OpenShift Container Platform 4.3

Serverless applications

OpenShift Serverless installation, usage, and release notes

Last Updated: 2020-04-07

OpenShift Container Platform 4.3 Serverless applications

OpenShift Serverless installation, usage, and release notes

Abstract

This document provides information on how to use OpenShift Serverless in OpenShift Container Platform

Table of Contents

CHAPTER 1. OPENSHIFT SERVERLESS RELEASE NOTES

For an overview of OpenShift Serverless functionality, see Getting started with OpenShift Serverless.

1.1. GETTING SUPPORT

If you experience difficulty with a procedure described in this documentation, visit the Customer Portal to learn more about support for Technology Preview features.

1.2. RELEASE NOTES FOR RED HAT OPENSHIFT SERVERLESS TECHNOLOGY PREVIEW 1.6.0

1.2.1. New features

- OpenShift Serverless 1.6.0 is available on OpenShift Container Platform 4.3 and newer versions.
- OpenShift Serverless now uses Knative Serving 0.13.1.
- OpenShift Serverless now uses Knative **kn** CLI 0.13.1.
- OpenShift Serverless now uses Knative Serving Operator 0.13.1.
- The **serving.knative.dev** API group has now been fully deprecated and is replaced by the **operator.knative.dev** API group.
  You must complete the steps that are described in the OpenShift Serverless 1.4.0 release notes, that replace the **serving.knative.dev** API group with the **operator.knative.dev** API group, before you can upgrade to the latest version of OpenShift Serverless.

  > **IMPORTANT**
  >
  > This change causes commands without a fully qualified APIGroup and kind, such as **oc get knativeserving**, to become unreliable and not always work correctly.
  >
  > After upgrading to OpenShift Serverless 1.6.0, you must remove the old CRD to fix this issue. You can remove the old CRD by entering the following command:
  >
  > ```
  > $ oc delete crd knativeservings.serving.knative.dev
  > ```

- The **Subscription Update Channel** for new OpenShift Serverless releases was updated from **techpreview** to **preview-4.3**.

  > **IMPORTANT**
  >
  > You must update your channel by following the upgrade documentation to use the latest OpenShift Serverless version.

- OpenShift Serverless now supports the use of **HTTP_PROXY**.
- OpenShift Serverless now supports **HTTPS_PROXY** cluster-proxy settings.

> **NOTE**
>
> This **HTTP_PROXY** support does not include using custom certificates.

- The **KnativeServing** CRD is now hidden from the Developer Catalog by default so that only users with cluster administrator permissions can view it.
- Parts of the **KnativeServing** control plane and data plane are now deployed as highly available (HA) by default.
- Kourier is now actively watched and reconciles changes automatically.
- OpenShift Serverless now supports use on OpenShift Container Platform nightly builds.

1.2.2. Fixed issues

- In previous versions, the **oc explain** command did not work correctly. The structural schema of the **KnativeServing** CRD was updated in OpenShift Serverless 1.6.0 so that the **oc explain** command now works correctly.
- In previous versions, it was possible to create more than one **KnativeServing** CR. Multiple **KnativeServing** CRs are now prevented synchronously in OpenShift Serverless 1.6.0. Attempting to create more than one **KnativeServing** CR now results in an error.
- In previous versions, OpenShift Serverless was not compatible with OpenShift Container Platform deployments on GCP. This issue was fixed in OpenShift Serverless 1.6.0.
- In previous releases, the Knative Serving webhook crashed with an out of memory error if the cluster had more than 170 namespaces. This issue was fixed in OpenShift Serverless 1.6.0.
- In previous releases, OpenShift Serverless did not automatically fix an OpenShift Container Platform route that it created if the route was changed by another component. This issue was fixed in OpenShift Serverless 1.6.0.
- In previous versions, deleting a **KnativeServing** CR occasionally caused the system to hang. This issue was fixed in OpenShift Serverless 1.6.0.
- Due to the ingress migration from Service Mesh to Kourier that occured in OpenShift Serverless 1.5.0, orphaned VirtualServices sometimes remained on the system. In OpenShift Serverless 1.6.0, orphaned VirtualServices are automatically removed.

1.2.3. Known issues

- In OpenShift Serverless 1.6.0, if a cluster administrator uninstalls OpenShift Serverless by following the uninstall procedure provided in the documentation, the **Serverless** dropdown is still be visible in the **Administrator** perspective of the OpenShift Container Platform web console, and the **Knative Service** resource is still be visible in the **Developer** perspective of the OpenShift Container Platform web console. Although you can create Knative services by using this option, these Knative services do not work.
  To prevent OpenShift Serverless from being visible in the OpenShift Container Platform web console, the cluster administrator must delete additional CRDs from the deployment after removing the Knative Serving CR.

  Cluster administrators can remove these CRDs by entering the following command:

```
$ oc get crd -oname | grep -E '(serving|internal).knative.dev' | xargs oc delete
```

1.3. RELEASE NOTES FOR RED HAT OPENSHIFT SERVERLESS TECHNOLOGY PREVIEW 1.5.0

1.3.1. New features

- OpenShift Serverless 1.5.0 is available on OpenShift Container Platform 4.3 and newer versions.
- OpenShift Serverless now uses Knative Serving 0.12.1.
- OpenShift Serverless now uses Knative **kn** CLI 0.12.0.
- OpenShift Serverless now uses Knative Serving Operator 0.12.1.
- OpenShift Serverless ingress implementation was updated to use Kourier in place of Service Mesh. No user intervention is necessary, as this change is automatic when the OpenShift Serverless Operator is upgraded to 1.5.0.

1.3.2. Fixed issues

- In previous releases, OpenShift Container Platform scale from zero latency caused a delay of approximately 10 seconds when creating pods. This issue was fixed in the OpenShift Container Platform 4.3.5 bug fix update.

1.3.3. Known issues

- Deleting **KnativeServing.operator.knative.dev** from the **knative-serving** namespace may cause the deletion process to hang. This is due to a race condition between deletion of the CRD and **knative-openshift-ingress** removing finalizers.

1.4. RELEASE NOTES FOR RED HAT OPENSHIFT SERVERLESS TECHNOLOGY PREVIEW 1.4.0

> **IMPORTANT**
>
> OpenShift Serverless 1.4.0 contains a bad owner reference that causes the Kubernetes Garbage Collector to incorrectly remove the entire Knative control plane, including all of your services. You must install OpenShift Serverless 1.4.1 to fix this issue.

1.4.1. New features

- OpenShift Serverless 1.4.0 is available on OpenShift Container Platform 4.2 and newer versions.
- OpenShift Serverless now uses Knative Serving 0.11.1.
- OpenShift Serverless now uses Knative **kn** CLI 0.11.0.
- OpenShift Serverless now uses Knative Serving Operator 0.11.1.
- The **kn** CLI is now available for download through the **Command Line Tools** page in the OpenShift Container Platform web console.

- The **KnativeServing** object's API group has changed in this release from **serving.knative.dev** to **operator.knative.dev**. You will need to adjust any of your scripts or applications that rely on the old API group to use the new API group. The OpenShift Serverless installation instructions have been updated to use the new API group.
  If you must keep using the old group temporarily, you can use the old custom resource (CR) as before. However, this CR is deprecated and will eventually be removed.

  After you update references to the new API group, you can remove any older CR versions and use the newly deployed **KnativeServing** CR instead. To safely do this without downtime, remove the owner reference from the newly deployed **KnativeServing** CR using:

  ```
  $ oc edit knativeserving.operator.knative.dev knative-serving -n knative-serving
  ```

  After the owner reference has been removed, you can safely remove any older CR versions and start using the new one.

  IMPORTANT

  If a previous version of the CR exists, changes to the new CR will be overwritten by the OpenShift Serverless Operator. While the old CR is still active, all changes need to be made to that CR.

1.4.2. Fixed issues

- Connecting to a private, cluster local Knative Service from a namespace that was not part of the **knative-serving-ingress** Service Mesh was failing on **i/o timeout**. This issue is now fixed.

- The **container_name** and **pod_name** metric labels were removed in OpenShift Container Platform 4.3. The documentation has been updated to use the new **container** and **pod** metric labels instead. If you are using metering with Serverless on OpenShift Container Platform 4.3 or later, you must update your Prometheus queries according to the current version of the Serverless metering documentation.

1.4.3. Known issues

- Unqualified usage of **knativeserving** in **oc** commands no longer works because of the migration to a new API group. For example, this command will not work:

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

  Use the explicit fully-qualified format instead. For example:

  ```
  $ oc get knativeserving.operator.knative.dev -n knative-serving
  ```

- OpenShift Container Platform scale from zero latency causes a delay of approximately 10 seconds when creating pods. This is a current OpenShift Container Platform limitation.

1.5. ADDITIONAL RESOURCES

OpenShift Serverless is based on the open source Knative project.

- For details about the latest Knative Serving release, see the Knative Serving releases page.

- For details about the latest Knative Serving Operator release, see the Knative Serving Operator releases page.
- For details about the latest Knative CLI release, see the Knative client releases page .
- For details about the latest Knative Eventing release, see the Knative Eventing releases page .

NOTE

Knative Eventing is currently available as a Developer Preview on OpenShift Container Platform. See the upstream Knative Eventing on OpenShift Container Platform documentation.

CHAPTER 2. GETTING STARTED WITH OPENSHIFT SERVERLESS

IMPORTANT

OpenShift Serverless is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see https://access.redhat.com/support/offerings/techpreview/.

OpenShift Serverless simplifies the process of delivering code from development into production by reducing the need for infrastructure set up or back-end development by developers.

2.1. HOW OPENSHIFT SERVERLESS WORKS

Developers on OpenShift Serverless can use the provided Kubernetes-native APIs, as well as familiar languages and frameworks, to deploy applications and container workloads. For information about installing OpenShift Serverless, see Installing OpenShift Serverless.

OpenShift Serverless on OpenShift Container Platform enables stateful, stateless, and serverless workloads to all run on a single multi-cloud container platform with automated operations. Developers can use a single platform for hosting their microservices, legacy, and serverless applications.

OpenShift Serverless is based on the open source Knative project, which provides portability and consistency across hybrid and multi-cloud environments by enabling an enterprise-grade serverless platform.

2.2. KNATIVE EVENTING

A developer preview version of Knative Eventing is available for use with OpenShift Serverless. However, this is not included in the OpenShift Serverless Operator and is not currently supported as part of this Technology Preview. For more information about Knative Eventing, including installation instructions and samples, see the Knative Eventing on OpenShift Container Platform documentation.

2.3. KNATIVE CLI

The Knative CLI (**kn**) extends the functionality of the **oc** or **kubectl** tools to enable interaction with Knative components on OpenShift Container Platform. **kn** allows developers to deploy and manage applications without editing YAML files directly.

2.4. APPLICATIONS ON OPENSHIFT SERVERLESS

Applications are created using Custom Resource Definitions (CRDs) and associated controllers in Kubernetes, and are packaged as OCI compliant Linux containers that can be run anywhere.

To deploy applications in OpenShift Serverless, you must create Knative Services. For more information see Getting started with Knative Services.

CHAPTER 3. INSTALLING OPENSHIFT SERVERLESS

3.1. INSTALLING OPENSHIFT SERVERLESS

NOTE

OpenShift Serverless is supported for installation in a restricted network environment. For more information, see Using Operator Lifecycle Manager on restricted networks .

3.1.1. Prerequisites

- To run OpenShift Serverless, the OpenShift Container Platform cluster must be sized correctly. The minimum requirement to use OpenShift Serverless is a cluster with 10 CPUs and 40GB memory.

  NOTE

  The total size requirements to run OpenShift Serverless are dependent on the applications deployed. By default, each pod requests ~400m of CPU, so the minimum requirements are based on this value.

  In the size requirement provided, an application can scale up to 10 replicas. Lowering the actual CPU request of applications can increase the number of possible replicas.

- For more advanced use-cases such as logging, monitoring, or metering on OpenShift Container Platform, you must deploy more resources. Recommended requirements for such use-cases are 24 CPUs and 96GB of memory.
- You can use the MachineSet API to manually scale your cluster up to the desired size. The minimum requirements usually mean that you must scale up one of the default MachineSets by two additional machines.

NOTE

The requirements provided relate only to the pool of worker machines of the OpenShift Container Platform cluster. Master nodes are not used for general scheduling and are omitted from the requirements.

Additional resources

- For more information on using the MachineSet API, see the documentation on Creating MachineSets.
- For more information on scaling a MachineSet manually, see the documentation on manually scaling MachineSets.

3.1.2. Installing the OpenShift Serverless Operator

The OpenShift Serverless Operator can be installed by a cluster administrator using the Operator Hub in the OpenShift Container Platform web console. For details, see the OpenShift Container Platform documentation on adding Operators to a cluster .

Next steps

- After the OpenShift Serverless Operator is installed, you can install the Knative Serving component. See the documentation on Installing Knative Serving.

3.2. INSTALLING KNATIVE SERVING

You must create a **KnativeServing** object to install Knative Serving using the OpenShift Serverless Operator.

> **IMPORTANT**
>
> You must create the **KnativeServing** object in the **knative-serving** namespace, as shown in the sample YAML, or it is ignored.

Sample serving.yaml

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

Prerequisite

- An account with cluster administrator access.
- Installed OpenShift Serverless Operator.

Procedure

1. Copy the sample YAML file into the **serving.yaml** file and enter the following command to apply it:

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

2. To verify the installation is complete, enter the following command:

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

   Results should be similar to:

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

3.3. UPGRADING OPENSHIFT SERVERLESS

If you have previously installed a Technology Preview version of OpenShift Serverless, follow the instructions in this guide to upgrade to the latest version.

3.3.1. Upgrading the Subscription Channel

In OpenShift Serverless versions 1.5.0 and older, the only available Subscription Update Channel was **techpreview**. To upgrade to the latest version, you must update the channel to **preview-4.3**.

> **NOTE**
>
> If you have selected Manual updates, you will need to complete additional steps after updating the channel as described in this guide. The Subscription's upgrade status will remain **Upgrading** until you review and approve its Install Plan. Information about the Install Plan can be found in the OpenShift Container Platform Operators documentation.

Prerequisites

- You have installed a Technology Preview version of OpenShift Serverless Operator, and have selected Automatic updates during the installation process.
- You have logged in to the OpenShift Container Platform web console.

Procedure

1. Select the **openshift-operators** namespace in the OpenShift Container Platform web console.
2. Navigate to the **Operators** → **Installed Operators** page.
3. Select the **OpenShift Serverless Operator Operator**.
4. Click **Subscription** → **Channel**.
5. In the **Change Subscription Update Channel** window, select **preview-4.3**, and then click **Save**.
6. Wait until all pods have been upgraded in the **knative-serving** namespace and the KnativeServing custom resource reports the latest Knative Serving version.

Verification steps

To verify that the upgrade has been successful, you can check the status of pods in the **knative-serving** namespace, and the version of the KnativeServing CR.

1. Check the status of the pods by entering the following command:

```
$ oc get knativeserving.operator.knative.dev knative-serving -n knative-serving -
o=jsonpath='{.status.conditions[?(@.type=="Ready")].status}'
```

   The previous command should return a status of **True**.

2. Check the version of the KnativeServing CR by entering the following command:

```
$ oc get knativeserving.operator.knative.dev knative-serving -n knative-serving -
o=jsonpath='{.status.version}'
```

The previous command should return the latest version of Knative Serving. You can check the latest version in the OpenShift Serverless Operator release notes.

3.3.2. Updating the API group

In OpenShift Serverless 1.4.0 and newer versions, the **KnativeServing** object's API group changed from **serving.knative.dev** to **operator.knative.dev**. You must adjust any of your scripts or applications that rely on the old API group to use the new API group.

Procedure

1. Update your scripts or applications that rely on the old API group to use the new API group.
2. After you update references to the new API group, you must remove any older custom resource (CR) versions and use the newly deployed **KnativeServing** CR instead.
   To safely do this without downtime, remove the owner reference from the newly deployed **KnativeServing** CR by entering the following command:

   ```
   $ oc edit knativeserving.operator.knative.dev knative-serving -n knative-serving
   ```

3. After the owner reference has been removed, you can safely remove any older CR versions and start using the new one.
4. This change to the API group causes commands without a fully qualified API group and kind, such as **oc get knativeserving**, to become unreliable and not always work correctly.
   After upgrading to OpenShift Serverless 1.6.0, you must remove the old custom resource definition (CRD) to fix this issue.

   You can remove the old CRD by entering the following command:

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

> **IMPORTANT**
>
> If a previous version of the CR exists, changes to the new CR will be overwritten by the OpenShift Serverless Operator. While the old CR is still active, all changes need to be made to that CR.

3.4. INSTALLING THE KNATIVE CLI (KN)

> **NOTE**
>
> **kn** does not have its own login mechanism. To log in to the cluster, you must install the **oc** CLI and use **oc** login.
>
> Installation options for the **oc** CLI will vary depending on your operating system.
>
> For more information on installing the **oc** CLI for your operating system and logging in with **oc**, see the CLI getting started documentation.

3.4.1. Installing the kn CLI using the OpenShift Container Platform web console

Once the OpenShift Serverless Operator is installed, you will see a link to download the **kn** CLI for Linux, macOS and Windows from the **Command Line Tools** page in the OpenShift Container Platform web console.

You can access the **Command Line Tools** page by clicking the ? icon in the top right corner of the web console and selecting **Command Line Tools** in the drop down menu.

Procedure

1. Download the **kn** CLI from the **Command Line Tools** page.

2. Unpack the archive:

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

3. Move the **kn** binary to a directory on your PATH.

4. To check your path, run:

```
$ echo $PATH
```

> **NOTE**
>
> If you do not use RHEL or Fedora, ensure that **libc** is installed in a directory on your library path. If **libc** is not available, you might see the following error when you run CLI commands:
>
> ```
> $ kn: No such file or directory
> ```

3.4.2. Installing the kn CLI for Linux using an RPM

For Red Hat Enterprise Linux (RHEL), you can install **kn** as an RPM if you have an active OpenShift Container Platform subscription on your Red Hat account.

Procedure

- Use the following command to install **kn**:

```
# 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 Pool ID for an active OpenShift Container Platform subscription

3.4.3. Installing the kn CLI for Linux

For Linux distributions, you can download the CLI directly as a **tar.gz** archive.

Procedure

1. Download the CLI.
2. Unpack the archive:

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

3. Move the **kn** binary to a directory on your PATH.
4. To check your path, run:

```
$ echo $PATH
```

> **NOTE**
>
> If you do not use RHEL or Fedora, ensure that **libc** is installed in a directory on your library path. If **libc** is not available, you might see the following error when you run CLI commands:
>
> ```
> $ kn: No such file or directory
> ```

3.4.4. Installing the kn CLI for macOS

kn for macOS is provided as a **tar.gz** archive.

Procedure

1. Download the CLI.
2. Unpack and unzip the archive.
3. Move the **kn** binary to a directory on your PATH.
4. To check your PATH, open a terminal window and run:

```
$ echo $PATH
```

3.4.5. Installing the kn CLI for Windows

The CLI for Windows is provided as a zip archive.

Procedure

1. Download the CLI.
2. Unzip the archive with a ZIP program.
3. Move the **kn** binary to a directory on your PATH.
4. To check your PATH, open the Command Prompt and run the command:

```
C:\> path
```

3.5. REMOVING OPENSHIFT SERVERLESS

This guide provides details of how to remove the OpenShift Serverless Operator and other OpenShift Serverless components.

NOTE

Before you can remove the OpenShift Serverless Operator, you must remove Knative Serving and Knative Eventing.

3.5.1. Uninstalling Knative Serving

To uninstall Knative Serving, you must remove its custom resource and delete the **knative-serving** namespace.

Procedure

1. To remove Knative Serving, enter the following command:

```
$ oc delete knativeservings.operator.knative.dev knative-serving -n knative-serving
```

2. After the command has completed and all pods have been removed from the **knative-serving** namespace, delete the namespace by entering the following command:

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

3.5.2. Removing the OpenShift Serverless Operator

You can remove the OpenShift Serverless Operator from the host cluster by following the documentation on deleting Operators from a cluster.

3.5.3. Deleting OpenShift Serverless CRDs

After uninstalling the OpenShift Serverless, the Operator and API CRDs remain on the cluster. You can use the following procedure to remove the remaining CRDs.

IMPORTANT

Removing the Operator and API CRDs also removes all resources that were defined using them, including Knative services.

Prerequisites

- You uninstalled Knative Serving and removed the OpenShift Serverless Operator.

Procedure

1. To delete the remaining OpenShift Serverless CRDs, enter the following command:

```
$ oc get crd -oname | grep 'knative.dev' | xargs oc delete
```

CHAPTER 4. KNATIVE SERVING

4.1. HOW KNATIVE SERVING WORKS

Knative Serving on OpenShift Container Platform builds on Kubernetes and Istio to support deploying and serving serverless applications.

It creates a set of Kubernetes Custom Resource Definitions (CRDs) that are used to define and control the behavior of serverless workloads on an OpenShift Container Platform cluster.

These CRDs are building blocks to address complex use cases, for example: * Rapidly deploying serverless containers. * Automatically scaling pods. * Viewing point-in-time snapshots of deployed code and configurations.

4.1.1. Knative Serving components

The components described in this section are the resources that Knative Serving requires to be configured and run correctly.

Knative service resource

The **service.serving.knative.dev** resource automatically manages the whole lifecycle of a serverless workload on a cluster. It controls the creation of other objects to ensure that an app has a route, a configuration, and a new revision for each update of the service. Services can be defined to always route traffic to the latest revision or to a pinned revision.

Knative route resource

The **route.serving.knative.dev** resource maps a network endpoint to one or more Knative revisions. You can manage the traffic in several ways, including fractional traffic and named routes.

Knative configuration resource

The **configuration.serving.knative.dev** resource maintains the required state for your deployment. Modifying a configuration creates a new revision.

Knative revision resource

The **revision.serving.knative.dev** resource is a point-in-time snapshot of the code and configuration for each modification made to the workload. Revisions are immutable objects and can be retained for as long as needed. Cluster administrators can modify the **revision.serving.knative.dev** resource to enable automatic scaling of Pods in your OpenShift Container Platform cluster.

4.2. GETTING STARTED WITH KNATIVE SERVICES

Knative services are Kubernetes services that a user creates to deploy a serverless application. Each Knative service is defined by a route and a configuration, contained in a **.yaml** file.

4.2.1. Creating a Knative service

To create a service, you must create the **service.yaml** file.

You can copy the sample below. This sample will create a sample golang application called **helloworld-go** and allows you to specify the image for that application.

```
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. Current version of Knative
2. The name of the application
3. The namespace the application will use
4. The URL to the image of the application
5. The environment variable printed out by the sample application

4.2.2. Deploying a serverless application

To deploy a serverless application, you must apply the **service.yaml** file.

Procedure

1. Navigate to the directory where the **service.yaml** file is contained.
2. Deploy the application by applying the **service.yaml** file.

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

Now that service has been created and the application has been deployed, Knative will create a new immutable revision for this version of the application.

Knative will also perform network programming to create a route, ingress, service, and load balancer for your application, and will automatically scale your pods up and down based on traffic, including inactive pods.

4.2.3. Connecting Knative Services to existing Kubernetes deployments

Knative Services can call a Kubernetes deployment in any namespace, provided that there are no existing additional network barriers.

A Kubernetes deployment can call a Knative Service if:

- The Kubernetes deployment is in the same namespace as the target Knative Service.
- The Kubernetes deployment is in a namespace that was manually added to the ServiceMeshMemberRoll in **knative-serving-ingress**.
- The Kubernetes deployment uses the target Knative Service's public URL.

NOTE

Knative Services are accessed using a public URL by default. The target Knative Service must not be configured as a private, **cluster-local** visibility service if you want to connect it to your existing Kubernetes deploying using a public URL.

4.3. CREATING SERVERLESS APPLICATIONS

You can create serverless applications by using the **Developer** perspective in the OpenShift Container Platform web console.

Prerequisites

To create serverless applications using the **Developer** perspective ensure that:

- You have logged in to the web console .
- You are in the **Developer** perspective.
- You have the appropriate roles and permissions in a project to create applications and other workloads in OpenShift Container Platform.
- You have installed the Openshift Serverless Operator.
- You have created a knative-serving namespace and a KnativeServing resource in the knative-serving namespace.

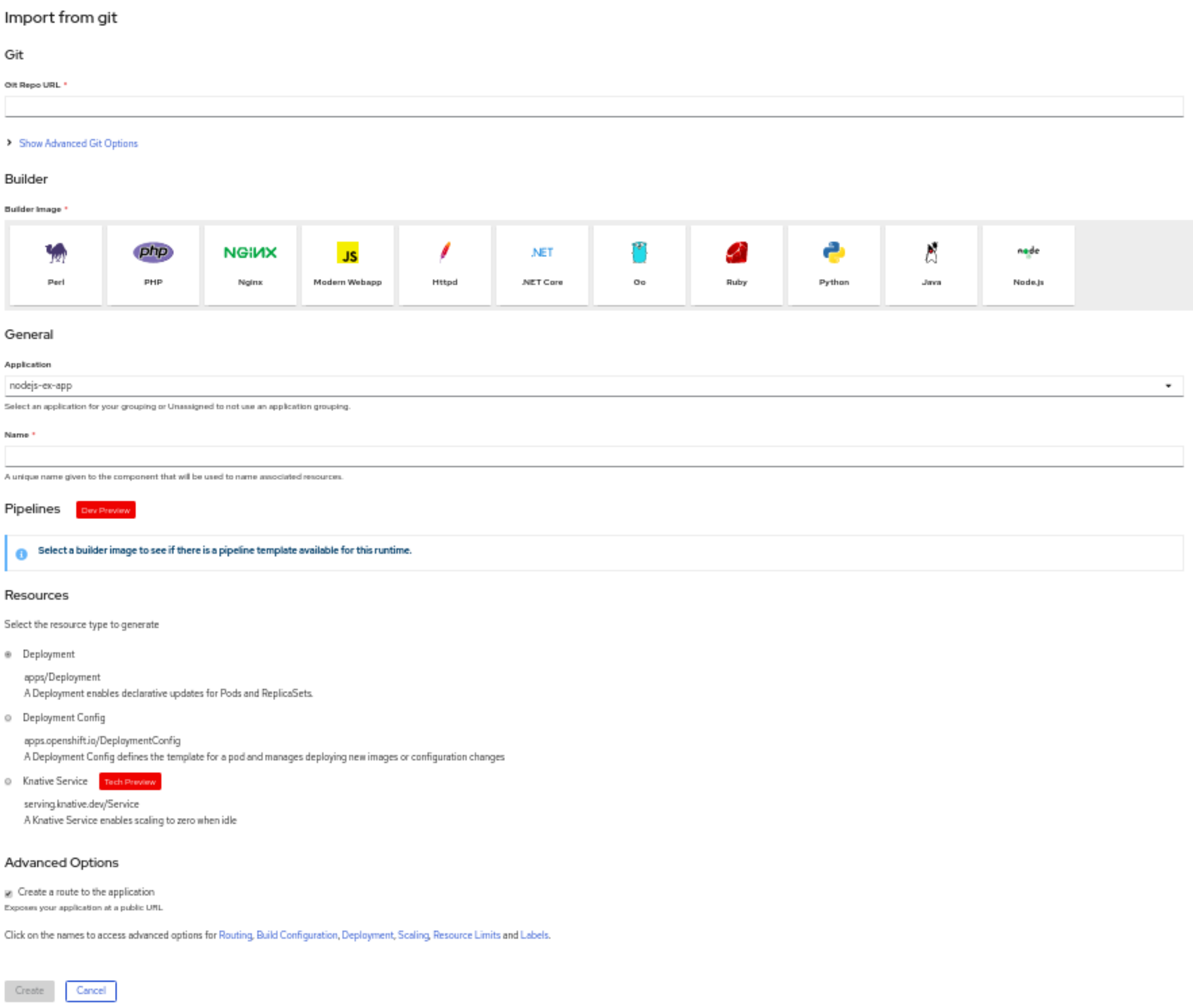
4.3.1. Importing a codebase from Git to create an application

The following procedure walks you through the **Import from Git** option in the **Developer** perspective to create an application.

Create, build, and deploy an application on OpenShift Container Platform using an existing codebase in GitHub as follows:

Procedure

1. In the **Add** view, click **From Git** to see the **Import from git** form.



2. In the **Git** section, enter the Git repository URL for the codebase you want to use to create an application. For example, enter the URL of this sample Node.js application **https://github.com/sclorg/nodejs-ex**. The URL is then validated.

3. Optionally, you can click **Show Advanced Git Options** to add details such as:

   - **Git Reference** to point to code in a specific branch, tag, or commit to be used to build the application.

   - **Context Dir** to specify the subdirectory for the application source code you want to use to build the application.

   - **Source Secret** to create a **Secret Name** with credentials for pulling your source code from a private repository.

4. In the **Builder** section, after the URL is validated, an appropriate builder image is detected, indicated by a star, and automatically selected. For the **https://github.com/sclorg/nodejs-ex** Git URL, the Node.js builder image is selected by default. If required, you can change the version using the **Builder Image Version** drop-down list.

5. In the **General** section:

   a. In the **Application** field, enter a unique name for the application grouping, for example, **myapp**. Ensure that the application name is unique in a namespace.

   b. The **Name** field to identify the resources created for this application is automatically populated based on the Git repository URL.

> **NOTE**
>
> The resource name must be unique in a namespace. Modify the resource name if you get an error.

6. In the **Resources** section, select:
   - **Deployment**, to create an application in plain Kubernetes style.
   - **Deployment Config**, to create an OpenShift style application.
   - **Knative Service**, to create a microservice.

> **NOTE**
>
> The **Knative Service** option is displayed in the **Import from git** form only if the **Serverless Operator** is installed in your cluster. For further details refer to documentation on installing OpenShift Serverless.

7. In the **Advanced Options** section, the **Create a route to the application** is selected by default so that you can access your application using a publicly available URL. You can clear the check box if you do not want to expose your application on a public route.

8. Optionally, you can use the following advanced options to further customize your application:

   Routing

   Click the **Routing** link to:
   - Customize the hostname for the route.
   - Specify the path the router watches.
   - Select the target port for the traffic from the drop-down list.
   - Secure your route by selecting the **Secure Route** check box. Select the required TLS termination type and set a policy for insecure traffic from the respective drop-down lists.
     For serverless applications, the Knative Service manages all the routing options above. However, you can customize the target port for traffic, if required. If the target port is not specified, the default port of **8080** is used.

   Build and Deployment Configuration

   Click the **Build Configuration** and **Deployment Configuration** links to see the respective configuration options. Some of the options are selected by default; you can customize them further by adding the necessary triggers and environment variables. For serverless applications, the **Deployment Configuration** option is not displayed as the Knative configuration resource maintains the desired state for your deployment instead of a DeploymentConfig.

   Scaling

   Click the **Scaling** link to define the number of Pods or instances of the application you want to deploy initially.
   For serverless applications, you can:

- Set the upper and lower limit for the number of pods that can be set by the autoscaler. If the lower limit is not specified, it defaults to zero.
- Define the soft limit for the required number of concurrent requests per instance of the application at a given time. It is the recommended configuration for autoscaling. If not specified, it takes the value specified in the cluster configuration.
- Define the hard limit for the number of concurrent requests allowed per instance of the application at a given time. This is configured in the revision template. If not specified, it defaults to the value specified in the cluster configuration.

Resource Limit

Click the **Resource Limit** link to set the amount of **CPU** and **Memory** resources a container is guaranteed or allowed to use when running.

Labels

Click the **Labels** link to add custom labels to your application.

9. Click **Create** to create the application and see its build status in the **Topology** view.

4.4. INTERACTING WITH YOUR SERVERLESS APPLICATION

This section provides information about various tasks that can be performed once you have successfully created a serverless application.

4.4.1. Verifying your serverless application deployment

To verify that your serverless application has been deployed successfully, you must get the application host created by Knative, and then send a request to that host and observe the output.

NOTE

If you changed the application name from **helloworld-go** to a new name when creating the **service.yaml** file, replace **helloworld-go** in the commands with the new name you created.

Procedure

1. Find the application host.

```
oc get ksvc helloworld-go
NAME            URL                                         LATESTCREATED         LATESTREADY
READY   REASON
helloworld-go   http://helloworld-go.default.example.com   helloworld-go-4wsd2   helloworld-
go-4wsd2   True
```

2. Make a request to your cluster and observe the output.

```
$ curl http://helloworld-go.default.example.com
Hello World: Go Sample v1!
```

4.4.2. Interacting with a serverless application using HTTP2 / gRPC

OpenShift Container Platform routes do not support HTTP2, and therefore do not support gRPC as this is transported by HTTP2. If you use these protocols in your application, you must call the application using the ingress gateway directly. To do this you must find the ingress gateway's public address and the application's specific host.

Procedure

1. Find the application host. See the instructions in *Verifying your serverless application deployment*.

2. The ingress gateway's public address can be determined using this command:

```
oc -n istio-system get svc istio-ingressgateway
```

   The output will be similar to this example:

```
NAME                   TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)
AGE
istio-ingressgateway   LoadBalancer   172.30.51.103
a83e86291bcdd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com
80:31380/TCP,443:31390/TCP,31400:31400/TCP,15029:30672/TCP,15030:30970/TCP,1503
1:32657/TCP,15032:32400/TCP,15443:30167/TCP,15020:32285/TCP   67m
```

   The public address is surfaced in the **EXTERNAL-IP** field, and in this case would be:

```
a83e86291bcdd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com
```

3. Manually set the host header of your HTTP request to the application's host, but direct the request itself against the public address of the ingress gateway.
   Here is an example, using the information obtained from the steps in *Verifying your serverless application deployment*:

```
$ curl -H "Host: helloworld-go.default.example.com" a83e86291bcdd11e993af02b7a65e514-
33544245.us-east-1.elb.amazonaws.com
Hello Go Sample v1!
```

   You can also make a gRPC request by setting the authority to the application's host, while directing the request against the ingress gateway directly.

   Here is an example of what that looks like in the Golang gRPC client:

   NOTE

   Ensure that you append the respective port (80 by default) to both hosts as shown in the example.

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

4.5. CONFIGURING KNATIVE SERVING AUTOSCALING

OpenShift Serverless provides capabilities for automatic Pod scaling, including scaling inactive Pods to zero, by enabling the Knative Serving autoscaling system in an OpenShift Container Platform cluster.

To enable autoscaling for Knative Serving, you must configure concurrency and scale bounds in the revision template.

> **NOTE**
>
> Any limits or targets set in the revision template are measured against a single instance of your application. For example, setting the **target** annotation to **50** will configure the autoscaler to scale the application so that each instance of it will handle 50 requests at a time.

4.5.1. Configuring concurrent requests for Knative Serving autoscaling

You can specify the number of concurrent requests that should be handled by each instance of an application (revision container) by adding the **target** annotation or the **containerConcurrency** field in the revision template.

Here is an example of **target** being used in a revision template:

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

Here is an example of **containerConcurrency** being used in a revision template:

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

Adding a value for both **target** and **containerConcurrency** will target the **target** number of concurrent requests, but impose a hard limit of the **containerConcurrency** number of requests.

For example, if the **target** value is 50 and the **containerConcurrency** value is 100, the targeted number of requests will be 50, but the hard limit will be 100.

If the **containerConcurrency** value is less than the **target** value, the **target** value will be tuned down, since there is no need to target more requests than the number that can actually be handled.

NOTE

containerConcurrency should only be used if there is a clear need to limit how many requests reach the application at a given time. Using **containerConcurrency** is only advised if the application needs to have an enforced constraint of concurrency.

4.5.1.1. Configuring concurrent requests using the target annotation

The default target for the number of concurrent requests is **100**, but you can override this value by adding or modifying the **autoscaling.knative.dev/target** annotation value in the revision template.

Here is an example of how this annotation is used in the revision template to set the target to **50**.

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

4.5.1.2. Configuring concurrent requests using the containerConcurrency field

containerConcurrency sets a hard limit on the number of concurrent requests handled.

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

0

allows unlimited concurrent requests.

1

guarantees that only one request is handled at a time by a given instance of the revision container.

2 or more

will limit request concurrency to that value.

NOTE

If there is no **target** annotation, autoscaling is configured as if **target** is equal to the value of **containerConcurrency**.

4.5.2. Configuring scale bounds Knative Serving autoscaling

The **minScale** and **maxScale** annotations can be used to configure the minimum and maximum number of Pods that can serve applications. These annotations can be used to prevent cold starts or to help control computing costs.

minScale

If the **minScale** annotation is not set, Pods will scale to zero (or to 1 if enable-scale-to-zero is false per the **ConfigMap**).

maxScale

If the **maxScale** annotation is not set, there will be no upper limit for the number of Pods created.

minScale and **maxScale** can be configured as follows in the revision template:

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

Using these annotations in the revision template will propagate this confguration to **PodAutoscaler** objects.

NOTE

These annotations apply for the full lifetime of a revision. Even when a revision is not referenced by any route, the minimal Pod count specified by **minScale** will still be provided. Keep in mind that non-routeable revisions may be garbage collected, which enables Knative to reclaim the resources.

4.6. CLUSTER LOGGING WITH OPENSHIFT SERVERLESS

4.6.1. Cluster logging

OpenShift Container Platform cluster administrators can deploy cluster logging using a few CLI commands and the OpenShift Container Platform web console to install the Elasticsearch Operator and Cluster Logging Operator. When the operators are installed, create a Cluster Logging Custom Resource (CR) to schedule cluster logging pods and other resources necessary to support cluster logging. The operators are responsible for deploying, upgrading, and maintaining cluster logging.

You can configure cluster logging by modifying the Cluster Logging Custom Resource (CR), named **instance**. The CR defines a complete cluster logging deployment that includes all the components of the logging stack to collect, store and visualize logs. The Cluster Logging Operator watches the **ClusterLogging** Custom Resource and adjusts the logging deployment accordingly.

Administrators and application developers can view the logs of the projects for which they have view access.

4.6.2. About deploying and configuring cluster logging

OpenShift Container Platform cluster logging is designed to be used with the default configuration, which is tuned for small to medium sized OpenShift Container Platform clusters.

The installation instructions that follow include a sample Cluster Logging Custom Resource (CR), which you can use to create a cluster logging instance and configure your cluster logging deployment.

If you want to use the default cluster logging install, you can use the sample CR directly.

If you want to customize your deployment, make changes to the sample CR as needed. The following describes the configurations you can make when installing your cluster logging instance or modify after installation. See the Configuring sections for more information on working with each component, including modifications you can make outside of the Cluster Logging Custom Resource.

4.6.2.1. Configuring and Tuning Cluster Logging

You can configure your cluster logging environment by modifying the Cluster Logging Custom Resource deployed in the **openshift-logging** project.

You can modify any of the following components upon install or after install:

Memory and CPU

You can adjust both the CPU and memory limits for each component by modifying the **resources** block with valid memory and CPU values:

```
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 storage

You can configure a persistent storage class and size for the Elasticsearch cluster using the **storageClass name** and **size** parameters. The Cluster Logging Operator creates a **PersistentVolumeClaim** for each data node in the Elasticsearch cluster based on these parameters.

```
spec:
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 3
      storage:
        storageClassName: "gp2"
        size: "200G"
```

This example specifies each data node in the cluster will be bound to a **PersistentVolumeClaim** that requests "200G" of "gp2" storage. Each primary shard will be backed by a single replica.

NOTE

Omitting the **storage** block results in a deployment that includes ephemeral storage only.

```
spec:
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 3
      storage: {}
```

Elasticsearch replication policy

You can set the policy that defines how Elasticsearch shards are replicated across data nodes in the cluster:

- **FullRedundancy**. The shards for each index are fully replicated to every data node.
- **MultipleRedundancy**. The shards for each index are spread over half of the data nodes.
- **SingleRedundancy**. A single copy of each shard. Logs are always available and recoverable as long as at least two data nodes exist.
- **ZeroRedundancy**. No copies of any shards. Logs may be unavailable (or lost) in the event a node is down or fails.

Curator schedule

You specify the schedule for Curator in the cron format.

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

4.6.2.2. Sample modified Cluster Logging Custom Resource

The following is an example of a Cluster Logging Custom Resource modified using the options previously described.

Sample modified Cluster Logging Custom Resource

```
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
```

4.6.3. Using cluster logging to find logs for Knative Serving components

Procedure

1. To open the Kibana UI, the visualization tool for Elasticsearch, use the following command to get the Kibana route:

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

2. Use the route's URL to navigate to the Kibana dashboard and log in.

3. Ensure the index is set to **.all**. If the index is not set to **.all**, only the OpenShift system logs will be listed.

4. You can filter the logs by using the **knative-serving** namespace. Enter **kubernetes.namespace_name:knative-serving** in the search box to filter results.

> NOTE
>
> Knative Serving uses structured logging by default. You can enable the parsing of these logs by customizing the cluster logging Fluentd settings. This makes the logs more searchable and enables filtering on the log level to quickly identify issues.

4.6.4. Using cluster logging to find logs for services deployed with Knative Serving

With OpenShift Cluster Logging, the logs that your applications write to the console are collected in Elasticsearch. The following procedure outlines how to apply these capabilities to applications deployed by using Knative Serving.

Procedure

1. Use the following command to find the URL to Kibana:

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

2. Enter the URL in your browser to open the Kibana UI.

3. Ensure the index is set to **.all**. If the index is not set to **.all**, only the OpenShift system logs will be listed.

4. Filter the logs by using the Kubernetes namespace your service is deployed in. Add a filter to identify the service itself: **kubernetes.namespace_name:default AND kubernetes.labels.serving_knative_dev\/service:{SERVICE_NAME}**.

> NOTE
>
> You can also filter by using **/configuration** or **/revision**.

5. You can narrow your search by using **kubernetes.container_name:<user-container>** to only display the logs generated by your application. Otherwise, you will see logs from the queue-proxy.

> **NOTE**
>
> Use JSON-based structured logging in your application to allow for the quick filtering of these logs in production environments.

4.7. SPLITTING TRAFFIC BETWEEN REVISIONS

4.7.1. Splitting traffic between revisions using the Developer perspective

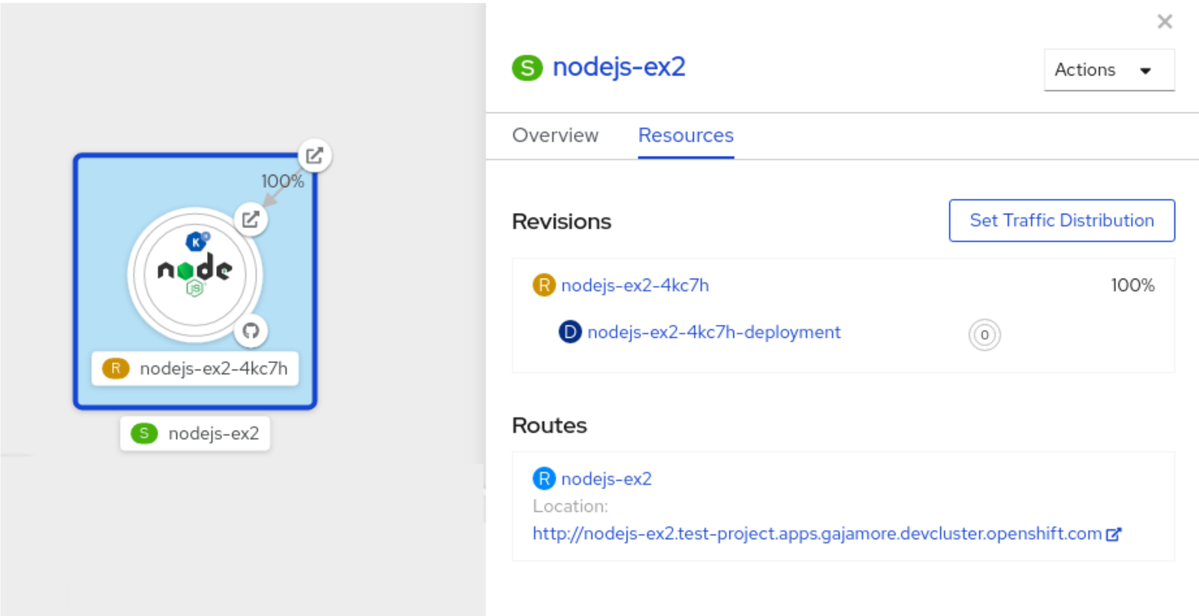
After you create a serverless application, the serverless application is displayed in the **Topology** view of the **Developer** perspective. The application revision is represented by the node and the serverless resource service is indicated by a quadrilateral around the node.

Any new change in the code or the service configuration triggers a revision, a snapshot of the code at a given time. For a service, you can manage the traffic between the revisions of the service by splitting and routing it to the different revisions as required.

Procedure

To split traffic between multiple revisions of an application in the **Topology** view:

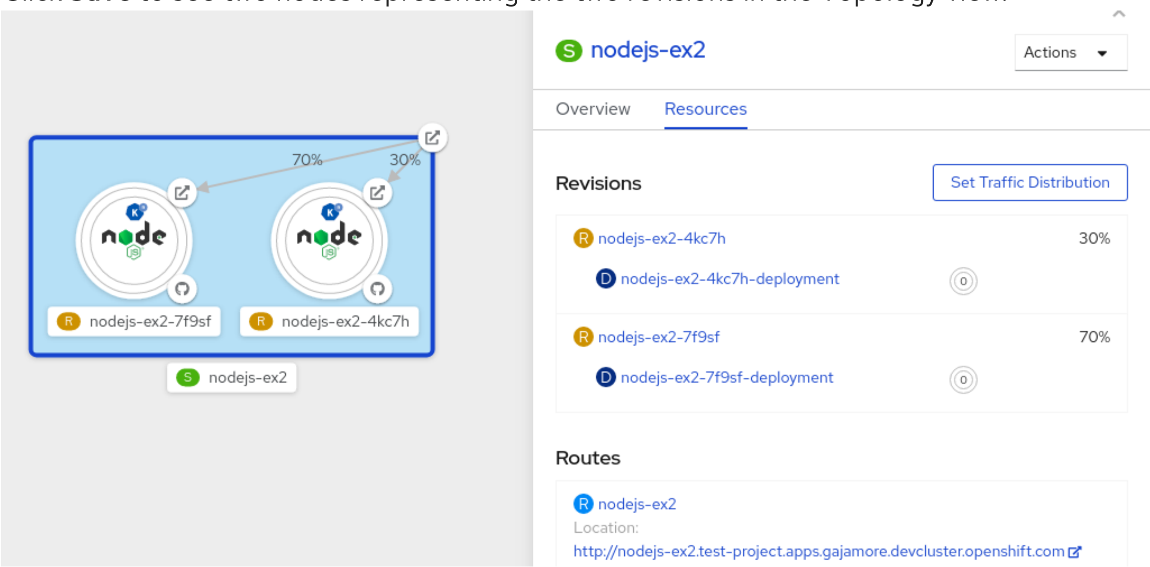
1. Click the serverless resource service, indicated by the quadrilateral, to see its overview in the side panel.
2. Click the **Resources** tab, to see a list of **Revisions** and **Routes** for the service.



3. Click the service, indicated by the **S** icon at the top of the side panel, to see an overview of the service details.
4. Click the **YAML** tab and modify the service configuration in the YAML editor, and click **Save**. For example, change the **timeoutseconds** from 300 to 301 . This change in the configuration triggers a new revision. In the **Topology** view, the latest revision is displayed and the **Resources** tab for the service now displays the two revisions.
5. In the **Resources** tab, click the **Set Traffic Distribution** button to see the traffic distribution dialog box:
    a. Add the split traffic percentage portion for the two revisions in the **Splits** field.

b. Add tags to create custom URLs for the two revisions.

c. Click **Save** to see two nodes representing the two revisions in the Topology view.



CHAPTER 5. KNATIVE CLI

5.1. GETTING STARTED WITH KNATIVE CLI (KN)

kn exposes commands for managing your applications, as well as lower level tools to interact with components of OpenShift Container Platform. With **kn**, you can create applications and manage OpenShift Container Platform projects from the terminal.

5.1.1. Basic workflow using kn

Use this basic workflow to create, read, update, delete (CRUD) operations on a service. The following example deploys a simple Hello World service that reads the environment variable **TARGET** and prints its output.

Procedure

1. Create a service in the **default** namespace from an image.

```
$ 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. List the service.

```
$ 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. Check if the service is working by using the **curl** service endpoint command:

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

Hello Knative!
```

4. Update the service.

```
$ 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
```

The service's environment variable **TARGET** is now set to **Kn**.

5. Describe the service.

```
$ 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. Delete the service.

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

You can then verify that the **hello** service is deleted by attempting to **list** it.

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

5.1.2. Autoscaling workflow using kn

You can access autoscaling capabilities by using **kn** to modify Knative services without editing YAML files directly.

Use the **service create** and **service update** commands with the appropriate flags to configure the autoscaling behavior.

Flag	Description
--concurrency-limit int	Hard limit of concurrent requests to be processed by a single replica.
--concurrency-target int	Recommendation for when to scale up based on the concurrent number of incoming requests. Defaults to **--concurrency-limit**.
--max-scale int	Maximum number of replicas.

Flag	Description
--min-scale int	Minimum number of replicas.

5.1.3. Traffic splitting using kn

kn helps you control which revisions get routed traffic on your Knative service.

Knative service allows for traffic mapping, which is the mapping of revisions of the service to an allocated portion of traffic. It offers the option to create unique URLs for particular revisions and has the ability to assign traffic to the latest revision.

With every update to the configuration of the service, a new revision is created with the service route pointing all the traffic to the latest ready revision by default.

You can change this behavior by defining which revision gets a portion of the traffic.

Procedure

- Use the **kn service update** command with the **--traffic** flag to update the traffic.

NOTE

--traffic RevisionName=Percent uses the following syntax:

- The **--traffic** flag requires two values separated by separated by an equals sign (**=**).
- The **RevisionName** string refers to the name of the revision.
- **Percent** integer denotes the traffic portion assigned to the revision.
- Use identifier **@latest** for the RevisionName to refer to the latest ready revision of the service. You can use this identifier only once with the **--traffic** flag.
- If the **service update** command updates the configuration values for the service along with traffic flags, the **@latest** reference will point to the created revision to which the updates are applied.
- **--traffic** flag can be specified multiple times and is valid only if the sum of the **Percent** values in all flags totals 100.

NOTE

For example, to route 10% of traffic to your new revision before putting all traffic on, use the following command:

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

5.1.3.1. Assigning tag revisions

A tag in a traffic block of service creates a custom URL, which points to a referenced revision. A user can define a unique tag for an available revision of a service which creates a custom URL by using the format **http(s)://TAG-SERVICE.DOMAIN**.

A given tag must be unique to its traffic block of the service. **kn** supports assigning and unassigning custom tags for revisions of services as part of the **kn service update** command.

NOTE

If you have assigned a tag to a particular revision, a user can reference the revision by its tag in the **--traffic** flag as **--traffic Tag=Percent**.

Procedure

- Use the following command:

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

NOTE

--tag RevisionName=Tag uses the following syntax:

- **--tag** flag requires two values separated by a **=**.
- **RevisionName** string refers to name of the **Revision**.
- **Tag** string denotes the custom tag to be given for this Revision.
- Use the identifier **@latest** for the RevisionName to refer to the latest ready revision of the service. You can use this identifier only once with the **--tag** flag.
- If the **service update** command is updating the configuration values for the Service (along with tag flags), **@latest** reference will be pointed to the created Revision after applying the update.
- **--tag** flag can be specified multiple times.
- **--tag** flag may assign different tags to the same revision.

5.1.3.2. Unassigning tag revisions

Tags assigned to revisions in a traffic block can be unassigned. Unassigning tags removes the custom URLs.

NOTE

If a revision is untagged and it is assigned 0% of the traffic, it is removed from the traffic block entirely.

Procedure

- A user can unassign the tags for revisions using the **kn service update** command:

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

> **NOTE**
>
> **--untag Tag** uses the following syntax:
>
> - The **--untag** flag requires one value.
> - The **tag** string denotes the unique tag in the traffic block of the service which needs to be unassigned. This also removes the respective custom URL.
> - The **--untag** flag can be specified multiple times.

5.1.3.3. Traffic flag operation precedence

All traffic-related flags can be specified using a single **kn service update** command. **kn** defines the precedence of these flags. The order of the flags specified when using the command is not taken into account.

The precedence of the flags as they are evaluated by **kn** are:

1. **--untag**: All the referenced revisions with this flag are removed from the traffic block.
2. **--tag**: Revisions are tagged as specified in the traffic block.
3. **--traffic**: The referenced revisions are assigned a portion of the traffic split.

5.1.3.4. Traffic splitting flags

kn supports traffic operations on the traffic block of a service as part of the **kn service update** command.

The following table displays a summary of traffic splitting flags, value formats, and the operation the flag performs. The "Repetition" column denotes whether repeating the particular value of flag is allowed in a **kn service update** command.

Flag	Value(s)	Operation	Repetition
--traffic	**RevisionName=Percent**	Gives **Percent** traffic to **RevisionName**	Yes
--traffic	**Tag=Percent**	Gives **Percent** traffic to the Revision having **Tag**	Yes
--traffic	**@latest=Percent**	Gives **Percent** traffic to the latest ready Revision	No
--tag	**RevisionName=Tag**	Gives **Tag** to **RevisionName**	Yes
--tag	**@latest=Tag**	Gives **Tag** to the latest ready Revision	No
--untag	**Tag**	Removes **Tag** from Revision	Yes

CHAPTER 6. MONITORING OPENSHIFT SERVERLESS COMPONENTS

As an OpenShift Container Platform cluster administrator, you can deploy the OpenShift Container Platform monitoring stack and monitor the metrics of OpenShift Serverless components.

When using the OpenShift Serverless Operator, the required ServiceMonitor objects are created automatically for monitoring the deployed components.

OpenShift Serverless components, such as Knative Serving, expose metrics data. Administrators can monitor this data by using the OpenShift Container Platform web console.

6.1. CONFIGURING CLUSTER FOR APPLICATION MONITORING

Before application developers can monitor their applications, the human operator of the cluster needs to configure the cluster accordingly. This procedure shows how to.

Prerequisites

- You must log in as a user that belongs to a role with administrative privileges for the cluster.

Procedure

1. In the OpenShift Container Platform web console, navigate to the **Operators → OperatorHub** page and install the Prometheus Operator in the namespace where your application is.
2. Navigate to the **Operators → Installed Operators** page and install Prometheus, Alertmanager, Prometheus Rule, and Service Monitor in the same namespace.

6.2. VERIFYING AN OPENSHIFT CONTAINER PLATFORM MONITORING INSTALLATION FOR USE WITH KNATIVE SERVING

Manual configuration for monitoring by an administrator is not required, but you can carry out these steps to verify that monitoring is installed correctly.

Procedure

1. Verify that the ServiceMonitor objects are deployed.

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

2. Verify that the **openshift.io/cluster-monitoring=true** label has been added to the Knative Serving namespace:

```
$ 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
```

6.3. MONITORING KNATIVE SERVING USING THE OPENSHIFT CONTAINER PLATFORM MONITORING STACK

This section provides example instructions for the visualization of Knative Serving Pod autoscaling metrics by using the OpenShift Container Platform monitoring tools.

Prerequisites

- You must have the OpenShift Container Platform monitoring stack installed.

Procedure

1. Navigate to the OpenShift Container Platform web console and authenticate.
2. Navigate to **Monitoring** → **Metrics**.
3. Enter the **Expression** and select **Run queries**. To monitor Knative Serving autoscaler Pods, use this example expression.

```
autoscaler_actual_pods
```

   You will now see monitoring information for the Knative Serving autoscaler Pods in the console.

CHAPTER 7. USING METERING WITH OPENSHIFT SERVERLESS

As an OpenShift Container Platform cluster administrator, you can use metering to analyze what is happening in your OpenShift Serverless cluster.

For more information about metering on OpenShift Container Platform, see About metering.

7.1. INSTALLING METERING

For information about installing metering on OpenShift Container Platform, see Installing Metering.

7.2. DATASOURCES FOR KNATIVE SERVING METERING

The following **ReportDataSources** are examples of how Knative Serving can be used with OpenShift Container Platform metering.

7.2.1. Datasource for CPU usage in Knative Serving

This datasource provides the accumulated CPU seconds used per Knative service over the report time period.

YAML file

```
apiVersion: metering.openshift.io/v1
kind: ReportDataSource
metadata:
  name: knative-service-cpu-usage
spec:
  prometheusMetricsImporter:
    query: >
      sum
          by(namespace,
             label_serving_knative_dev_service,
             label_serving_knative_dev_revision)
          (

label_replace(rate(container_cpu_usage_seconds_total{container!="POD",container!="",pod!=""}
[1m]), "pod", "$1", "pod", "(.*)")
            *
            on(pod, namespace)
            group_left(label_serving_knative_dev_service, label_serving_knative_dev_revision)
            kube_pod_labels{label_serving_knative_dev_service!=""}
          )
```

7.2.2. Datasource for memory usage in Knative Serving

This datasource provides the average memory consumption per Knative service over the report time period.

YAML file

```
apiVersion: metering.openshift.io/v1
kind: ReportDataSource
metadata:
  name: knative-service-memory-usage
spec:
  prometheusMetricsImporter:
    query: >
      sum
          by(namespace,
             label_serving_knative_dev_service,
             label_serving_knative_dev_revision)
          (
            label_replace(container_memory_usage_bytes{container!="POD", container!="",pod!=""},
"pod", "$1", "pod", "(.*)")
            *
            on(pod, namespace)
            group_left(label_serving_knative_dev_service, label_serving_knative_dev_revision)
            kube_pod_labels{label_serving_knative_dev_service!=""}
          )
```

7.2.3. Applying Datasources for Knative Serving metering

You can apply the **ReportDataSources** by using the following command:

```
$ oc apply -f <datasource-name>.yaml
```

Example

```
$ oc apply -f knative-service-memory-usage.yaml
```

7.3. QUERIES FOR KNATIVE SERVING METERING

The following **ReportQuery** resources reference the example **DataSources** provided.

7.3.1. Query for CPU usage in Knative Serving

YAML file

```
apiVersion: metering.openshift.io/v1
kind: ReportQuery
metadata:
  name: knative-service-cpu-usage
spec:
  inputs:
  - name: ReportingStart
    type: time
  - name: ReportingEnd
    type: time
  - default: knative-service-cpu-usage
    name: KnativeServiceCpuUsageDataSource
    type: ReportDataSource
  columns:
```

```
  - name: period_start
    type: timestamp
    unit: date
  - name: period_end
    type: timestamp
    unit: date
  - name: namespace
    type: varchar
    unit: kubernetes_namespace
  - name: service
    type: varchar
  - name: data_start
    type: timestamp
    unit: date
  - name: data_end
    type: timestamp
    unit: date
  - name: service_cpu_seconds
    type: double
    unit: cpu_core_seconds
  query: |
    SELECT
      timestamp '{| default .Report.ReportingStart .Report.Inputs.ReportingStart| prestoTimestamp |}'
AS period_start,
      timestamp '{| default .Report.ReportingEnd .Report.Inputs.ReportingEnd | prestoTimestamp |}' AS
period_end,
      labels['namespace'] as project,
      labels['label_serving_knative_dev_service'] as service,
      min("timestamp") as data_start,
      max("timestamp") as data_end,
      sum(amount * "timeprecision") AS service_cpu_seconds
    FROM {| dataSourceTableName .Report.Inputs.KnativeServiceCpuUsageDataSource |}
    WHERE "timestamp" >= timestamp '{| default .Report.ReportingStart .Report.Inputs.ReportingStart
| prestoTimestamp |}'
    AND "timestamp" < timestamp '{| default .Report.ReportingEnd .Report.Inputs.ReportingEnd |
prestoTimestamp |}'
    GROUP BY labels['namespace'],labels['label_serving_knative_dev_service']
```


7.3.2. Query for memory usage in Knative Serving

YAML file


```
apiVersion: metering.openshift.io/v1
kind: ReportQuery
metadata:
  name: knative-service-memory-usage
spec:
  inputs:
  - name: ReportingStart
    type: time
  - name: ReportingEnd
    type: time
  - default: knative-service-memory-usage
    name: KnativeServiceMemoryUsageDataSource
    type: ReportDataSource
```

```
  columns:
  - name: period_start
    type: timestamp
    unit: date
  - name: period_end
    type: timestamp
    unit: date
  - name: namespace
    type: varchar
    unit: kubernetes_namespace
  - name: service
    type: varchar
  - name: data_start
    type: timestamp
    unit: date
  - name: data_end
    type: timestamp
    unit: date
  - name: service_usage_memory_byte_seconds
    type: double
    unit: byte_seconds
  query: |
    SELECT
      timestamp '{| default .Report.ReportingStart .Report.Inputs.ReportingStart| prestoTimestamp |}'
AS period_start,
      timestamp '{| default .Report.ReportingEnd .Report.Inputs.ReportingEnd | prestoTimestamp |}' AS
period_end,
      labels['namespace'] as project,
      labels['label_serving_knative_dev_service'] as service,
      min("timestamp") as data_start,
      max("timestamp") as data_end,
      sum(amount * "timeprecision") AS service_usage_memory_byte_seconds
    FROM {| dataSourceTableName .Report.Inputs.KnativeServiceMemoryUsageDataSource |}
    WHERE "timestamp" >= timestamp '{| default .Report.ReportingStart .Report.Inputs.ReportingStart
| prestoTimestamp |}'
    AND "timestamp" < timestamp '{| default .Report.ReportingEnd .Report.Inputs.ReportingEnd |
prestoTimestamp |}'
    GROUP BY labels['namespace'],labels['label_serving_knative_dev_service']
```

7.3.3. Applying Queries for Knative Serving metering

You can apply the **ReportQuery** by using the following command:

```
$ oc apply -f <query-name>.yaml
```

Example

```
$ oc apply -f knative-service-memory-usage.yaml
```

7.4. METERING REPORTS FOR KNATIVE SERVING

You can run metering reports against Knative Serving by creating **Report** resources. Before you run a report, you must modify the input parameter within the **Report** resource to specify the start and end dates of the reporting period.

YAML file

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: knative-service-cpu-usage
spec:
  reportingStart: '2019-06-01T00:00:00Z' (1)
  reportingEnd: '2019-06-30T23:59:59Z' (2)
  query: knative-service-cpu-usage (3)
runImmediately: true
```

1. Start date of the report, in ISO 8601 format.
2. End date of the report, in ISO 8601 format.
3. Either **knative-service-cpu-usage** for CPU usage report or **knative-service-memory-usage** for a memory usage report.

7.4.1. Running a metering report

Once you have provided the input parameters, you can run the report using the command:

```
$ oc apply -f <report-name>.yml
```

You can then check the report as shown in the following example:

```
$ kubectl get report

NAME                      QUERY                     SCHEDULE   RUNNING    FAILED   LAST REPORT
TIME       AGE
knative-service-cpu-usage   knative-service-cpu-usage            Finished            2019-06-
30T23:59:59Z   10h
```