OpenShift Container Platform 4.1

Serverless アプリケーション

OpenShift Serverless のインストール、使用法およびリリースノート

Last Updated: 2020-04-16

OpenShift Container Platform 4.1 Serverless アプリケーション

OpenShift Serverless のインストール、使用法およびリリースノート

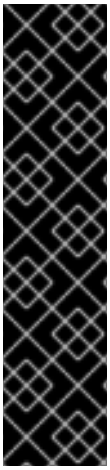
法律上の通知

概要

本書では、OCP 4.1 で OpenShift Serverless を使用する方法についての情報を提供します。

目次

第1章 OPENSHIFT SERVERLESS の使用開始



重要

OpenShift Serverless 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 の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。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。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

OpenShift Serverless は、開発者のインフラストラクチャーのセットアップまたはバックエンド開発に対する要件を軽減することにより、開発から実稼働までのコードの提供プロセスを単純化します。

1.1. OPENSHIFT SERVERLESS の仕組み

OpenShift Serverless 上の開発者は、使い慣れた言語およびフレームワークと共に、提供される Kubernetes ネイティブの API を使用してアプリケーションおよびコンテナーのワークロードをデプロイできます。OpenShift Serverless のインストールについての詳細は、「OpenShift Serverless のインストール」を参照してください。

OpenShift Container Platform 上の OpenShift Serverless を使用することにより、ステートフル、ステートレスおよびサーバーレスワークロードのすべてを、自動化された操作によって単一のマルチクラウドコンテナープラットフォームで実行することができます。開発者は、それぞれのマイクロサービス、レガシーおよびサーバーレスアプリケーションをホストするために単一プラットフォームを使用することができます。

OpenShift Serverless はオープンソースの Knative プロジェクトをベースとし、エンタープライズレベルのサーバーレスプラットフォームを有効にすることで、ハイブリッドおよびマルチクラウド環境における移植性と一貫性をもたらします。

1.2. OPENSHIFT SERVERLESS 上のアプリケーション

アプリケーションは、カスタムリソース定義 (CRD、Customer Resource Definition) および Kubernetes の関連コントローラーを使用して作成され、任意の場所で実行できる OCI 準拠の Linux コンテナーとしてパッケージ化されます。

OpenShift Serverless でアプリケーションをデプロイするには、Knative サービスを作成する必要があります。詳細は、「Knative サービスの使用開始」を参照してください。

第2章 OPENSHIFT SERVERLESS 製品アーキテクチャー

2.1. KNATIVE SERVING

OpenShift Container Platform 上の Knative Serving は、サーバーレスアプリケーションのデプロイおよび提供をサポートするために Kubernetes および Istio をベースにビルドされます。

これは、OpenShift Container Platform クラスター上のサーバーレスワークロードの動作を定義し、制御するために使用される Kubernetes カスタムリソース定義 (CRD) のセットを作成します。

これらの CRD は、サーバーレスコンテナーの迅速なデプロイメント、Pod の自動スケーリング、Istio コンポーネントのルーティングおよびネットワークプログラミング、デプロイされたコードおよび設定の特定の時点のスナップショットの表示などの複雑なユースケースに対応するビルディングブロックとして使用できます。

2.1.1. Knative Serving コンポーネント

このセクションで説明するコンポーネントは、Knative Serving が正しく設定され、実行されるために必要なリソースです。

Knative サービスリソース

service.serving.knative.dev リソースは、クラスター上のサーバーレスワークロードのライフサイクル全体を自動的に管理します。これは、サービスの各アップデートについてのルート、設定、および新規リビジョンがアプリケーションに設定されるように他のオブジェクトの作成を制御します。サービスは、トラフィックを最新バージョンまたは固定されたバージョンに常にルーティングするように定義できます。

Knative ルートリソース

route.serving.knative.dev リソースは、ネットワークのエンドポイントを、1 つ以上の Knative リビジョンにマップします。部分的なトラフィックや名前付きルートなどのトラフィックを複数の方法で管理することができます。

Knative 設定リソース

configuration.serving.knative.dev リソースは、デプロイメントの必要な状態を維持します。設定を変更すると、新規リビジョンが作成されます。

Knative リビジョンリソース

revision.serving.knative.dev リソースは、ワークロードに対して加えられるそれぞれの変更についてのコードおよび設定の特定の時点におけるスナップショットです。リビジョンはイミュータブル (変更不可) オブジェクトであり、必要な期間保持することができます。クラスター管理者は、**revision.serving.knative.dev** リソースを変更して、OpenShift Container Platform クラスターでの Pod の自動スケーリングを有効にできます。

2.2. KNATIVE CLIENT

Knative Client (**kn**) は、**oc** または **kubectl** ツールの機能を拡張し、OpenShift Container Platform での Knative コンポーネントとの対話を可能にします。**kn** は、開発者が YAML ファイルを直接編集しなくてもアプリケーションをデプロイし、管理できるようにします。

第3章 OPENSHIFT SERVERLESS のインストール

注記

OpenShift Serverless は、ネットワークが制限された環境でのインストールに対してテストされておらず、サポートされません。

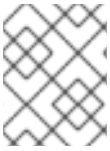
3.1. クラスターサイズの要件

OpenShift Serverless が正常に実行されるようにクラスターのサイズを適切に設定する必要があります。MachineSet API を使用して、クラスターを必要なサイズにスケールアップすることができます。

最初のサーバーレスアプリケーションを初めて使用する場合は、CPU が 10 つ、メモリーが 40 GB の OpenShift クラスターが最低要件となります。これは、通常 2 つのマシンを追加することによってデフォルト MachineSet のいずれかをスケールアップする必要があることを意味します。

注記

この設定では、要件はデプロイされたアプリケーションによって異なります。デフォルトで、各 Pod は CPU ~400m を要求し、推奨値のベースはこの値になります。特定の推奨例としては、アプリケーションは最大 10 のレプリカにスケールアップできます。アプリケーションの実際の CPU 要求を減らして、さらに境界を引き上げます。



注記

指定された数は、OpenShift クラスターのワーカーマシンのプールにのみ関連します。マスターノードは一般的なスケジューリングには使用されず、省略されます。

OpenShift Container Platform でのロギング、モニタリング、メータリングおよびトレーシングなどの高度なユースケースの場合は、追加のリソースをデプロイする必要があります。このようなユースケースで推奨される要件は 24 vCPU および 96GB メモリーです。

追加リソース

MachineSet API の使用についての詳細は、「MachineSet の作成」を参照してください。

3.1.1. MachineSet の手動によるスケーリング

MachineSet のマシンのインスタンスを追加したり、削除したりする必要がある場合、MachineSet を手動でスケーリングできます。

前提条件

- OpenShift Container Platform クラスターおよび **oc** コマンドラインをインストールします。
- **cluster-admin** パーミッションを持つユーザーとして、**oc** にログインします。

手順

1. クラスターにある MachineSet を表示します。

```
$ oc get machinesets -n openshift-machine-api
```

MachineSet は **<clusterid>-worker-<aws-region-az>** の形式で一覧表示されます。

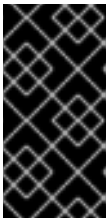
2. MachineSet をスケーリングします。

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

または、以下を実行します。

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

MachineSet をスケールアップまたはスケールダウンできます。新規マシンが利用可能になるまで数分の時間がかかります。

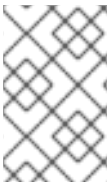


重要

デフォルトで、OpenShift Container Platform ルーター Pod はワーカーにデプロイされます。ルーターは Web コンソールなどの一部のクラスターリソースにアクセスすることが必要であるため、 ルーター Pod をまず再配置しない限り、ワーカー MachineSet を **0** にスケーリングできません。

3.2. SERVICE MESH のインストール

OpenShift Serverless のインストールには、サービスメッシュのインストールされたバージョンが必要です。詳細は、OpenShift Container Platform ドキュメントで「Installing Service Mesh」を参照してください。

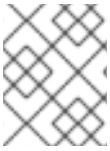


注記

Operator のインストールについてのみサービスメッシュのドキュメントを使用してください。Operator のインストール後は、以下のドキュメントを使用してサービスメッシュのコントロールプレーンおよびメンバーロールをインストールします。

3.2.1. ServiceMeshControlPlane のインストール

サービスメッシュは、データプレーンおよびコントロールプレーンで構成されます。ServiceMesh Operator のインストール後に、コントロールプレーンをインストールできます。コントロールプレーンは、サイドプロキシーを管理し、ポリシーを適用し、Telemetry を収集するようにこれを設定します。以下の手順では、アプリケーションに対する ingress として機能するサービスメッシュのコントロールプレーンのバージョンをインストールします。



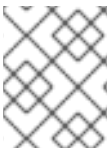
注記

コントロールプレーンを **istio-system** namespace にインストールする必要があります。現在、他の namespace はサポートされていません。

サンプル YAML ファイル

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
metadata:
  name: basic-install
  namespace: istio-system
spec:
```

```
istio:
  global:
    multitenant: true
    proxy:
      autoInject: disabled
    omitSidecarInjectorConfigMap: true
    disablePolicyChecks: false
    defaultPodDisruptionBudget:
      enabled: false
  istio_cni:
    enabled: true
  gateways:
    istio-ingressgateway:
      autoscaleEnabled: false
      type: LoadBalancer
    istio-egressgateway:
      enabled: false
    cluster-local-gateway:
      autoscaleEnabled: false
      enabled: true
      labels:
        app: cluster-local-gateway
        istio: cluster-local-gateway
      ports:
        - name: status-port
          port: 15020
        - name: http2
          port: 80
          targetPort: 8080
        - name: https
          port: 443
  mixer:
    enabled: false
    policy:
      enabled: false
    telemetry:
      enabled: false
  pilot:
    autoscaleEnabled: false
    sidecar: false
  kiali:
    enabled: false
  tracing:
    enabled: false
  prometheus:
    enabled: false
  grafana:
    enabled: false
  sidecarInjectorWebhook:
    enabled: false
```



注記

このバージョンでは自動スケーリングが無効になっています。本リリースは実稼働環境での使用を目的としていません。

注記

OpenShift Serverless で有効にされているサイドカーコンテナー挿入を使用してサービスメッシュを実行することは現時点で推奨されません。

前提条件

- クラスター管理者アクセスのあるアカウント
- ServiceMesh Operator がインストールされている。

手順

1. クラスター管理者として OpenShift Container Platform インストールにログインします。
2. 以下のコマンドを実行して **istio-system** namespace を作成します。

```
$ oc new-project istio-system
```

3. サンプル YAML ファイルを **smcp.yaml** ファイルにコピーします。
4. 以下のコマンドを使用して YAML ファイルを適用します。

```
$ oc apply -f smcp.yaml
```

5. 以下のコマンドを実行して、インストールプロセス時に Pod の進捗を確認します。

```
$ oc get pods -n istio-system -w
```

3.2.2. ServiceMeshMemberRoll のインストール

サービスメッシュがマルチテナンシー用に設定されている場合、コントロールプレーン namespace のサービスメッシュのメンバーロールが必要です。アプリケーションがデプロイされたコントロールプレーンおよび ingress を使用するには、それらの namespace をメンバーロールの一部にする必要があります。

マルチテナントコントロールプレーンのインストールは、サービスメッシュの一部として設定された namespace のみに影響を与えます。**ServiceMeshControlPlane** リソースと同じ namespace にある **ServiceMeshMemberRoll** リソースのサービスメッシュに関連付けられた namespace を指定し、これを **default** として指定する必要があります。

ServiceMeshMemberRoll カスタムリソースの例

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system
spec:
  members:
  - knative-serving
  - mynamespace
```

前提条件

- Service Mesh Operator がインストールされていること。
- Red Hat OpenShift Service Mesh コントロールプレーンのパラメーターを定義するカスタムリソースファイル。

手順

1. ServiceMeshMemberRoll カスタムリソースサンプルを複製する YAML ファイルを作成します。
2. 関連する namespace を含めるように YAML ファイルを設定します。

注記

サーバーレスアプリケーションをデプロイするすべての namespace を追加します。**knative-serving** namespace をメンバーロールに保持するようにしてください。

3. 以下を使用して、設定した YAML をファイル **smmr.yaml** にコピーし、これを適用します。

```
$ oc apply -f smmr.yaml
```

3.3. OPENSHIFT SERVERLESS OPERATOR のインストール

OpenShift Serverless Operator は、OpenShift Container Platform の Operator のインストール手順に従ってインストールできます。

OpenShift Container Platform の Operator のインストール手順に従い、OpenShift Serverless Operator をホストクラスターにインストールできます。

注記

OpenShift Serverless Operator は、OpenShift Container Platform バージョン 4.1.13 以降でのみ動作します。

詳細は、OpenShift Container Platform ドキュメントの「Operator のクラスターへの追加」を参照してください。

3.4. KNATIVE SERVING のインストール

OpenShift Serverless Operator を使用して Knative Serving をインストールするには、**KnativeServing** オブジェクトを作成する必要があります。

重要

サンプル YAML に示されるように、**KnativeServing** オブジェクトを **knative-serving** namespace に作成する必要があります。そうでない場合、これは無視されます。

サンプル serving.yaml

```
apiVersion: v1
kind: Namespace
metadata:
 name: knative-serving
---
apiVersion: serving.knative.dev/v1alpha1
kind: KnativeServing
metadata:
 name: knative-serving
 namespace: knative-serving
```

前提条件

- クラスター管理者アクセスのあるアカウント
- OpenShift Serverless Operator がインストールされていること。

手順

1. 以下を使用して、サンプル YAML ファイルを **serving.yaml** にコピーし、これを適用します。

```
$ oc apply -f serving.yaml
```

2. 以下のコマンドを使用して、インストールが完了したことを確認します。

```
$ oc get knativeserving/knative-serving -n knative-serving --template='{{range
.status.conditions}}{{printf "%s=%s\n" .type .status}}{{end}}'
```

結果は以下のようになります。

```
DeploymentsAvailable=True
InstallSucceeded=True
Ready=True
```

3.5. KNATIVE SERVING のアンインストール

Knative Serving をアンインストールするには、そのカスタムリソースを削除してから **knative-serving** namespace を削除する必要があります。

前提条件

- Knative Serving がインストールされていること。

手順

1. Knative Serving を削除するには、以下のコマンドを使用します。

```
$ oc delete knativeserving knative-serving -n knative-serving
```

2. コマンドが実行され、すべての Pod が **knative-serving** namespace から削除された後に、以下のコマンドを使用して namespace を削除します。

```
$ oc delete namespace knative-serving
```

3.6. OPENSHIFT SERVERLESS OPERATOR の削除

Operator の削除に関する OpenShift Container Platform の手順に従い、OpenShift Serverless Operator をホストクラスターから削除できます。

詳細は、OpenShift Container Platform ドキュメントの「クラスターからの Operator の削除」を参照してください。

3.7. OPERATOR からの KNATIVE SERVING CRD の削除

OpenShift Serverless Operator のアンインストール後に、Operator CRD および API サービスはクラスター上に残ります。この手順を使用して、残りのコンポーネントを完全にアンインストールします。

前提条件

- 先の手順を使用して、Knative Serving をアンインストールし、OpenShift Serverless Operator を削除していること。

手順

1. 以下のコマンドを実行して残りの Knative Serving CRD を削除します。

```
$ oc delete crd knativeservings.serving.knative.dev
```

第4章 KNATIVE サービスの使用開始

Knative サービスは、ユーザーがサーバーレスアプリケーションをデプロイするために作成する Kubernetes サービスです。各 Knative サービスは、ルートおよび **.yaml** ファイルに含まれる設定によって定義されます。

4.1. KNATIVE サービスの作成

サービスを作成するには、**service.yaml** ファイルを作成する必要があります。

以下のサンプルをコピーできます。このサンプルは **helloworld-go** というサンプルの golang アプリケーションを作成し、このサンプルを使用してそのアプリケーションのイメージを指定することができます。

```
apiVersion: serving.knative.dev/v1alpha1 (1)
kind: Service
metadata:
  name: helloworld-go (2)
  namespace: default (3)
spec:
  template:
    spec:
      containers:
        - image: gcr.io/knative-samples/helloworld-go (4)
          env:
            - name: TARGET (5)
              value: "Go Sample v1"
```

1. Knative の現行バージョン
2. アプリケーションの名前
3. アプリケーションが使用する namespace
4. アプリケーションのイメージへの URL
5. サンプルアプリケーションで出力される環境変数

4.2. サーバーレスアプリケーションのデプロイ

サーバーレスアプリケーションをデプロイするには、**service.yaml** ファイルを使用する必要があります。

手順

1. **service.yaml** ファイルが含まれているディレクトリーに移動します。
2. **service.yaml** ファイルを使用してアプリケーションをデプロイします。

```
$ oc apply --filename service.yaml
```

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ので、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新規に作成します。

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、アクティブでない Pod を含む 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。

第5章 OPENSHIFT SERVERLESS コンポーネントのモニタリング

OpenShift Container Platform クラスター管理者は、OpenShift Container Platform モニタリングスタックをデプロイし、OpenShift Serverless コンポーネントのメトリクスをモニターできます。

OpenShift Serverless Operator を使用する場合、デプロイされたコンポーネントをモニターするために必要な ServiceMonitor オブジェクトが自動的に作成されます。

Knative Serving などの OpenShift Serverless コンポーネントはメトリクスデータを公開します。管理者は、OpenShift Container Platform Web コンソールを使用してこのデータをモニターできます。

5.1. アプリケーションをモニターするためのクラスターの設定

アプリケーション開発者がアプリケーションをモニターできるようにするには、クラスターの人による操作によって、クラスターを設定する必要があります。以下の手順では、その方法を示します。

前提条件

- クラスターの管理権限を持つロールに属するユーザーとしてログインする必要があります。

手順

1. OpenShift Container Platform Web コンソールで、**Catalog → OperatorHub** に移動し、アプリケーションが置かれている namespace に Prometheus Operator をインストールします。

2. **Catalog → Developer Catalog** に移動し、同じ namespace に Prometheus、Alertmanager、Prometheus Rule、および Service Monitor をインストールします。

5.2. OPENSHIFT CONTAINER PLATFORM モニタリングの KNATIVE SERVING との使用についてのインストールの確認

管理者によるモニタリングの手動の設定は不要ですが、以下の手順を実行してモニタリングが正常にインストールしていることを確認できます。

手順

1. ServiceMonitor オブジェクトがデプロイされていることを確認します。

```
$ oc get servicemonitor -n knative-serving
NAME         AGE
activator    11m
autoscaler   11m
controller   11m
```

2. **openshift.io/cluster-monitoring=true** ラベルが Knative Serving namespace に追加されていることを確認します。

```
$ oc get namespace knative-serving --show-labels
NAME              STATUS   AGE   LABELS
knative-serving   Active   4d    istio-injection=enabled,openshift.io/cluster-
monitoring=true,serving.knative.dev/release=v0.7.0
```

5.3. OPENSHIFT CONTAINER PLATFORM モニタリングスタックの使用による KNATIVE SERVING のモニタリング

このセクションでは、OpenShift Container Platform モニタリングツールの使用により Knative Serving Pod の自動スケーリングメトリクスを視覚化する方法例を示します。

前提条件

- OpenShift Container Platform モニタリングスタックがインストールされていること。

手順

1. OpenShift Container Platform Web コンソールに移動し、認証します。
2. **Monitoring → Metrics** に移動します。
3. **Expression** に入力し、**Run queries** を選択します。Knative Serving Autoscaler Pod をモニターするには、以下のサンプル式を使用します。

   ```
   autoscaler_actual_pods
   ```

   これで、Knative Serving Autoscaler Pod のモニタリング情報をコンソールで表示できます。

第6章 OPENSHIFT SERVERLESS を使用したクラスターロギング

6.1. クラスターロギングについて

OpenShift Container Platform クラスター管理者は、いくつかの CLI コマンドを使用してクラスターロギングをデプロイでき、OpenShift Container Platform Web コンソールを使用して Elasticsearch Operator および Cluster Logging Operator をインストールできます。Operator がインストールされている場合、クラスターロギングのカスタムリソース (CR) を作成してクラスターロギング Pod およびクラスターロギングのサポートに必要な他のリソースをスケジュールします。Operator はクラスターロギングのデプロイ、アップグレード、および維持を行います。

クラスターロギングは、**instance** という名前のクラスターロギングのカスタムリソース (CR) を変更することで設定できます。CR は、ログを収集し、保存し、視覚化するために必要なロギングスタックのすべてのコンポーネントを含む完全なクラスターロギングデプロイメントを定義します。Cluster Logging Operator は **ClusterLogging** カスタムリソースを監視し、ロギングデプロイメントを適宜調整します。

管理者およびアプリケーション開発者は、表示アクセスのあるプロジェクトのログを表示できます。

6.2. クラスターロギングのデプロイおよび設定について

OpenShift Container Platform クラスターロギングは、小規模および中規模の OpenShift Container Platform クラスター用に調整されたデフォルト設定で使用されるように設計されています。

以下のインストール方法には、サンプルのクラスターロギングのカスタムリソース (CR) が含まれます。これを使用して、クラスターロギングインスタンスを作成し、クラスターロギングのデプロイメントを設定することができます。

デフォルトのクラスターロギングインストールを使用する必要がある場合は、サンプル CR を直接使用できます。

デプロイメントをカスタマイズする必要がある場合、必要に応じてサンプル CR に変更を加えます。以下では、クラスターロギングのインスタンスをインストール時に実行し、インストール後に変更する設定について説明します。クラスターロギングのカスタムリソース外で加える変更を含む、各コンポーネントの使用方法については、設定についてのセクションを参照してください。

6.2.1. クラスターロギングの設定およびチューニング

クラスターロギング環境は、**openshift-logging** プロジェクトにデプロイされるクラスターロギングのカスタムリソースを変更することによって設定できます。

インストール時またはインストール後に、以下のコンポーネントのいずれかを変更することができます。

メモリーおよび CPU

resources ブロックを有効なメモリーおよび CPU 値で変更することにより、各コンポーネントの CPU およびメモリーの両方の制限を調整することができます。

```
spec:
  logStore:
    elasticsearch:
      resources:
        limits:
          cpu:
```

```
        memory:
      requests:
        cpu: 1
        memory: 16Gi
    type: "elasticsearch"
  collection:
    logs:
      fluentd:
        resources:
          limits:
            cpu:
            memory:
          requests:
            cpu:
            memory:
        type: "fluentd"
  visualization:
    kibana:
      resources:
        limits:
          cpu:
          memory:
        requests:
          cpu:
          memory:
      type: kibana
  curation:
    curator:
      resources:
        limits:
          memory: 200Mi
        requests:
          cpu: 200m
          memory: 200Mi
      type: "curator"
```

Elasticsearch ストレージ

storageClass name および **size** パラメーターを使用し、Elasticsearch クラスターの永続ストレージのクラスおよびサイズを設定できます。Cluster Logging Operator は、これらのパラメーターに基づいて、Elasticsearch クラスターの各データノードについて **PersistentVolumeClaim** を作成します。

```
  spec:
    logStore:
      type: "elasticsearch"
      elasticsearch:
        storage:
          storageClassName: "gp2"
          size: "200G"
```

この例では、クラスターの各データノードが「gp2」ストレージの「200G」を要求する **PersistentVolumeClaim** にバインドされるように指定します。それぞれのプライマリーシャードは単一のレプリカによってサポートされます。

注記

storage ブロックを省略すると、一時ストレージのみを含むデプロイメントになります。

```
spec:
  logStore:
    type: "elasticsearch"
    elasticsearch:
      storage: {}
```

Elasticsearch レプリケーションポリシー

Elasticsearch シャードをクラスター内のデータノードにレプリケートする方法を定義するポリシーを設定できます。

- **FullRedundancy**:各インデックスのシャードはすべてのデータノードに完全にレプリケートされます。
- **MultipleRedundancy**:各インデックスのシャードはデータノードの半分に分散します。
- **SingleRedundancy**:各シャードの単一コピー。2 つ以上のデータノードが存在する限り、ログは常に利用可能かつ回復可能です。
- **ZeroRedundancy**:シャードのコピーはありません。ログは、ノードの停止または失敗時に利用不可になる (または失われる) 可能性があります。

Curator スケジュール

Curator のスケジュールを [cron format] で指定できます (https://en.wikipedia.org/wiki/Cron)。

```
spec:
  curation:
  type: "curator"
  resources:
  curator:
    schedule: "30 3 * * *"
```

6.2.2. 変更されたクラスターロギングカスタムリソースのサンプル

以下は、前述のオプションを使用して変更されたクラスターロギングのカスタムリソースの例です。

変更されたクラスターロギングカスタムリソースのサンプル

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"
  namespace: "openshift-logging"
spec:
  managementState: "Managed"
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 2
```

```
    resources:
      limits:
        memory: 2Gi
      requests:
        cpu: 200m
        memory: 2Gi
    storage: {}
    redundancyPolicy: "SingleRedundancy"
visualization:
  type: "kibana"
  kibana:
    resources:
      limits:
        memory: 1Gi
      requests:
        cpu: 500m
        memory: 1Gi
    replicas: 1
curation:
  type: "curator"
  curator:
    resources:
      limits:
        memory: 200Mi
      requests:
        cpu: 200m
        memory: 200Mi
    schedule: "*/5 * * * *"
collection:
  logs:
    type: "fluentd"
    fluentd:
      resources:
        limits:
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 1Gi
```

6.3. クラスターロギングの使用による KNATIVE SERVING コンポーネントのログの検索

手順

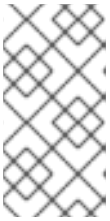
1. Elasticsearch の仮想化ツール Kibana UI を開くには、 以下のコマンドを使用して Kibana ルートを取得します。

```
$ oc -n openshift-logging get route kibana
```

2. ルートの URL を使用して Kibana ダッシュボードに移動し、ログインします。

3. インデックスが **.all** に設定されていることを確認します。インデックスが **.all** に設定されていない場合、OpenShift システムログのみが一覧表示されます。

4. **knative-serving** namespace を使用してログをフィルターできます。**kubernetes.namespace_name:knative-serving** を検索ボックスに入力して結果をフィルターします。



注記

Knative Serving はデフォルトで構造化ロギングを使用します。クラスターロギング Fluentd 設定をカスタマイズしてこれらのログの解析を有効にできます。これにより、ログの検索がより容易になり、ログレベルでのフィルターにより問題を迅速に特定できるようになります。

6.4. クラスターロギングを使用した KNATIVE SERVING でデプロイされたサービスのログの検索

OpenShift クラスターロギングにより、アプリケーションがコンソールに書き込むログは Elasticsearch で収集されます。以下の手順で、Knative Serving を使用してデプロイされたアプリケーションにこれらの機能を適用する方法の概要を示します。

手順

1. 以下のコマンドを使用して、Kibana の URL を見つけます。

```
$ oc -n cluster-logging get route kibana`
```

2. URL をブラウザーに入力し、Kibana UI を開きます。

3. インデックスが **.all** に設定されていることを確認します。インデックスが **.all** に設定されていない場合、OpenShift システムログのみが一覧表示されます。

4. サービスがデプロイされている Kubernetes namespace を使用してログをフィルターします。フィルターを追加してサービス自体を特定します: **kubernetes.namespace_name:default AND kubernetes.labels.serving_knative_dev\/service:{SERVICE_NAME}**

注記

/configuration または **/revision** を使用してフィルターすることもできます。

5. **kubernetes.container_name:<user-container>** を使用して検索を絞り込み、ご使用のアプリケーションで生成されるログのみを表示することができます。それ以外の場合は、queue-proxy からのログが表示されます。

注記

アプリケーションで JSON ベースの構造化ロギングを使用することで、実稼働環境でのこれらのログの迅速なフィルターを実行できます。

第7章 KNATIVE SERVING 自動スケーリングの設定

OpenShift Serverless は、Knative Serving 自動スケーリングシステムを OpenShift Container Platform クラスターで有効にすることで、アクティブでない Pod をゼロにスケーリングする機能など、Pod の自動スケーリングの各種機能を提供します。

Knative Serving の自動スケーリングを有効にするには、リビジョンテンプレートで同時実行 (concurrency) およびスケール境界 (scale bound) を設定する必要があります。

注記

リビジョンテンプレートでの制限およびターゲットの設定は、アプリケーションの単一インスタンスに対して行われます。たとえば、**target** アノテーションを **50** に設定することにより、アプリケーションの各インスタンスが一度に 50 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。

7.1. KNATIVE SERVING 自動スケーリングの同時要求の設定

アプリケーションの各インスタンス (リビジョンコンテナー) によって処理される同時要求の数は、リビジョンテンプレートに **target** アノテーションまたは **containerConcurrency** フィールドを追加して指定できます。

以下は、リビジョンテンプレートで使用される **target** のサンプルです。

```
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
  name: myapp
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/target: 50
    spec:
      containers:
      - image: myimage
```

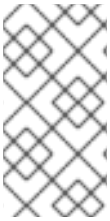
以下は、リビジョンテンプレートで使用される **containerConcurrency** のサンプルです。

```
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
  name: myapp
spec:
  template:
    metadata:
      annotations:
    spec:
      containerConcurrency: 100
      containers:
      - image: myimage
```

target と **containerConcurrency** の両方の値を追加することにより、同時要求の **target** 数をターゲットとして設定できますが、これにより要求の **containerConcurrency** 数のハード制限も設定されます。

たとえば、**target** 値が 50 で、 **containerConcurrency** 値が 100 の場合、要求のターゲットに設定された数は 50 になりますが、ハード制限は 100 になります。

containerConcurrency 値が **target** 値よりも低い場合、実際に処理できる数よりも多くの要求をターゲットとして設定する必要はないため、**target** 値は小さい値に調整されます。



注記

containerConcurrency は、特定の時点にアプリケーションに到達する要求の数を制限する明らかな必要がある場合にのみ使用する必要があります。**containerConcurrency** は、アプリケーションで同時実行の制約を実行する必要がある場合にのみ使用することを推奨します。

7.1.1. ターゲットアノテーションの使用による同時要求の設定

同時要求数のデフォルトターゲットは **100** ですが、リビジョンテンプレートで **autoscaling.knative.dev/target** アノテーション値を追加または変更することによってこの値を上書きできます。

以下は、ターゲットを **50** に設定するためにこのアノテーションをリビジョンテンプレートで使用する方法の例を示しています。

```
autoscaling.knative.dev/target: 50
```

7.1.2. containerConcurrency フィールドを使用した同時要求の設定

containerConcurrency は、処理される同時要求数にハード制限を設定します。

```
containerConcurrency: 0 | 1 | 2-N
```

0
: 無制限の同時要求を許可します。

1
: リビジョンコンテナーの所定インスタンスによって一度に処理される要求は 1 つのみであることを保証します。

2 以上
: 同時要求をこの数に制限します。

注記

target アノテーションがない場合、自動スケーリングは、**target** が **containerConcurrency** の値と等しい場合のように設定されます。

7.2. KNATIVE SERVING 自動スケーリングのスケール境界の設定

minScale および **maxScale** アノテーションは、アプリケーションを提供できる Pod の最小および最大数を設定するために使用できます。これらのアノテーションは、コールドスタートを防いだり、コンピューティングコストをコントロールするために使用できます。

minScale

minScale アノテーションが設定されていない場合、Pod はゼロ (または、enable-scale-to-zero が **ConfigMap** に基づいて false の場合は 1) にスケーリングします。

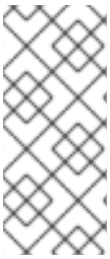
maxScale

maxScale アノテーションが設定されていない場合、作成される Pod の上限はありません。

minScale および **maxScale** は、リビジョンテンプレートで以下のように設定できます。

```
spec:
  template:
    metadata:
      autoscaling.knative.dev/minScale: "2"
      autoscaling.knative.dev/maxScale: "10"
```

これらのアノテーションをリビジョンテンプレートで使用することで、この設定が **PodAutoscaler** オブジェクトに伝播します。



注記

これらのアノテーションは、リビジョンの有効期間全体で適用されます。リビジョンがルートで参照されていない場合でも、 **minScale** によって指定される最小 Pod 数は依然として指定されます。ルーティングできないリビジョンについては、ガベージコレクションの対象になることに留意してください。これにより、Knative はリソースを回収できます。

第8章 KNATIVE CLIENT の使用

Knative Client (**kn**) は、Knative コマンドラインインターフェース (CLI) です。CLI は、アプリケーションを管理するためのコマンド、および OpenShift Container Platform の各種コンポーネントと対話する低レベルのツールを公開しています。**kn** を使用すると、ターミナルからアプリケーションを作成し、OpenShift Container Platform プロジェクトを管理できます。

Knative Client のメカニズムには独自のログは含まれません。クラスターにログインするには、**oc** CLI をインストールし、**oc** ログインを使用する必要があります。CLI のインストールオプションは、お使いのオペレーティングシステムによって異なります。

8.1. OPENSHIFT コマンドラインインターフェースのインストール

oc として知られる OpenShift コマンドラインインターフェース (CLI) をダウンロードし、インストールすることができます。

重要

以前のバージョンの **oc** をインストールしている場合、これを使用して OpenShift Container Platform 4.1 のすべてのコマンドを実行することはできません。新規バージョンの **oc** をダウンロードし、インストールする必要があります。

手順

1. Red Hat OpenShift Cluster Manager サイトの Infrastructure Provider ページから、選択するインストールタイプのページに移動し、**Download Command-line Tools** をクリックします。
2. 表示されているサイトから、オペレーティングシステムの圧縮されたファイルをダウンロードします。

注記

oc は Linux、Windows、または macOS にインストールできます。

3. 圧縮ファイルを展開して、指定のパスにあるディレクトリーに配置します。

8.1.1. Linux の kn CLI のインストール

Linux ディストリビューションの場合、CLI を **tar.gz** アーカイブとして直接ダウンロードできます。

手順

1. CLI をダウンロードします。
2. アーカイブを展開します。

```
$ tar -xf <file>
```

3. **kn** バイナリーをパスにあるディレクトリーに移動します。
4. PATH を確認するには、以下を実行します。

```
$ echo $PATH
```

注記

RHEL または Fedora を使用しない場合は、**libc** がライブラリーパスのディレクトリーにインストールされていることを確認してください。**libc** が利用できない場合は、CLI コマンドの実行時に以下のエラーが表示される場合があります。

```
$ kn: No such file or directory
```

8.1.2. RPM を使用した Linux 用の kn CLI のインストール

Red Hat Enterprise Linux (RHEL) の場合、Red Hat アカウントに有効な OpenShift Container Platform サブスクリプションがある場合は、**kn** を RPM としてインストールできます。

手順

- 以下のコマンドを使用して、**kn**をインストールします。

```
# subscription-manager register
# subscription-manager refresh
# subscription-manager attach --pool=<pool_id> 1
# subscription-manager repos --enable="openshift-serverless-1-for-rhel-8-x86_64-rpms"
# yum install openshift-serverless-clients
```

1 有効な OpenShift Container Platform サブスクリプションのプール ID

8.1.3. macOS の kn CLI のインストール

macOS の**kn**は、**tar.gz**アーカイブとして提供されます。

手順

1. CLI をダウンロードします。
2. アーカイブを展開および解凍します。
3. **kn** バイナリーをパスにあるディレクトリーに移動します。
4. パスを確認するには、ターミナルウィンドウを開き、以下を実行します。

```
$ echo $PATH
```

8.1.4. Windows の kn CLI のインストール

Windows の CLI は zip アーカイブとして提供されます。

手順

1. CLI をダウンロードします。
2. ZIP プログラムでアーカイブを解凍します。
3. **kn** バイナリーをパスにあるディレクトリーに移動します。

4. パスを確認するには、コマンドプロンプトを開いて以下のコマンドを実行します。

```
C:\> path
```

8.2. CLI へのログイン

oc CLI にログインしてクラスターにアクセスし、これを管理できます。

前提条件

- OpenShift Container Platform クラスターへのアクセスが必要になります。
- CLI をインストールしていること。

手順

- **oc login** コマンドを使用して CLI にログインし、プロンプトが出されたら必要な情報を入力します。

```
$ oc login
Server [https://localhost:8443]: https://openshift.example.com:6443 (1)
The server uses a certificate signed by an unknown authority.
You can bypass the certificate check, but any data you send to the server could be
intercepted by others.
Use insecure connections? (y/n): y (2)

Authentication required for https://openshift.example.com:6443 (openshift)
Username: user1 (3)
Password: (4)
Login successful.

You don't have any projects. You can try to create a new project, by running

    oc new-project <projectname>

Welcome! See 'oc help' to get started.
```

1. OpenShift Container Platform サーバー URL を入力します。
2. 非セキュアな接続を使用するかどうかを入力します。
3. ログインに使用するユーザー名を入力します。
4. ユーザーのパスワードを入力します。

これで、プロジェクトを作成でき、クラスターを管理するための他のコマンドを実行することができます。

8.3. KNATIVE CLIENT を使用した基本的なワークフロー

この基本的なワークフローを使用して、サービス上で作成 (create)、読み取り (read)、更新 (update)、削除 (delete) (CRUD) 操作を行います。以下の例では、環境変数 **TARGET** を読み取る単純な Hello World サービスをデプロイし、その出力を印刷します。

手順

1. イメージからサービスを **デフォルト** namespace に作成します。

```
$ kn service create hello --image gcr.io/knative-samples/helloworld-go --env
TARGET=Knative
Creating service 'hello' in namespace 'default':

  0.085s The Route is still working to reflect the latest desired specification.
  0.101s Configuration "hello" is waiting for a Revision to become ready.
 11.590s ...
 11.650s Ingress has not yet been reconciled.
 11.726s Ready to serve.

Service 'hello' created with latest revision 'hello-gsdks-1' and URL:
http://hello.default.apps-crc.testing
```

2. サービスを一覧表示します。

```
$ kn service list
NAME    URL                                   LATEST         AGE     CONDITIONS   READY
REASON
hello   http://hello.default.apps-crc.testing   hello-gsdks-1   8m35s   3 OK / 3     True
```

3. **curl** サービスエンドポイントコマンドを使用して、サービスが機能しているかどうかを確認します。

```
$ curl http://hello.default.apps-crc.testing

Hello Knative!
```

4. サービスを更新します。

```
$ kn service update hello --env TARGET=Kn
Updating Service 'hello' in namespace 'default':

 10.136s Traffic is not yet migrated to the latest revision.
 10.175s Ingress has not yet been reconciled.
 10.348s Ready to serve.

Service 'hello' updated with latest revision 'hello-dghll-2' and URL:
http://hello.default.apps-crc.testing
```

サービスの環境変数 **TARGET** が **Kn** に設定されます。

5. サービスを記述します。

```
$ kn service describe hello
Name:       hello
Namespace:  default
Age:        13m
URL:        http://hello.default.apps-crc.testing
Address:    http://hello.default.svc.cluster.local

Revisions:
```

```
  100%  @latest (hello-dghll-2) [2] (1m)
        Image:  gcr.io/knative-samples/helloworld-go (pinned to 5ea96b)

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

6. サービスを削除します。

```
$ kn service delete hello
Service 'hello' successfully deleted in namespace 'default'.
```

次に、**hello** サービスに対して **list** を試行することにより、このサービスが削除されていることを確認できます。

```
$ kn service list hello
No services found.
```

8.4. KNATIVE CLIENT を使用した自動スケーリングのワークフロー

YAML ファイルを直接編集せずに **kn** を使用して Knative サービスを変更することで、自動スケーリング機能にアクセスできます。

適切なフラグと共に **service create** および **service update** コマンドを使用して、自動スケーリング動作を設定します。

フラグ	説明
--concurrency-limit int	単一レプリカによって処理される同時要求のハード制限。
--concurrency-target int	受信する同時要求の数に基づくスケールアップのタイミングの推奨。デフォルトは **--concurrency-limit** に設定されます。
--max-scale int	レプリカの最大数。
--min-scale int	レプリカの最小数。

8.5. KNATIVE CLIENT を使用したトラフィック分割

kn は、Knative サービス上でルート指定されたトラフィックを取得するリビジョンを制御するのに役立ちます。

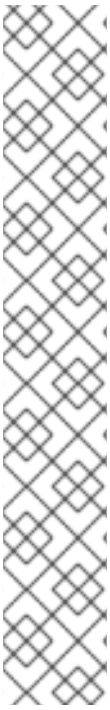
Knative サービスは、トラフィックのマッピングを許可します。これは、サービスのリビジョンのトラフィックの割り当てられた部分へのマッピングです。これは特定のリビジョンに固有の URL を作成するオプションを提供し、トラフィックを最新リビジョンに割り当てる機能を持ちます。

サービスの設定が更新されるたびに、サービスルートがすべてのトラフィックを準備状態にある最新リビジョンにポイントする状態で、新規リビジョンが作成されます。

この動作は、トラフィックの一部を取得するリビジョンを定義して変更することができます。

手順

- **kn service update** コマンドを **--traffic** フラグと共に使用して、トラフィックを更新します。



注記

--traffic RevisionName=Percent は以下の構文を使用します。

- **--traffic** フラグには、等号 (**=**) で区切られた 2 つの値が必要です。
- **RevisionName** 文字列はリビジョンの名前を参照します。
- **Percent** 整数はトラフィックのリビジョンに割り当てられた部分を示します。
- RevisionName の識別子 **@latest** を使用して、サービスの準備状態にある最新のリビジョンを参照します。この識別子は **--traffic** フラグと共に 1 回のみ使用できます。
- **service update** コマンドがトラフィックフラグと共にサービスの設定値を更新する場合、 **@latest** 参照は更新が適用される作成済みリビジョンをポイントします。
- **--traffic** フラグは複数回指定でき、すべてのフラグの **Percent** 値の合計が 100 になる場合にのみ有効です。

注記

たとえば、すべてのトラフィックを配置する前に 10% のトラフィックを新規リビジョンにルート指定するには、以下のコマンドを使用します。

```
$ kn service update svc --traffic @latest=10 --traffic svc-vwxyz=90
```

8.5.1. タグリビジョンの割り当て

サービスのトラフィックブロック内のタグは、参照されるリビジョンをポイントするカスタム URL を作成します。ユーザーは、**http(s)://TAG-SERVICE.DOMAIN** 形式を使用して、カスタム URL を作成するサービスの利用可能なリビジョンの固有タグを定義できます。

指定されたタグは、サービスのトラフィックブロックに固有のものである必要があります。**kn** は **kn service update** コマンドの一環として、サービスのリビジョンのカスタムタグの割り当ておよび割り当て解除に対応します。

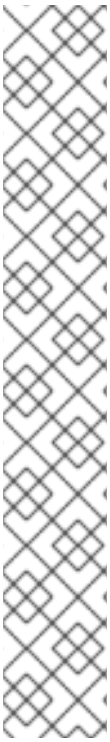
注記

タグを特定のリビジョンに割り当てた場合、ユーザーは、**--traffic** フラグ内で **--traffic Tag=Percent** として示されるタグでこのリビジョンを参照できます。

手順

- 以下のコマンドを使用します。

```
$ kn service update svc --tag @latest=candidate --tag svc-vwxyz=current
```



注記

--tag RevisionName=Tag は以下の構文を使用します。

- **--tag** フラグには、**=** で区切られる 2 つの値が必要です。
- **RevisionName** 文字列は **Revision** の名前を参照します。
- **Tag** 文字列は、このリビジョンに指定されるカスタムタグを示します。
- RevisionName の識別子 **@latest** を使用して、サービスの準備状態にある最新のリビジョンを参照します。この識別子は **--tag** フラグで 1 回のみ使用できます。
- **service update** コマンドがサービスの設定値を (タグフラグと共に) 更新している場合、**@latest** 参照は更新の適用後に作成されるリビジョンをポイントします。
- **--tag** フラグは複数回指定できます。
- **--tag** フラグは、同じリビジョンに複数の異なるタグを割り当てる場合があります。

8.5.2. タグリビジョンの割り当て解除

トラフィックブロックのリビジョンに割り当てられたタグは、割り当て解除できます。タグの割り当てを解除すると、カスタム URL が削除されます。

注記

リビジョンのタグが解除され、0% のトラフィックが割り当てられる場合、この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。

手順

- ユーザーは、**kn service update** コマンドを使用してリビジョンのタグの割り当てを解除できます。

```
$ kn service update svc --untag candidate
```

注記

--untag Tag は以下の構文を使用します。

- **--untag** フラグには 1 つの値が必要です。
- **tag** 文字列は、割り当てを解除する必要のあるサービスのトラフィックブロックの固有のタグを示します。これにより、それぞれのカスタム URL も削除されます。
- **--untag** フラグは複数回指定できます。

8.5.3. トラフィックフラグ操作の優先順位

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。

8.5.4. トラフィック分割フラグ

kn 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に対応します。

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。「繰り返し」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	オペレーション	繰り返し
--traffic	**RevisionName=Percent**	**Percent** トラフィックを **RevisionName** に指定します。	Yes
--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。	Yes
--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。	No
--tag	**RevisionName=Tag**	**Tag** を **RevisionName** に指定します。	Yes
--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。	No
--untag	tag	リビジョンから **Tag** を削除します。	Yes

第9章 OPENSHIFT SERVERLESS リリースノート

OpenShift Serverless 機能の概要については、「OpenShift Serverless の使用開始」を参照してください。

9.1. サポート

本書で説明されている手順に関連して問題が生じた場合には、カスタマーポータルにアクセスして、テクノロジープレビュー機能のサポートについて確認してください。

9.2. RED HAT OPENSHIFT SERVERLESS テクノロジープレビューリリースノート (1.2.0)

9.2.1. 新機能

- OpenShift Serverless は Knative Serving 0.9.0 を使用するように更新されました
- OpenShift Serverless は Knative Client (**kn** CLI) 0.9.0 を使用するように更新されました。
- OpenShift Container Platform 4.2 の OpenShift Serverless は、Operator Lifecycle Manager (OLM) の依存関係解決メカニズムを使用して ServiceMesh Operator を自動的にインストールするようになりました。必要な ServiceMeshControlPlane および ServiceMeshMemberRoll もユーザー向けにインストールされ、管理されます。
- KnativeServing リソースへのアクセスは、すべてのユーザーがリソースをブロックしないように **cluster-admin** ロールに制限されるようになりました。KnativeServing CR を作成できるのは **cluster-admin** ロールのみです。
- OpenShift Serverless Operator は、「knative」を検索して OperatorHub に表示されるようになりました。
- OpenShift Container Platform Web コンソールでは、KnativeServing リソースのステータス状況が表示されるようになりました。
- バージョン 1.2.0 では、OpenShift Serverless Operator は namespace のネットワークポリシーを検査します。
  ネットワークポリシーがない場合、Operator は幅広いオープンポリシーを自動的に作成し、トラフィックが namespace への/からの送信が行われ、OpenShift ルートを使用できるようにします。

  既存のネットワークポリシーがある場合、OpenShift Serverless は新規ポリシーを作成しません。Operator はユーザーがアプリケーションに必要な独自のネットワークポリシーを管理し続けることを予想します。たとえば、ユーザーは namespace への/からのトラフィックの送信を許可し、OpenShift ルートが namespace の ServiceMeshMemberRoll に追加された後もそのまま使用されることを許可するポリシーを設定する必要があります。

9.2.2. 修正された問題

- 以前のリリースでは、異なる namespace にある同じサービスまたはルートを使用すると、サービスが適切に機能せず、OpenShift Container Platform ルートが上書きされることがありました。この問題は修正されています。

- 以前のリリースでは、複数の異なるトラフィック分割ターゲットに必須のタグが必要でした。単一のトラフィック分割は **untagged** トラフィックターゲットで定義できるようになりました。

- OpenShift Serverless Operator バージョン 1.1.0 で作成され、公開されている既存の Knative サービスおよびルートは、cluster-local visibility に更新できませんでした。この問題は修正されています。

- **Unknown Uninitialized : Waiting for VirtualService** エラーが修正されています。

- Knative サービスは、クラスターが長時間実行される場合も 503 ステータスコードを返さなくなりました。

9.2.3. 既知の問題

- OLM を使用して OpenShift Serverless Operator を 4.2.4 よりも古い OpenShift Container Platform バージョンにインストールすると、コミュニティーバージョンの必要な依存関係を誤って使用する可能性があります。回避策として、4.2.4 よりも古いバージョンの OpenShift Container Platform は、OpenShift Serverless Operator をインストールする前に Red Hat が提供するバージョンの Elastic Search、Jaeger、Kiali、および ServiceMesh Operator を明示的にインストールします。

- OpenShift Serverless をバージョン 1.1.0 からバージョン 1.2.0 にアップグレードし、ServiceMeshControlPlane および ServiceMeshMemberRoll を Knative Serving インスタンスと共に機能するように設定している場合、**knative-serving** namespace および Knative サービスが含まれるその他の namespace を **istio-system** の ServiceMeshMemberRoll から削除する必要があります。
  また、ServiceMeshControlPlane が他のアプリケーションで必要がない場合には、これを namespace から完全に削除することもできます。

  アップグレードが開始されると、既存のサービスは以前と同じように機能しますが、新規サービスが準備状態になることはありません。**knative-serving** およびその他の関連する namespace を ServiceMeshMemberRoll から削除してリリースのブロックを解除すると、すべてのアクティブなサービスが短時間停止します。これは自己修正されます。Knative サービスを含むすべての namespace を元の ServiceMeshMemberRoll から削除することを確認します。

- gRPC および HTTP2 はルートに対して機能しません。これは OpenShift ルートの既知の制限です。

9.3. RED HAT OPENSHIFT SERVERLESS テクノロジープレビューリリースノート (1.1.0)

9.3.1. 新機能

- OpenShift Serverless が Knative Serving 0.8.1 を使用するように更新されました。

- 拡張された Operator メタデータには、サポート状態および公式のインストールドキュメントへのリンクに関する詳細情報が追加されました。

- Knative Eventing の開発者プレビューバージョンは OpenShift Serverless で使用できるようになりましたが、これは OpenShift Serverless Operator に含まれず、現時点ではこのテクノロジープレビューの一部としてサポートされていません。詳細は、「Knative Eventing on OpenShift Container Platform」を参照してください。

9.3.2. 修正された問題

- プロジェクト管理者ではないユーザーには、OpenShift Serverless を使用する際に以下のエラーが表示されていました。

  ```
  revisions.serving.knative.dev: User "sounds" cannot list resource "revisions
  ```

  この問題は、新しい RBAC ルールが追加されることにより修正されました。

- 競合状態により、Istio サイドカーの挿入が正常に機能しませんでした。Istio は、Pod の作成時に **knative-serving** namespace が ServiceMeshMemberRoll にあると見なしませんでした。Istio は ServiceMeshMemberRoll からのステータス情報を待機するようになり、これによりこの問題が修正されます。

9.3.3. 既知の問題

- ユーザーには、新たに作成された namespace のサービスが準備状態になるのを待機している間に **Unknown Uninitialized : Waiting for VirtualService to be ready** というエラーが表示される可能性があります。この待機時間は数分に及ぶ可能性があります。ユーザーが namespace の作成から namespace のサービスの作成までの間に十分な時間を確保する場合 (約 1 分)、このエラーは回避できる可能性があります。

- public visibility で作成された既存の Knative サービスおよびルートは cluster-local visibility に更新することができません。Knative サービスおよびルートで cluster-local visibility が必要な場合、これはこれらのリソースの作成時に設定される必要があります。

- Knative サービスは、クラスターが長時間実行される場合に 503 ステータスコードを返します。Knative Serving Pod にはエラーは表示されません。**istio-pilot** Pod を再起動すると、問題は一時的に解決されます。

- gRPC および HTTP2 はルートに対して機能しません。これは OpenShift ルートの既知の制限です。

9.4. RED HAT OPENSHIFT SERVERLESS テクノロジープレビューリリースノート (1.0.0)

9.4.1. 新機能

OpenShift Serverless の今回のリリースでは OpenShift Serverless Operator を導入しています。これは Knative Serving 0.7.1 に対応し、OpenShift Service Mesh 1.0 についてテスト済みです。

9.4.2. 既知の問題

現時点で OpenShift Serverless には以下の制限があります。

- Knative Serving Operator は、ServiceMeshMemberRoll が **knative-serving** namespace を組み込むまで待機する必要があります。インストール手順では、**knative-serving** namespace を作成してから Operator をインストールすることが推奨されます。Istio では、Knative Serving Pod の作成時に **knative-serving** namespace が ServiceMeshMemberRoll にあると認識しません。そのため、サイドカーは挿入されません。

- Knative サービスは、クラスターが長時間実行される場合に 503 ステータスコードを返します。Knative Serving Pod にはエラーは表示されません。**istio-pilot** Pod を再起動すると、問題は一時的に解決されます。

- gRPC および HTTP2 はルートに対して機能しません。これは OpenShift ルートの既知の制限です。