPU およびメモリー割り当てを設定します。

KnativeServing CR の例

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

1. 最小 CPU 割り当てを設定します。
2. 最小 RAM 割り当てを設定します。
3. 最大 CPU 割り当てを設定します。
4. 最大 RAM 割り当てを設定します。

第8章 NET-KOURIER のバーストと QPS の設定

1 秒あたりのクエリー数 (QPS) とバースト値によって、API サーバーへのリクエストまたは API 呼び出しの頻度が決まります。

8.1. NET-KOURIER のバースト値と QPS 値の設定

1 秒あたりのクエリー数 (QPS) の値によって、API サーバーに送信されるクライアントリクエストまたは API 呼び出しの数が決まります。

バースト値によって、処理のために保存できるクライアントからのリクエストの数が決まります。このバッファーを超えるリクエストはドロップされます。これは、バースト性が高く、要求が時間内に均一に分散されないコントローラーに役立ちます。

net-kourier-controller が再起動すると、クラスターにデプロイされたすべての **ingress** リソースが解析されるため、かなりの数の API 呼び出しが発生します。このため、**net-kourier-controller** の起動に時間がかかることがあります。

KnativeServing CR で **net-kourier-controller** の QPS およびバースト値を調整できます。

KnativeServing CR の例

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

1. コントローラーと API サーバー間の通信の QPS レート。デフォルト値は 200 です。
2. Kubelet と API サーバー間の通信のバースト容量。デフォルト値は 200 です。

第9章 KNATIVE サービスのカスタムドメインの設定

9.1. KNATIVE サービスのカスタムドメインの設定

Knative サービスには、クラスターの設定に基づいてデフォルトのドメイン名が自動的に割り当てられます。例: **<service_name>-<namespace>.example.com**.所有するカスタムドメイン名を Knative サービスにマッピングすることで、Knative サービスのドメインをカスタマイズできます。

これを行うには、サービスの **DomainMapping** リソースを作成します。複数の **DomainMapping** を作成して、複数のドメインおよびサブドメインを単一サービスにマップすることもできます。

9.2. カスタムドメインマッピング

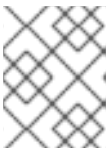
所有するカスタムドメイン名を Knative サービスにマッピングすることで、Knative サービスのドメインをカスタマイズできます。カスタムドメイン名をカスタムリソース (CR) にマッピングするには、Knative サービスまたは Knative ルートなどのアドレス指定可能なターゲット CR にマッピングする **DomainMapping** CR を作成する必要があります。

9.2.1. カスタムドメインマッピングの作成

所有するカスタムドメイン名を Knative サービスにマッピングすることで、Knative サービスのドメインをカスタマイズできます。カスタムドメイン名をカスタムリソース (CR) にマッピングするには、Knative サービスまたは Knative ルートなどのアドレス指定可能なターゲット CR にマッピングする **DomainMapping** CR を作成する必要があります。

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。
- OpenShift CLI (**oc**) が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。
- Knative サービスを作成し、そのサービスにマップするカスタムドメインを制御できる。



注記

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

手順

1. マップ先となるターゲット CR と同じ namespace に **DomainMapping** CR が含まれる YAML ファイルを作成します。

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: <domain_name> 1
 namespace: <namespace> 2
```

```
spec:
 ref:
   name: <target_name> 3
   kind: <target_type> 4
   apiVersion: serving.knative.dev/v1
```

1. ターゲット CR にマップするカスタムドメイン名。
2. **DomainMapping** CR とターゲット CR の両方の namespace。
3. カスタムドメインにマップするサービス名。
4. カスタムドメインにマップされる CR のタイプ。

サービスドメインマッピングの例

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: example.com
 namespace: default
spec:
 ref:
   name: showcase
   kind: Service
   apiVersion: serving.knative.dev/v1
```

ルートドメインマッピングの例

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: example.com
 namespace: default
spec:
 ref:
   name: example-route
   kind: Route
   apiVersion: serving.knative.dev/v1
```

2. **DomainMapping** CR を YAML ファイルとして適用します。

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

9.3. KNATIVE CLI を使用した KNATIVE サービスのカスタムドメイン

所有するカスタムドメイン名を Knative サービスにマッピングすることで、Knative サービスのドメインをカスタマイズできます。Knative (**kn**) CLI を使用して、Knative サービスまたは Knative ルートなどのアドレス指定可能なターゲット CR にマップする **DomainMapping** カスタムリソース (CR) を作成できます。

9.3.1. 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 showcase
```

--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は、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:showcase: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
```

9.4. 開発者パースペクティブを使用したドメインマッピング

所有するカスタムドメイン名を Knative サービスにマッピングすることで、Knative サービスのドメインをカスタマイズできます。OpenShift Container Platform Web コンソールの **Developer** パースペクティブを使用して、**DomainMapping** カスタムリソース (CR) を Knative サービスにマッピングできます。

9.4.1. 開発者パースペクティブを使用したカスタムドメインのサービスへのマッピング

前提条件

- Web コンソールにログインしている。
- **Developer** パースペクティブを使用している。
- 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。これはクラスター管理者が完了する必要があります。
- 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。
- Knative サービスを作成し、そのサービスにマップするカスタムドメインを制御できる。

注記

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

手順

1. **Topology** ページに移動します。
2. ドメインにマッピングするサービスを右クリックし、サービス名が含まれる **Edit** オプションを選択します。たとえば、サービスの名前が **showcase** の場合は、**Edit showcase** オプションを選択します。
3. **Advanced options** セクションで、**Show advanced Routing options** をクリックします。
   a. サービスにマッピングするドメインマッピング CR がすでに存在する場合は、**ドメインマッピング** リストで選択できます。
   b. 新規ドメインマッピング CR を作成する場合は、ドメイン名をボックスに入力し、**Create** オプションを選択します。たとえば、**example.com** と入力すると、**Create** オプションは **Create "example.com"** になります。
4. **Save** をクリックしてサービスへの変更を保存します。

検証

1. **Topology** ページに移動します。
2. 作成したサービスをクリックします。
3. サービス情報ウィンドウの **Resources** タブで、**Domain mappings** セクションにサービスにマッピングしたドメインが表示されます。

9.5. ADMINISTRATOR パースペクティブを使用したドメインマッピング

OpenShift Container Platform Web コンソールで **Developer** パースペクティブに切り替えたり、Knative (**kn**) CLI または YAML ファイルを使用しない場合は、OpenShift Container Platform Web コンソールの **Administator** パースペクティブを使用できます。

9.5.1. Administrator パースペクティブを使用したカスタムドメインのサービスへのマッピング

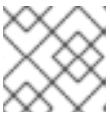
Knative サービスには、クラスターの設定に基づいてデフォルトのドメイン名が自動的に割り当てられます。例: **<service_name>-<namespace>.example.com**.所有するカスタムドメイン名を Knative サービスにマッピングすることで、Knative サービスのドメインをカスタマイズできます。

これを行うには、サービスの **DomainMapping** リソースを作成します。複数の **DomainMapping** を作成して、複数のドメインおよびサブドメインを単一サービスにマップすることもできます。

OpenShift Container Platform のクラスター管理者権限 (または OpenShift Dedicated または Red Hat OpenShift Service on AWS のクラスターまたは専用管理者権限) が割り当てられている場合は、Web コンソールの **管理者** パースペクティブを使用して **DomainMapping** カスタムリソース (CR) を作成できます。

前提条件

- Web コンソールにログインしている。
- **Administrator** パースペクティブに切り替えられている。
- OpenShift Serverless Operator がインストールされている。
- Knative Serving がインストールされています。
- 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が割り当てられたプロジェクトにアクセスできる。
- Knative サービスを作成し、そのサービスにマップするカスタムドメインを制御できる。



注記

カスタムドメインがクラスターの IP アドレスを参照する。

手順

1. **CustomResourceDefinitions** に移動し、検索ボックスを使用して **DomainMapping** カスタムリソース定義 (CRD) を検索します。
2. **DomainMapping** CRD をクリックしてから **Instances** タブに移動します。
3. **Create DomainMapping** をクリックします。
4. インスタンスの以下の情報が含まれるように **DomainMapping** CR の YAML を変更します。

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: <domain_name> 1
```

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

1 ターゲット CR にマップするカスタムドメイン名。

2 **DomainMapping** CR とターゲット CR の両方の namespace。

3 カスタムドメインにマップするサービス名。

4 カスタムドメインにマップされる CR のタイプ。

Knative サービスへのドメインマッピングの例

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: custom-ksvc-domain.example.com
 namespace: default
spec:
 ref:
   name: showcase
   kind: Service
   apiVersion: serving.knative.dev/v1
```

検証

- **curl** リクエストを使用してカスタムドメインにアクセスします。以下に例を示します。

  コマンドの例

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

  出力例

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

9.6. TLS 証明書を使用してマッピングされたサービスを保護する

9.6.1. TLS 証明書を使用してカスタムドメインでサービスを保護する

Knative サービスのカスタムドメインを設定したら、TLS 証明書を使用して、マップされたサービスを保護できます。これを行うには、Kubernetes TLS シークレットを作成してから、作成した TLS シークレットを使用するように **DomainMapping** CR を更新する必要があります。

注記

Ingress に **net-istio** を使用し、**security.dataPlane.mtls: true** を使用して SMCP 経由で mTLS を有効にする場合、Service Mesh は ***.local** ホストの **DestinationRules** をデプロイしますが、これは OpenShift Serverless の **DomainMapping** を許可しません。

この問題を回避するには、**security.dataPlane.mtls: true** を使用する代わりに **PeerAuthentication** をデプロイして mTLS を有効にします。

前提条件

- Knative サービスのカスタムドメインを設定し、有効な **DomainMapping** CR がある。
- 認証局プロバイダーからの TLS 証明書または自己署名証明書がある。
- 認証局プロバイダーまたは自己署名証明書から **cert** ファイルおよび **key** ファイルを取得している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. Kubernetes TLS シークレットを作成します。

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

2. **networking.internal.knative.dev/certificate-uid: <id>' ラベル** を Kubernetes TLS シークレットに追加します。

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

cert-manager などのサードパーティーのシークレットプロバイダーを使用している場合は、Kubernetes TLS シークレットに自動的にラベルを付けるようにシークレットマネージャーを設定できます。Cert-manager ユーザーは、提供されたシークレットテンプレートを使用して、正しいラベルを持つシークレットを自動的に生成できます。この場合、シークレットのフィルタリングはキーのみに基づいて行われますが、この値には、シークレットに含まれる証明書 ID などの有用な情報が含まれている可能性があります。

注記

Red Hat OpenShift の cert-manager Operator は、テクノロジープレビューの機能です。詳細は、**Red Hat OpenShift ドキュメントの cert-manager Operator のインストール** を参照してください。

3. 作成した TLS シークレットを使用するように **DomainMapping** CR を更新します。

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
  name: <domain_name>
  namespace: <namespace>
spec:
  ref:
```

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

検証

1. **DomainMapping** CR のステータスが **True** であることを確認し、出力の **URL** 列に、マップされたドメインをスキームの **https** で表示していることを確認します。

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

出力例

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

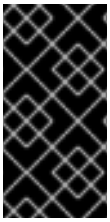
2. オプション: サービスが公開されている場合は、以下のコマンドを実行してこれが利用可能であることを確認します。

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

証明書が自己署名されている場合は、**curl** コマンドに **-k** フラグを追加して検証を省略します。

9.6.2. シークレットフィルタリングを使用した net-kourier のメモリー使用量の改善

デフォルトでは、Kubernetes **client-go** ライブラリーの informers の実装は、特定のタイプのすべてのリソースをフェッチします。これにより、多くのリソースが使用可能な場合にかなりのオーバーヘッドが発生する可能性があり、メモリーリークが原因で大規模なクラスターで Knative **net-kourier** Ingress コントローラーが失敗する可能性があります。ただし、Knative **net-kourier** Ingress コントローラーではフィルタリングメカニズムを使用できます。これにより、コントローラーは Knative 関連のシークレットのみを取得できます。このメカニズムを有効にするには、環境変数を **KnativeServing** カスタムリソース(CR)に設定します。



重要

シークレットフィルタリングを有効にする場合は、すべてのシークレットに **networking.internal.knative.dev/certificate-uid: "<id>"** というラベルを付ける必要があります。そうしないと、Knative Serving がシークレットを検出しないため、障害が発生します。新規および既存のシークレットの両方にラベルを付ける必要があります。

前提条件

- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。
- 自分で作成したプロジェクトまたは、アプリケーションや他のワークロードを作成するパーミッションおよびロールがあるプロジェクト。
- OpenShift Serverless Operator および Knative Serving をインストールしている。

- OpenShift CLI (**oc**) がインストールされている。

手順

- **KnativeServing** CR の **net-kourier-controller** に対して **ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID** 変数を **true** に設定します。

  KnativeServing CR の例

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
 name: knative-serving
 namespace: knative-serving
spec:
 deployments:
   - env:
     - container: controller
       envVars:
       - name: ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID
         value: 'true'
     name: net-kourier-controller
```

第10章 KNATIVE サービスの高可用性の設定

10.1. KNATIVE サービスの高可用性

高可用性 (HA) は Kubernetes API の標準的な機能で、中断が生じる場合に API が稼働を継続するのに役立ちます。HA デプロイメントでは、アクティブなコントローラーがクラッシュまたは削除されると、別のコントローラーをすぐに使用できます。このコントローラーは、現在使用できないコントローラーによって処理されていた API の処理を引き継ぎます。

OpenShift Serverless の HA は、リーダーの選択によって利用できます。これは、Knative Serving または Eventing コントロールプレーンのインストール後にデフォルトで有効になります。リーダー選択の HA パターンを使用する場合は、必要時に備えてコントローラーのインスタンスがスケジュールされ、クラスター内で実行されます。このコントローラーインスタンスは、リーダー選出ロックと呼ばれる共有リソースを使用するために競合します。リーダー選択ロックのリソースにアクセスできるコントローラーのインスタンスはリーダーと呼ばれます。

10.2. KNATIVE サービスの高可用性

高可用性 (HA) は、デフォルトで Knative Serving **activator**、**autoscaler**、**autoscaler-hpa**、**controller**、**webhook**、**kourier-control**、および **kourier-gateway** コンポーネントで使用できます。これらのコンポーネントは、デフォルトでそれぞれ 2 つのレプリカを持つように設定されています。**KnativeServing** カスタムリソース (CR) の **spec.high-availability.replicas** 値を変更して、これらのコンポーネントのレプリカ数を変更できます。

10.2.1. Knative Serving の高可用性レプリカの設定

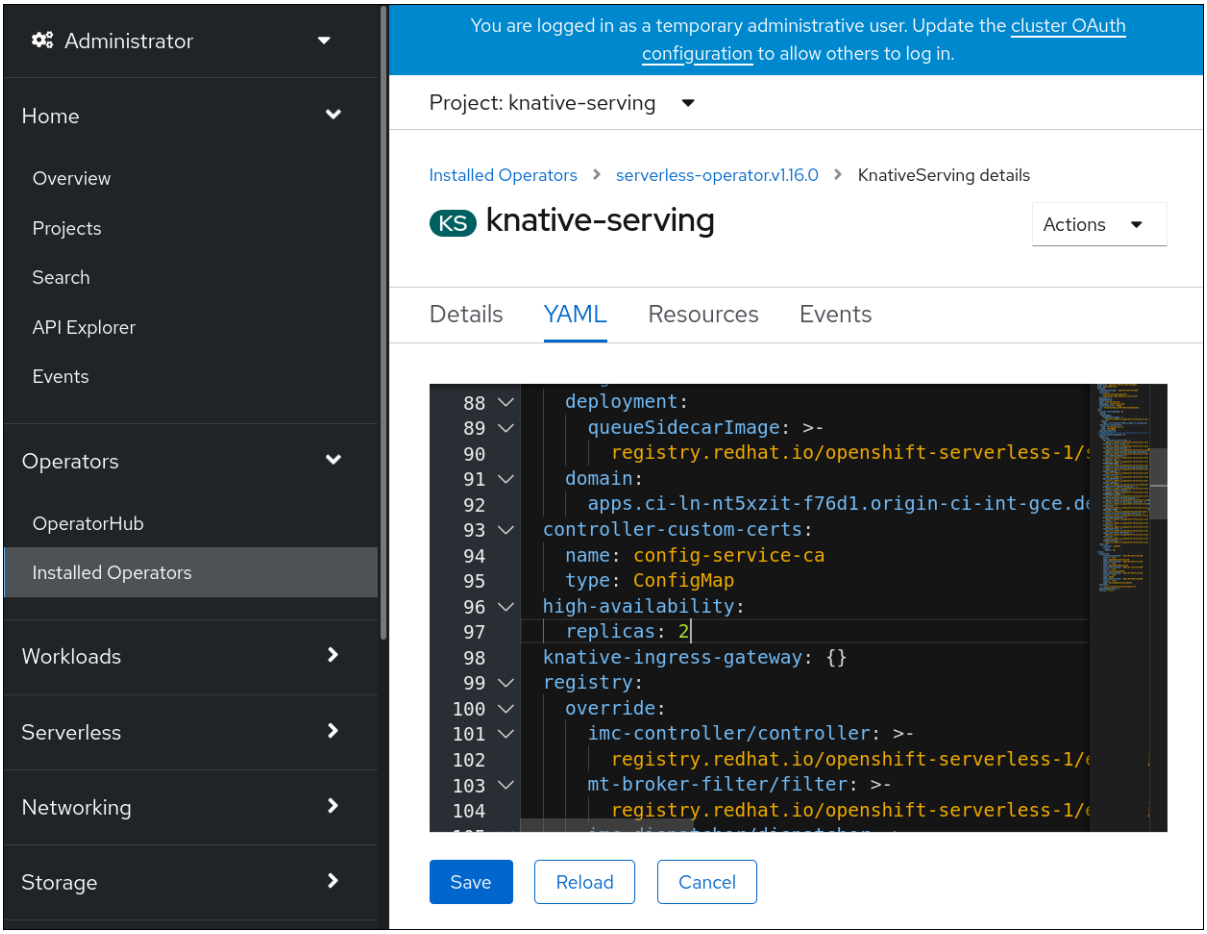
適格なデプロイメントリソースに最小 3 つのレプリカを指定するには、カスタムリソースのフィールド **spec.high-availability.replicas** の値を **3** に設定します。

前提条件

- OpenShift Container Platform に対するクラスター管理者権限があるか、Red Hat OpenShift Service on AWS または OpenShift Dedicated に対するクラスターまたは専用管理者権限がある。
- 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。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**OperatorHub → Installed Operators** に移動します。
2. **knative-serving** namespace を選択します。
3. OpenShift Serverless Operator の **Provided API** 一覧で **Knative Serving** をクリックし、**Knative Serving** タブに移動します。
4. **knative-serving** をクリックしてから、**knative-serving** ページの **YAML** タブに移動します。



5. **KnativeServing** CR のレプリカ数を変更します。

サンプル YAML

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