OpenShift Container Platform 4.4

Serverless applications

OpenShift Serverless installation, usage, and release notes

Last Updated: 2020-06-20
OpenShift Container Platform 4.4 Serverless applications

OpenShift Serverless installation, usage, and release notes
Abstract

This document provides information on how to use OpenShift Serverless in OpenShift Container Platform
4.4.1. Knative Serving supported installation configuration options
4.4.1.1. Controller Custom Certs
4.4.1.2. High availability
4.5. UPGRADING OPENSshift SERVERLESS
4.5.1. Updating Knative services URL formats
4.5.2. Upgrading the Subscription Channel
4.6. REMOVING OPENSshift SERVERLESS
4.6.1. Uninstalling Knative Serving
4.6.2. Uninstalling Knative Eventing
4.6.3. Removing the OpenShift Serverless Operator
4.6.4. Deleting OpenShift Serverless CRDs
4.7. INSTALLING THE KNATIVE CLI (KN)
4.7.1. Installing the kn CLI using the OpenShift Container Platform web console
4.7.2. Installing the kn CLI for Linux using an RPM
4.7.3. Installing the kn CLI for Linux
4.7.4. Installing the kn CLI for macOS
4.7.5. Installing the kn CLI for Windows

CHAPTER 5. CREATING AND MANAGING SERVERLESS APPLICATIONS
5.1. SERVERLESS APPLICATIONS USING KNATIVE SERVICES
5.2. CREATING SERVERLESS APPLICATIONS USING THE OPENSshift CONTAINER PLATFORM WEB CONSOLE
5.2.1. Creating serverless applications using the Administrator perspective
5.2.2. Creating serverless applications using the Developer perspective
5.3. CREATING SERVERLESS APPLICATIONS USING THE KN CLI
5.4. CREATING SERVERLESS APPLICATIONS USING YAML
5.5. VERIFYING YOUR SERVERLESS APPLICATION DEPLOYMENT
5.6. INTERACTING WITH A SERVERLESS APPLICATION USING HTTP2 / GRPC

CHAPTER 6. HIGH AVAILABILITY ON OPENSshift SERVERLESS
6.1. CONFIGURING HIGH AVAILABILITY REPLICAS ON OPENSshift SERVERLESS

CHAPTER 7. TRACING REQUESTS USING JAEGER
7.1. CONFIGURING JAEGER FOR USE WITH OPENSshift SERVERLESS

CHAPTER 8. KNATIVE CLI
8.1. GETTING STARTED WITH KNATIVE CLI (KN)
8.1.1. Basic workflow using kn
8.1.2. Autoscaling workflow using kn
8.1.3. Traffic splitting using kn
8.1.3.1. Assigning tag revisions
8.1.3.2. Unassigning tag revisions
8.1.3.3. Traffic flag operation precedence
8.1.3.4. Traffic splitting flags

CHAPTER 9. KNATIVE SERVING
9.1. HOW KNATIVE SERVING WORKS
9.1.1. Knative Serving resources
9.1.2. Serverless applications using Knative services
9.1.3. Next steps
9.2. CONFIGURING KNATIVE SERVING AUTOSCAlING
9.2.1. Configuring concurrent requests for Knative Serving autoscaling
9.2.1.1. Configuring concurrent requests using the target annotation
9.2.1.2. Configuring concurrent requests using the containerConcurrency field
9.2.2. Configuring scale bounds Knative Serving autoscaling
9.3. CLUSTER LOGGING WITH OPENSHIFT SERVERLESS
  9.3.1. Cluster logging
  9.3.2. About deploying and configuring cluster logging
   9.3.2.1. Configuring and Tuning Cluster Logging
   9.3.2.2. Sample modified Cluster Logging Custom Resource
  9.3.3. Using cluster logging to find logs for Knative Serving components
  9.3.4. Using cluster logging to find logs for services deployed with Knative Serving
9.4. SPLITTING TRAFFIC BETWEEN REVISIONS
  9.4.1. Splitting traffic between revisions using the Developer perspective

CHAPTER 10. KNATIVE EVENTING .......................................................... 66
  10.1. HOW KNATIVE EVENTING WORKS
   10.1.1. Knative Eventing workflows
   10.1.2. Additional resources
  10.2. USING THE KN CLI TO LIST EVENT SOURCES AND EVENT SOURCE TYPES
   10.2.1. Listing available event source types using kn
   10.2.2. Listing available event sources using kn
   10.2.2.1. Listing event sources of a specific type only
  10.3. USING APIERVERSOURCE
   10.3.1. Using the ApiServerSource with the Knative CLI (kn)
   10.3.1.1. Deleting the ApiServerSource
   10.3.2. Using the ApiServerSource with the YAML method
   10.3.2.1. Deleting the ApiServerSource
  10.4. USING A PINGSOURCE
   10.4.1. Using a PingSource with the kn CLI
   10.4.1.1. Remove the PingSource
   10.4.2. Using a PingSource with YAML
   10.4.2.1. Remove the PingSource
  10.5. USING SINKBINDING
   10.5.1. Using SinkBinding with the Knative CLI (kn)
   10.5.2. Using SinkBinding with the YAML method
  10.6. USING TRIGGERS
   10.6.1. Creating a trigger using kn
   10.6.2. Listing triggers using kn
   10.6.3. Describing a trigger using kn
   10.6.4. Deleting a trigger using kn
   10.6.5. Updating a trigger using kn
   10.6.6. Filtering events using triggers
  10.7. USING BROKERS WITH KNATIVE EVENTING
   10.7.1. Creating a broker manually
   10.7.2. Creating a broker automatically using namespace annotation
   10.7.3. Deleting a broker that was created using namespace annotation
  10.8. USING CHANNELS
   10.8.1. Supported channel types
   10.8.2. Creating a channel with cluster default configuration

CHAPTER 11. USING METERING WITH OPENSHIFT SERVERLESS .................. 92
  11.1. INSTALLING METERING
  11.2. DATASOURCES FOR KNATIVE SERVING METERING
   11.2.1. Datasource for CPU usage in Knative Serving
   11.2.2. Datasource for memory usage in Knative Serving
   11.2.3. Applying Datasources for Knative Serving metering
11.3. QUERIES FOR KNATIVE SERVING METERING

11.3.1. Query for CPU usage in Knative Serving

11.3.2. Query for memory usage in Knative Serving

11.3.3. Applying Queries for Knative Serving metering

11.4. METERING REPORTS FOR KNATIVE SERVING

11.4.1. Running a metering report
CHAPTER 1. OPENSOURCE 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 Red Hat Customer Portal at http://access.redhat.com. Through the customer portal, you can:

- Search or browse through the Red Hat Knowledgebase of technical support articles about Red Hat products
- Submit a support case to Red Hat Global Support Services (GSS)
- Access other product documentation

If you have a suggestion for improving this guide or have found an error, please submit a Bugzilla report at http://bugzilla.redhat.com against Product for the Documentation component. Please provide specific details, such as the section number, guide name, and OpenShift Serverless version so we can easily locate the content.

IMPORTANT

Knative Eventing 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/.

1.2. RELEASE NOTES FOR RED HAT OPENSOURCE SERVERLESS 1.7.1

1.2.1. New features

- OpenShift Serverless now uses Knative Serving 0.13.3.
- OpenShift Serverless now uses Knative Serving Operator 0.13.3.
- OpenShift Serverless now uses Knative kn CLI 0.13.2.
- OpenShift Serverless uses Knative Eventing 0.13.0.
- OpenShift Serverless now uses Knative Eventing Operator 0.13.3.

1.2.2. Fixed issues

- In OpenShift Serverless 1.7.0, routes were reconciled continuously when this was not required. This bug is fixed in OpenShift Serverless 1.7.1.

1.3. RELEASE NOTES FOR RED HAT OPENSOURCE SERVERLESS 1.7.0
1.3.1. New features

- OpenShift Serverless 1.7.0 is now Generally Available (GA) on OpenShift Container Platform 4.3 and newer versions. In previous versions, OpenShift Serverless was a Technology Preview.

- OpenShift Serverless now uses Knative Serving 0.13.2.

- OpenShift Serverless now uses Knative Serving Operator 0.13.2.

- OpenShift Serverless now uses Knative kn CLI 0.13.2.

- Knative kn CLI downloads now support disconnected, or restricted network installations.

- Knative kn CLI libraries are now signed by Red Hat.

- Knative Eventing is now available as a Technology Preview with OpenShift Serverless. OpenShift Serverless uses Knative Eventing 0.13.2.

**IMPORTANT**

Before upgrading to the latest Serverless release, you must remove the community Knative Eventing Operator if you have previously installed it. Having the Knative Eventing Operator installed will prevent you from being able to install the latest Technology Preview version of Knative Eventing that is included with OpenShift Serverless 1.7.0.

- High availability (HA) is now enabled by default for the autoscaler-hpa, controller, activator, kourier-control, and kourier-gateway components.
  
  If you have installed a previous version of OpenShift Serverless, after the KnativeServing custom resource (CR) is updated, the deployment will default to a HA configuration with KnativeServing.spec.high-availability.replicas = 2.
  
  You can disable HA for these components by completing the procedure in the Configuring high availability components documentation.

- OpenShift Serverless now supports the trustedCA setting in OpenShift Container Platform’s cluster-wide proxy, and is now fully compatible with OpenShift Container Platform’s proxy settings.

- OpenShift Serverless now supports HTTPS using the wildcard certificate that is registered for OpenShift Container Platform routes. For more information on http and https on Knative Serving, see the documentation on Verifying your serverless application deployment.

1.3.2. Fixed issues

- In previous versions, requesting KnativeServing custom resources (CRs) without specifying an API group, for example, oc get knativeserving -n knative-serving, occasionally caused errors. This issue is fixed in OpenShift Serverless 1.7.0.

- In previous versions, the Knative Serving controller was not notified when a new service CA certificate was generated due to service CA certificate rotation. New revisions created after a service CA certificate rotation were failing with the error:

  Revision “foo-1” failed with message: Unable to fetch image “image-registry.openshift-image-registry.svc:5000/eap/eap-app”: failed to resolve image to digest: failed to fetch image information: Get https://image-registry.openshift-image-registry.svc:5000/v2/: x509: certificate signed by unknown authority.
1.3.3. Known issues

- When upgrading from OpenShift Serverless 1.6.0 to 1.7.0, support for HTTPS requires a change to the format of routes. Knative services created on OpenShift Serverless 1.6.0 are no longer reachable at the old format URLs. You must retrieve the new URL for each service after upgrading OpenShift Serverless. For more information, see the documentation on Upgrading OpenShift Serverless.

- If you are using Knative Eventing on an Azure cluster, it is possible that the `imc-dispatcher` pod may not start. This is due to the pod’s default `resources` settings. As a work-around, you can remove the `resources` settings.

- If you have 1000 Knative services on a cluster, and then perform a reinstall or upgrade of Knative Serving, there will be a delay when you create the first new service after KnativeServing becomes Ready. `3scale-kourier-control` reconciles all previous Knative services before processing the creation of a new service, which causes the new service to spend approximately 800 seconds in an `IngressNotConfigured` or `Unknown` state before the state will update to `Ready`.

1.4. RELEASE NOTES FOR RED HAT OPENSIGHT SERVERLESS TECHNOLOGY PREVIEW 1.6.0

1.4.1. 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.

### 1.4.2. 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 occurred in OpenShift Serverless 1.5.0, orphaned VirtualServices sometimes remained on the system. In OpenShift Serverless 1.6.0, orphaned VirtualServices are automatically removed.

### 1.4.3. 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
```

### 1.5. RELEASE NOTES FOR RED HAT OPENSOURCESERVERLESS

#### TECHNOLOGY PREVIEW 1.5.0

#### 1.5.1. New features

- OpenShift Serverless 1.5.0 is available on OpenShift Container Platform 4.3 and newer versions.
- OpenShift Serverless has been updated to use Knative Serving 0.12.1.
- OpenShift Serverless has been updated to use Knative `kn` CLI 0.12.0.
- OpenShift Serverless has been updated to use Knative Serving Operator 0.12.1.
- OpenShift Serverless ingress implementation has been 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.

#### 1.5.2. Fixed issues

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

#### 1.5.3. 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.

#### 1.6. 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.
CHAPTER 2. OPENSSHIFT SERVERLESS SUPPORT

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support.

The **must-gather** tool enables you to collect diagnostic information about your OpenShift Container Platform cluster, including data related to OpenShift Serverless.

For prompt support, supply diagnostic information for both OpenShift Container Platform and OpenShift Serverless.

### 2.1. ABOUT THE MUST-GATHER TOOL

The **oc adm must-gather** CLI command collects the information from your cluster that is most likely needed for debugging issues, such as:

- Resource definitions
- Audit logs
- Service logs

You can specify one or more images when you run the command by including the **--image** argument. When you specify an image, the tool collects data related to that feature or product.

When you run **oc adm must-gather**, a new Pod is created on the cluster. The data is collected on that Pod and saved in a new directory that starts with **must-gather.local**. This directory is created in the current working directory.

### 2.2. ABOUT COLLECTING OPENSSHIFT SERVERLESS DATA

You can use the **oc adm must-gather** CLI command to collect information about your cluster, including features and objects associated with OpenShift Serverless.

To collect OpenShift Serverless data with **must-gather**, you must specify the OpenShift Serverless image:

```
$ oc adm must-gather --image=registry.redhat.io/openshift-serverless-1/svls-must-gather-rhel8
```
CHAPTER 3. GETTING STARTED WITH OPENSHIFT SERVERLESS

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.

3.1. 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.

OpenShift Serverless on OpenShift Container Platform enables stateless 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.

3.2. SUPPORTED CONFIGURATIONS

The set of supported features, configurations, and integrations for OpenShift Serverless (current and past versions) are available at the Supported Configurations page.

3.3. OPENSHIFT SERVERLESS COMPONENTS

This section describes the components of OpenShift Serverless.

3.3.1. Knative Serving

Knative Serving on OpenShift Container Platform builds on Kubernetes and Kourier to support deploying and managing 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.

3.3.2. Knative Eventing

Knative Eventing on OpenShift Container Platform enables developers to more easily declare how components of their system communicate, using an event-driven architecture for serverless applications. Event-driven architecture is based on the concept of decoupled relationships between event producers and event consumers.

For more information about event-driven architecture, see What is event-driven architecture?
IMPORTANT

Knative Eventing 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/.

3.4. NEXT STEPS

- Install the OpenShift Serverless Operator on your OpenShift Container Platform cluster to get started with serverless applications.

- View the OpenShift Serverless release notes.
CHAPTER 4. INSTALLING OPENSSHIFT SERVERLESS

4.1. INSTALLING OPENSIGNED SERVERLESS

This guide walks cluster administrators through installing the OpenShift Serverless Operator to an OpenShift Container Platform cluster.

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

4.1.1. Cluster sizing requirements

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.

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.

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.

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.

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.

NOTE
The following limitations apply to all OpenShift Serverless deployments:

- Maximum number of Knative services: 1000
- Maximum number of Knative revisions: 1000

4.1.1.1. Additional requirements for advanced use-cases

For more advanced use-cases such as logging or metering on OpenShift Container Platform, you must deploy more resources. Recommended requirements for such use-cases are 24 CPUs and 96GB of memory.

If you have high availability (HA) enabled on your cluster, this requires between 0.5 - 1.5 cores and between 200MB - 2GB of memory for each replica of the Knative Serving control plane. HA is enabled.
for some Knative Serving components by default. You can disable HA by following the documentation on Configuring high availability replicas on OpenShift Serverless.

**IMPORTANT**

Before upgrading to the latest Serverless release, you must remove the community Knative Eventing operator if you have previously installed it. Having the Knative Eventing operator installed will prevent you from being able to install the latest Technology Preview version of Knative Eventing using the OpenShift Serverless Operator.

### 4.1.2. Installing the OpenShift Serverless Operator

This procedure describes how to install and subscribe to the OpenShift Serverless Operator from the OperatorHub using the OpenShift Container Platform web console.

**Procedure**

1. In the OpenShift Container Platform web console, navigate to the Operators ➔ OperatorHub page.

2. Scroll, or type the keyword **Serverless** into the Filter by keyword box to find the OpenShift Serverless Operator.

3. Review the information about the Operator and click **Install**.
4. On the Create Operator Subscription page:

   a. The **Installation Mode** is **All namespaces on the cluster (default)**. This mode installs the Operator in the default **openshift-operators** namespace to watch and be made available to all namespaces in the cluster.

   b. The **Installed Namespace** will be **openshift-operators**.

   c. Select the **4.4** channel as the **Update Channel**. The **4.4** channel will enable installation of the latest stable release of the OpenShift Serverless Operator.

   d. Select **Automatic** or **Manual** approval strategy.

5. Click **Subscribe** to make the Operator available to the selected namespaces on this OpenShift Container Platform cluster.

6. From the **Catalog → Operator Management** page, you can monitor the OpenShift Serverless Operator subscription’s installation and upgrade progress.
a. If you selected a **Manual** approval strategy, the subscription’s upgrade status will remain **Upgrading** until you review and approve its install plan. After approving on the **Install Plan** page, the subscription upgrade status moves to **Up to date**.

b. If you selected an **Automatic** approval strategy, the upgrade status should resolve to **Up to date** without intervention.

**Verification steps**

After the Subscription’s upgrade status is **Up to date**, select **Catalog → Installed Operators** to verify that the OpenShift Serverless Operator eventually shows up and its **Status** ultimately resolves to **InstallSucceeded** in the relevant namespace.

If it does not:

1. Switch to the **Catalog → Operator Management** page and inspect the **Operator Subscriptions** and **Install Plans** tabs for any failure or errors under **Status**.

2. Check the logs in any pods in the `openshift-operators` project on the **Workloads → Pods** page that are reporting issues to troubleshoot further.

**Additional resources**

- For more information on installing Operators, see the OpenShift Container Platform documentation on [Adding Operators to a cluster](#).

**4.1.3. Next steps**

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

- After the OpenShift Serverless Operator is installed, you can install the Knative Eventing component. See the documentation on [Installing Knative Eventing](#).

**4.2. INSTALLING KNATIVE SERVING**

After you install the OpenShift Serverless Operator, you can install Knative Serving by following the procedures described in this guide.
This guide provides information about installing Knative Serving using the default settings. However, you can configure more advanced settings in the KnativeServing custom resource definition.

For more information about configuration options for the KnativeServing custom resource definition, see Advanced installation configuration options.

4.2.1. Creating the knative-serving namespace

When you create the knative-serving namespace, a knative-serving project will also be created.

**IMPORTANT**

You must complete this procedure before installing Knative Serving.

If the KnativeServing object created during Knative Serving’s installation is not created in the knative-serving namespace, it will be ignored.

Prerequisites

- An OpenShift Container Platform account with cluster administrator access
- Installed OpenShift Serverless Operator

4.2.1.1. Creating the knative-serving namespace using the web console

Procedure

1. In the OpenShift Container Platform web console, navigate to Administration → Namespaces.

2. Enter knative-serving as the Name for the project. The other fields are optional.
3. Click Create.

4.2.1.2. Creating the knative-serving namespace using the CLI

Procedure

1. Create the knative-serving namespace by entering:

   ```
   $ oc create namespace knative-serving
   ```

Prerequisites

- An OpenShift Container Platform account with cluster administrator access.
- Installed OpenShift Serverless Operator.
- Created the knative-serving namespace.

4.2.2. Installing Knative Serving using the web console

Procedure

1. In the Administrator perspective of the OpenShift Container Platform web console, navigate to Operators → Installed Operators.

2. Check that the Project dropdown at the top of the page is set to Project: knative-serving.

3. Click Knative Serving in the list of Provided APIs for the OpenShift Serverless Operator to go to the Knative Serving tab.
4. Click the Create Knative Serving button.

5. In the Create Knative Serving page, you can install Knative Serving using the default settings by clicking Create.

   You can also modify settings for the Knative Serving installation by editing the KnativeServing object using either the form provided, or by editing the YAML.

   - Using the form is recommended for simpler configurations that do not require full control of KnativeServing object creation.

   - Editing the YAML is recommended for more complex configurations that require full control of KnativeServing object creation. You can access the YAML by clicking the edit YAML link in the top right of the Create Knative Serving page.

   After you complete the form, or have finished modifying the YAML, click Create.

   **NOTE**

   For more information about configuration options for the KnativeServing custom resource definition, see the documentation on Advanced installation configuration options.
6. After you have installed Knative Serving, the **KnativeServing** object is created, and you will be automatically directed to the **Knative Serving** tab.

You will see **knative-serving** in the list of resources.

**Verification steps**

1. Click on **knative-serving** in the **Knative Serving** tab.

2. You will be automatically directed to the **Knative Serving Overview** page.
3. Scroll down to look at the list of Conditions.

4. You should see a list of conditions with a status of True, as shown in the example image.

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Status</th>
<th>Updated</th>
<th>Reason</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>DependenciesInstalled</td>
<td>True</td>
<td>3 minutes ago</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DeploymentsAvailable</td>
<td>True</td>
<td>3 minutes ago</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>InstallSuccess</td>
<td>True</td>
<td>3 minutes ago</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Ready</td>
<td>True</td>
<td>3 minutes ago</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**NOTE**

It may take a few seconds for the Knative Serving resources to be created. You can check their status in the Resources tab.

5. If the conditions have a status of Unknown or False, wait a few moments and then check again after you have confirmed that the resources have been created.

### 4.2.3. Installing Knative Serving using YAML

**Procedure**
1. Create a file named `serving.yaml`.

2. Copy the following sample YAML into `serving.yaml`:

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

3. Apply the `serving.yaml` file:

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

**Verification steps**

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

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

   The output should be similar to:

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

   **NOTE**

   It may take a few seconds for the Knative Serving resources to be created.

2. If the conditions have a status of **Unknown** or **False**, wait a few moments and then check again after you have confirmed that the resources have been created.

3. Check that the Knative Serving resources have been created by entering:

   ```bash
   $ oc get pods -n knative-serving
   ```

   The output should look similar to:

   ```
   NAME                               READY   STATUS    RESTARTS   AGE
   activator-5c596cf8d6-5i86c         1/1     Running   0          9m37s
   activator-5c596cf8d6-gkn5k         1/1     Running   0          9m22s
   autoscaler-5854f586f6-gj597        1/1     Running   0          9m36s
   autoscaler-hpa-78665569b8-qmlmn    1/1     Running   0          9m26s
   autoscaler-hpa-78665569b8-tqww    1/1     Running   0          9m26s
   controller-7fd565f49-9gxz5         1/1     Running   0          9m32s
   controller-7fd565f49-pncv5         1/1     Running   0          9m14s
   kn-cli-downloads-8c65d4cbf-mt4t7   1/1     Running   0          9m42s
   webhook-5c7d878c7c-n267]          1/1     Running   0          9m35s
   ```
4.2.4. Next steps

- For cloud events functionality on OpenShift Serverless, you can install the Knative Eventing component. See the documentation on Installing Knative Eventing.

- Install the Knative CLI to use kn commands with Knative Serving. For example, kn service commands. See the documentation on Installing the Knative CLI (kn).

4.3. INSTALLING KNATIVE EVENTING

After you install the OpenShift Serverless Operator, you can install Knative Eventing by following the procedures described in this guide.

**IMPORTANT**

Knative Eventing 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/.

This guide provides information about installing Knative Eventing using the default settings.

4.3.1. Creating the knative-eventing namespace

When you create the knative-eventing namespace, a knative-eventing project will also be created.

**IMPORTANT**

You must complete this procedure before installing Knative Eventing.

If the KnativeEventing object created during Knative Eventing’s installation is not created in the knative-eventing namespace, it will be ignored.

**Prerequisites**

- An OpenShift Container Platform account with cluster administrator access
- Installed OpenShift Serverless Operator

4.3.1.1. Creating the knative-eventing namespace using the web console

**Procedure**

1. In the OpenShift Container Platform web console, navigate to Administration → Namespaces.
2. Click Create Namespace.
3. Enter `knative-eventing` as the Name for the project. The other fields are optional.

4. Click Create.

### 4.3.1.2. Creating the `knative-eventing` namespace using the CLI

**Procedure**

1. Create the `knative-eventing` namespace by entering:

   ```bash
   $ oc create namespace knative-eventing
   ```

**Prerequisites**

- An OpenShift Container Platform account with cluster administrator access
- Installed OpenShift Serverless Operator
- Created the `knative-eventing` namespace
4.3.2. Installing Knative Eventing using the web console

Procedure

1. In the Administrator perspective of the OpenShift Container Platform web console, navigate to Operators → Installed Operators.

2. Check that the Project dropdown at the top of the page is set to Project: knative-eventing.

3. Click Knative Eventing in the list of Provided APIs for the OpenShift Serverless Operator to go to the Knative Eventing tab.

4. Click the Create Knative Eventing button.

5. In the Create Knative Eventing page, you can choose to configure the KnativeEventing object by using either the default form provided, or by editing the YAML.

   - Using the form is recommended for simpler configurations that do not require full control of KnativeEventing object creation.
     Optional. If you are configuring the KnativeEventing object using the form, make any changes that you want to implement for your Knative Eventing deployment.

6. Click Create.

   - Editing the YAML is recommended for more complex configurations that require full control of KnativeEventing object creation. You can access the YAML by clicking the edit YAML link in the top right of the Create Knative Eventing page.
Optional. If you are configuring the **KnativeEventing** object by editing the YAML, make any changes to the YAML that you want to implement for your Knative Eventing deployment.

7. Click **Create**.

8. After you have installed Knative Eventing, the **KnativeEventing** object is created, and you will be automatically directed to the **Knative Eventing** tab.

You will see **knative-eventing** in the list of resources.

**Verification steps**

1. Click on **knative-eventing** in the **Knative Eventing** tab.

2. You will be automatically directed to the **Knative Eventing Overview** page.
3. Scroll down to look at the list of **Conditions**.

4. You should see a list of conditions with a status of **True**, as shown in the example image.

   ![Knative Eventing Overview](image)

   **Knative Eventing Overview**
   - **Name**: knative-eventing
   - **Version**: 0.13
   - **Namespace**: knative-eventing
   - **Labels**: No labels
   - **Annotations**: No annotations
   - **Created At**: 2 minutes ago
   - **Owner**: No owner

   **Conditions**
   - **Type**: InstalledSucceeded
   - **Status**: True
   - **Updated**: 2 minutes ago
   - **Reason**: -
   - **Message**: -

   ![Knative Eventing Conditions](image)

   **NOTE**
   
   It may take a few seconds for the Knative Eventing resources to be created. You can check their status in the **Resources** tab.

5. If the conditions have a status of **Unknown** or **False**, wait a few moments and then check again after you have confirmed that the resources have been created.
4.3.3. Installing Knative Eventing using YAML

**Procedure**

1. Create a file named `eventing.yaml`.
2. Copy the following sample YAML into `eventing.yaml`:

   ```yaml
   apiVersion: operator.knative.dev/v1alpha1
   kind: KnativeEventing
   metadata:
     name: knative-eventing
     namespace: knative-eventing
   ```
3. Optional. Make any changes to the YAML that you want to implement for your Knative Eventing deployment.
4. Apply the `eventing.yaml` file by entering:

   ```bash
   $ oc apply -f eventing.yaml
   ```

**Verification steps**

1. To verify the installation is complete, enter:

   ```bash
   $ oc get knativeeventing.operator.knative.dev/knative-eventing -n knative-eventing --template='{{range .status.conditions}}{{printf "%s=%s\n" .type .status}}{{end}}'
   ```
   The output should be similar to:

   ```plaintext
   InstallSucceeded=True
   Ready=True
   ```

   **NOTE**
   It may take a few seconds for the Knative Eventing resources to be created.

2. If the conditions have a status of **Unknown** or **False**, wait a few moments and then check again after you have confirmed that the resources have been created.

3. Check that the Knative Eventing resources have been created by entering:

   ```bash
   $ oc get pods -n knative-eventing
   ```
   The output should look similar to:

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>broker-controller-58765d9d49-g9zp6</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>7m21s</td>
</tr>
<tr>
<td>eventing-controller-65fdd66b54-jw7bh</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>7m31s</td>
</tr>
<tr>
<td>eventing-webhook-57fd74b5bd-kvhlz</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>7m31s</td>
</tr>
<tr>
<td>imc-controller-5b75d458fc-ptvm2</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>7m19s</td>
</tr>
<tr>
<td>imc-dispatcher-64f6d5fccb-kkc4c</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>7m18s</td>
</tr>
</tbody>
</table>
4.3.4. Next steps

- For services and serving functionality on OpenShift Serverless, you can install the Knative Serving component. See the documentation on Installing Knative Serving.

- Install the Knative CLI to use `kn` commands with Knative Eventing. For example, `kn source` commands. See the documentation on Installing the Knative CLI (`kn`).

4.4. ADVANCED INSTALLATION CONFIGURATION OPTIONS

This guide provides information for cluster administrators about advanced installation configuration options for Knative Serving and Knative Eventing.

The YAML configuration is shown by default when you are creating the Knative Serving or Knative Eventing custom resource definitions. However, you can access the form view by selecting Edit Form in the top right corner of the Create Knative Serving or Create Knative Eventing pages.

**IMPORTANT**

Do not modify any YAML contained inside the `config` field. Some of the configuration values in this field are injected by the OpenShift Serverless Operator, and modifying them will cause your deployment to become unsupported.

4.4.1. Knative Serving supported installation configuration options

4.4.1.1. Controller Custom Certs

If your registry uses a self-signed certificate, you must enable tag-to-digest resolution by creating a ConfigMap or Secret. The OpenShift Serverless Operator then automatically configures Knative Serving controller access to the registry.

To enable tag-to-digest resolution, the Knative Serving controller requires access to the container registry.
The ConfigMap or Secret must reside in the same namespace as the Knative Serving custom resource definition.

The following example triggers the OpenShift Serverless Operator to:

1. Create and mount a volume containing the certificate in the controller.
2. Set the required environment variable properly.

**Example YAML**

```yaml
apiVersion: operator.knative.dev/v1alpha1
description: KnativeServing
metadata:
  name: knative-serving
namespace: knative-serving
spec:
  controller-custom-certs:
    name: certs
    type: ConfigMap
```

The following example uses a certificate in a ConfigMap named `certs` in the `knative-serving` namespace.

You can also configure controller custom certs using the form view. The supported types are **ConfigMap** and **Secret**.

If no controller custom cert is specified, this defaults to the `config-service-ca` ConfigMap.

**Example default YAML**

```yaml
spec:
  controller-custom-certs:
    name: config-service-ca
    type: ConfigMap
```

### 4.4.1.2. High availability

High availability (HA) defaults to 2 replicas per controller if no number of replicas is specified.

You can set this to 1 to disable HA, or add more replicas by setting a higher integer.

For more information about configuring HA, see [High availability on OpenShift Serverless](#).

**Example YAML**

```yaml
spec:
  high-availability:
    replicas: 2
```

### 4.5. UPGRADING OPENSILFHT SERVERLESS
If you installed a previous version of OpenShift Serverless, follow the instructions in this guide to upgrade to the latest version.

**IMPORTANT**

Before upgrading to the latest Serverless release, you must remove the community Knative Eventing operator if you have previously installed it. Having the Knative Eventing operator installed will prevent you from being able to install the latest Technology Preview version of Knative Eventing.

### 4.5.1. Updating Knative services URL formats

When upgrading from older versions of OpenShift Serverless to 1.7.0, support for HTTPS requires a change to the format of routes. Knative services created on OpenShift Serverless 1.6.0 or older versions are no longer reachable at the old format URLs. You must retrieve the new URL for each service after upgrading OpenShift Serverless.

For more information on retrieving Knative services URLs, see [Verifying your serverless application deployment](#).

### 4.5.2. Upgrading the Subscription Channel

To upgrade to the latest version of OpenShift Serverless on OpenShift Container Platform 4.4, you must update the channel to **4.4**.

If you are upgrading from OpenShift Serverless version 1.5.0, or earlier, to version 1.7.0, you must complete the following steps:

- Upgrade to OpenShift Serverless version 1.5.0, by selecting the **techpreview** channel.
- After you have upgraded to 1.5.0, upgrade to 1.6.0 by selecting the **preview-4.3** channel.
- Finally, after you have upgraded to 1.6.0, upgrade to the latest version by selecting the **4.4** channel.

**IMPORTANT**

After each channel change, wait for the pods in the **knative-serving** namespace to get upgraded before changing the channel again.

**Prerequisites**

- You have installed a previous version of OpenShift Serverless Operator, and have selected Automatic updates during the installation process.

**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.

- 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.
4. Click Subscription → Channel.
5. In the Change Subscription Update Channel window, select 4.4, 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:

   ```bash
   $ 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:

   ```bash
   $ 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.

4.6. 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.

4.6.1. 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:

   ```bash
   $ 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:

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

### 4.6.2. Uninstalling Knative Eventing

To uninstall Knative Eventing, you must remove its custom resource and delete the `knative-eventing` namespace.

Procedure

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

   ```bash
   $ oc delete knativeeventings.operator.knative.dev knative-eventing -n knative-eventing
   ```

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

   ```bash
   $ oc delete namespace knative-eventing
   ```

### 4.6.3. 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.

### 4.6.4. 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:

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

### 4.7. INSTALLING THE KNATIVE CLI (KN)

---

CHAPTER 4. INSTALLING OPENSHIFT SERVERLESS

35
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.

4.7.1. 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

4.7.2. 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>
# 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

## 4.7.3. 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
``` 

## 4.7.4. 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
   ```

## 4.7.5. 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
   ```
CHAPTER 5. CREATING AND MANAGING SERVERLESS APPLICATIONS

5.1. SERVERLESS APPLICATIONS USING KNATIVE SERVICES

To deploy a serverless application using OpenShift Serverless, you must create a Knative service. Knative services are Kubernetes services, defined by a route and a configuration, and contained in a YAML file.

Example Knative service YAML

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: hello
namespace: default
spec:
template:
spec:
  containers:
  - image: docker.io/openshift/hello-openshift
    env:
      - name: RESPONSE
        value: "Hello Serverless!"
```

1. The name of the application.
2. The namespace the application will use.
3. The image of the application.
4. The environment variable printed out by the sample application.

You can create a serverless application by using one of the following methods:

- Create a Knative service from the OpenShift Container Platform web console.
- Create a Knative service using the `kn` CLI.
- Create and apply a YAML file.

5.2. CREATING SERVERLESS APPLICATIONS USING THE OPENSHIFT CONTAINER PLATFORM WEB CONSOLE

You can create a serverless application using either the Developer or Administrator perspective in the OpenShift Container Platform web console.

5.2.1. Creating serverless applications using the Administrator perspective

Prerequisites

To create serverless applications using the Administrator perspective, ensure that you have completed the following steps.
The OpenShift Serverless Operator and Knative Serving are installed.

You have logged in to the web console and are in the Administrator perspective.

Procedure

1. Navigate to the Serverless → Services page.

2. Click Create Service.

3. Manually enter YAML or JSON definitions, or by dragging and dropping a file into the editor.

4. Click Create.

5.2.2. Creating serverless applications using the Developer perspective

For more information about creating applications using the Developer perspective in OpenShift Container Platform, see the documentation on Creating applications using the Developer perspective.

5.3. CREATING SERVERLESS APPLICATIONS USING THE KN CLI

The following procedure describes how you can create a basic serverless application using the kn CLI.

Prerequisites

- OpenShift Serverless Operator and Knative Serving are installed on your cluster.
You have installed kn CLI.

Procedure

1. Create the Knative service by entering the following command:

   
   $ kn service create <SERVICE-NAME> --image <IMAGE> --env <KEY=VALUE>

Example command

   
   $ kn service create hello --image docker.io/openshift/hello-openshift --env RESPONSE="Hello Serverless!"

Example output

   
   Creating service 'hello' in namespace 'default':
   
   0.271s The Route is still working to reflect the latest desired specification.
   0.580s Configuration "hello" is waiting for a Revision to become ready.
   3.857s ...
   3.861s Ingress has not yet been reconciled.
   4.270s Ready to serve.
   
   Service 'hello' created with latest revision 'hello-bxshg-1' and URL:
   http://hello-default.apps-crc.testing

5.4. CREATING SERVERLESS APPLICATIONS USING YAML

To create a serverless application, you can create a YAML file and apply it using oc apply.

You can create a YAML file by copying the following example:

Example Knative service YAML

   
   apiVersion: serving.knative.dev/v1
   kind: Service
   metadata:
     name: hello
     namespace: default
   spec:
     template:
       spec:
         containers:
           - image: docker.io/openshift/hello-openshift
             env:
               - name: RESPONSE
                 value: "Hello Serverless!"

In this example, the YAML file is named hello-service.yaml.

Procedure
1. Navigate to the directory where the `hello-service.yaml` file is contained, and deploy the application by applying the YAML file:

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

After the 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.

5.5. VERIFYING YOUR SERVERLESS APPLICATION DEPLOYMENT

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

**NOTE**

OpenShift Serverless supports the use of both HTTP and HTTPS URLs, however the output from `oc get ksvc <SERVICE-NAME>` will always print URLs using the `http://` format.

**Procedure**

1. Find the application URL by entering:

```
$ oc get ksvc hello
```

**Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>URL</th>
<th>LATESTCREATED</th>
<th>LATESTREADY</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td><a href="http://hello-default.example.com">http://hello-default.example.com</a></td>
<td>hello-4wsd2</td>
<td>hello-4wsd2</td>
</tr>
<tr>
<td></td>
<td><strong>READY REASON</strong></td>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>

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

**Example HTTP request**

```
$ curl http://hello-default.example.com
Hello Serverless!
```

**Example HTTPS request**

```
$ curl https://hello-default.example.com
Hello Serverless!
```

3. Optional. If you receive an error relating to a self-signed certificate in the certificate chain, you can add the `--insecure` flag to the curl command to ignore the error.
IMPORTANT

Self-signed certificates must not be used in a production deployment. This method is only for testing purposes.

Example

```bash
$ curl https://hello-default.example.com --insecure
Hello Serverless!
```

4. Optional. If your OpenShift Container Platform cluster is configured with a certificate that is signed by a certificate authority (CA) but not yet globally configured for your system, you can specify this with the curl command. The path to the certificate can be passed to the curl command by using the `--cacert` flag.

Example

```bash
$ curl https://hello-default.example.com --cacert <file>
Hello Serverless!
```

5.6. 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:

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

   The output will be similar to this example:

   ```
   NAME                   TYPE           CLUSTER-IP      EXTERNAL-IP
   PORT(S)                AGE
   kourier   LoadBalancer   172.30.51.103   a83e86291bcdd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com   80:31380/TCP,443:31390/TCP   67m
   ```

   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: hello-default.example.com" a83e86291bcdd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com
Hello Serverless!
```

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:

```go
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("hello-default.example.com:80"),
    grpc.WithInsecure(),
)
```
CHAPTER 6. HIGH AVAILABILITY ON OPENSSHIFT SERVERLESS

High availability (HA) is a standard feature of Kubernetes APIs that helps to ensure that APIs stay operational if a disruption occurs. In an HA deployment, if an active controller crashes or is deleted, another controller is available to take over processing of the APIs that were being serviced by the controller that is now unavailable.

HA in OpenShift Serverless is available through leader election, which is enabled by default after the Knative Serving control plane is installed.

When using a leader election HA pattern, instances of controllers are already scheduled and running inside the cluster before they are required. These controller instances compete to use a shared resource, known as the leader election lock. The instance of the controller that has access to the leader election lock resource at any given time is referred to as the leader.

6.1. CONFIGURING HIGH AVAILABILITY REPLICAS ON OPENSHEET SERVERLESS

High availability (HA) functionality is available by default on OpenShift Serverless for the autoscaler-hpa, controller, activator, kourier-control, and kourier-gateway components. These components are configured with two replicas by default.

You modify the number of replicas that are created per controller by changing the configuration of KnativeServing.spec.highAvailability in the KnativeServing custom resource definition.

Prerequisites

- An OpenShift Container Platform account with cluster administrator access.
- Installed the OpenShift Serverless Operator and Knative Serving.

Procedure

1. In the OpenShift Container Platform web console Administrator perspective, navigate to OperatorHub → Installed Operators.

2. Select the knative-serving namespace.

3. Click Knative Serving in the list of Provided APIs for the OpenShift Serverless Operator to go to the Knative Serving tab.
4. Click `knative-serving`, then go to the YAML tab in the `knative-serving` page.

5. Edit the custom resource definition YAML:

```
Example YAML

spec:
  high-availability:
    replicas: 3
```

**IMPORTANT**

Do not modify any YAML contained inside the `config` field. Some of the configuration values in this field are injected by the OpenShift Serverless Operator, and modifying them will cause your deployment to become unsupported.

- The default `replicas` value is 2.
Changing the value to 1 will disable HA, or you can increase the number of replicas as required. The example configuration shown specifies a replica count of 3 for all HA controllers.
CHAPTER 7. TRACING REQUESTS USING JAEGER

Using Jaeger with OpenShift Serverless allows you to enable distributed tracing for your serverless applications on OpenShift Container Platform.

Distributed tracing records the path of a request through the various services that make up an application.

It is used to tie information about different units of work together, to understand a whole chain of events in a distributed transaction. The units of work might be executed in different processes or hosts.

Developers can visualize call flows in large architectures with distributed tracing. which is useful for understanding serialization, parallelism, and sources of latency.

For more information about Jaeger, see Jaeger architecture and Installing Jaeger.

7.1. CONFIGURING JAEGER FOR USE WITH OPENSSHIFT SERVERLESS

Prerequisites

To configure Jaeger for use with OpenShift Serverless, you will need:

- Cluster administrator permissions on an OpenShift Container Platform cluster.
- A current installation of the Jaeger Operator.

Procedure

1. Create and apply the Jaeger custom resource:

```bash
$ cat <<EOF | oc apply -f -
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger
  namespace: default
EOF
```

2. Enable tracing for Knative Serving, by editing the KnativeServing resource and adding a YAML configuration for tracing.

Tracing YAML example

```yaml
apiVersion: operator.knative.dev/v1alpha1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  config:
    tracing:
      sample-rate: "0.1"
```

backend: zipkin
zipkin-endpoint: http://jaeger-collector.default.svc.cluster.local:9411/api/v2/spans
debug: "false"

1. The **sample-rate** defines sampling probability. Using **sample-rate: "0.1"** means that 1 in 10 traces will be sampled.

2. **backend** must be set to **zipkin**.

3. The **zipkin-endpoint** must point to your **jaeger-collector** service endpoint. To get this endpoint, substitute the namespace where the Jaeger custom resource is applied.

4. Debugging should be set to **false**. Enabling debug mode by setting **debug: "true"** allows all spans to be sent to the server, bypassing sampling.

**Verification steps**

Access the Jaeger web console to see tracing data. You can access the Jaeger web console by using the **jaeger** route.

1. Get the **jaeger** route’s hostname:

   ```
   $ oc get route jaeger
   NAME     HOST/PORT                         PATH   SERVICES       PORT    TERMINATION
   WILDCARD
   jaeger   jaeger-default.apps.example.com          jaeger-query   <all>   reencrypt     None
   ```

2. Open the endpoint address in your browser to view the console.
CHAPTER 8. KNATIVE CLI

8.1. GETTING STARTED WITH KNATIVE CLI (KN)

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.

8.1.1. Basic workflow using kn

The following basic workflow deploys a simple `hello` service that reads the environment variable `RESPONSE` and prints its output.

You can use this guide as a reference to perform create, read, update, and delete (CRUD) operations on a service.

Procedure

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

   ```
   $ kn service create hello --image docker.io/openshift/hello-openshift --env RESPONSE="Hello Serverless!"
   ```

   **Example output**

   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
   ```

   **Example output**

   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
   ```

   **Example output**

   -
Hello Serverless!

4. Update the service:

```bash
$ kn service update hello --env RESPONSE="Hello OpenShift!"
```

**Example output**

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 **RESPONSE** is now set to "Hello OpenShift!".

5. Describe the service.

```bash
$ kn service describe hello
```

**Example output**

<table>
<thead>
<tr>
<th>Name</th>
<th>hello</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace</td>
<td>default</td>
</tr>
<tr>
<td>Age</td>
<td>13m</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hello-default.apps-crc.testing">http://hello-default.apps-crc.testing</a></td>
</tr>
</tbody>
</table>

Revisions:

- 100% @latest (hello-dghll-2) [2] (1m)
  - Image: docker.io/openshift/hello-openshift (pinned to 5ea96b)

Conditions:

<table>
<thead>
<tr>
<th>OK TYPE</th>
<th>AGE REASON</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ Ready</td>
<td>1m</td>
</tr>
<tr>
<td>++ Configurations</td>
<td>Ready</td>
</tr>
<tr>
<td>++ RoutesReady</td>
<td>1m</td>
</tr>
</tbody>
</table>

6. Delete the service:

```bash
$ kn service delete hello
```

**Example output**

Service 'hello' successfully deleted in namespace 'default'.

Verify that the **hello** service is deleted by attempting to **list** it:

```bash
$ kn service list hello
```
Example output

No services found.

8.1.2. 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.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--concurrency-limit int</code></td>
<td>Hard limit of concurrent requests to be processed by a single replica.</td>
</tr>
<tr>
<td><code>--concurrency-target int</code></td>
<td>Recommendation for when to scale up based on the concurrent number of incoming requests. Defaults to <code>--concurrency-limit</code>.</td>
</tr>
<tr>
<td><code>--max-scale int</code></td>
<td>Maximum number of replicas.</td>
</tr>
<tr>
<td><code>--min-scale int</code></td>
<td>Minimum number of replicas.</td>
</tr>
</tbody>
</table>

8.1.3. 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 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:

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

8.1.3.1. 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:

  ```bash
  $ 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.

8.1.3.2. 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.

8.1.3.3. 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.

### 8.1.3.4. 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.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value(s)</th>
<th>Operation</th>
<th>Repetition</th>
</tr>
</thead>
<tbody>
<tr>
<td>--traffic</td>
<td>RevisionName= Percent</td>
<td>Gives Percent traffic to RevisionName</td>
<td>Yes</td>
</tr>
<tr>
<td>--traffic</td>
<td>Tag=Percent</td>
<td>Gives Percent traffic to the Revision having Tag</td>
<td>Yes</td>
</tr>
<tr>
<td>--traffic</td>
<td>@latest=Percent</td>
<td>Gives Percent traffic to the latest ready Revision</td>
<td>No</td>
</tr>
<tr>
<td>--tag</td>
<td>RevisionName= Tag</td>
<td>Gives Tag to RevisionName</td>
<td>Yes</td>
</tr>
<tr>
<td>--tag</td>
<td>@latest=Tag</td>
<td>Gives Tag to the latest ready Revision</td>
<td>No</td>
</tr>
<tr>
<td>--untag</td>
<td>Tag</td>
<td>Removes Tag from Revision</td>
<td>Yes</td>
</tr>
</tbody>
</table>
9.1. HOW KNATIVE SERVING WORKS

Knative Serving on OpenShift Container Platform builds on Kubernetes and Kourier to support deploying and managing 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.

9.1.1. Knative Serving resources

The resources described in this section are required for Knative Serving 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.

9.1.2. Serverless applications using Knative services

To deploy a serverless application using OpenShift Serverless, you must create a Knative service. Knative services are Kubernetes services, defined by a route and a configuration, and contained in a YAML file.

Example Knative service YAML

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: hello
```

56
You can create a serverless application by using one of the following methods:

- Create a Knative service from the OpenShift Container Platform web console.
- Create a Knative service using the `kn` CLI.
- Create and apply a YAML file.

9.1.3. **Next steps**

- Create a serverless application. For more information, see the documentation on Creating and managing serverless applications.

## 9.2. 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.

### 9.2.1. 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:

```yaml
namespace: default
spec:
template:
spec:
containers:
- image: docker.io/openshift/hello-openshift
  env:
  - name: RESPONSE
    value: "Hello Serverless!"
```
Here is an example of containerConcurrency being used in a revision template:

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: myapp
spec:
template:
  metadata:
    annotations:
      autoscaling.knative.dev/target: 50
spec:
  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.

9.2.1.1. 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.

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

9.2.1.2. 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.

### 9.2.2. 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:

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

Using these annotations in the revision template will propagate this configuration 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.

### 9.3. CLUSTER LOGGING WITH OPENSSHIFT SERVERLESS

#### 9.3.1. 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.

### 9.3.2. 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.

### 9.3.2.1. 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:

```yaml
spec:
  logStore:
    elasticsearch:
      resources:
        limits:
          cpu: 1
          memory: 16Gi
        requests:
          cpu: 1
          memory: 16Gi
    type: "elasticsearch"
  collection:
    fluentd:
      resources:
```

OpenShift Container Platform 4.4 Serverless applications
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.

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.
**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**.

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

**9.3.2.2. 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**

```yaml
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:
```
9.3.3. 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.

9.3.4. 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.

9.4. SPLITTING TRAFFIC BETWEEN REVISIONS

9.4.1. 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.
10.1. HOW KNATIVE EVENTING WORKS

IMPORTANT

Knative Eventing 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/.

Knative Eventing on OpenShift Container Platform enables developers to more easily declare how components of their system communicate, using an event-driven architecture for serverless applications.

Event-driven architecture is based on the concept of decoupled relationships between event producers and event consumers.

For more information about event-driven architecture, see What is event-driven architecture?

10.1.1. Knative Eventing workflows

In a Knative Eventing workflow, event producers send information to an event source about changes to system state. Examples of event producers include a Kafka cluster or the Kubernetes API server.

An event source is a resource object, which is the link between an event producer and a sink (or consumer) that receives those events. Currently, OpenShift Serverless supports the following event source types:

ApiServerSource

Connects a sink to the Kubernetes API server.

PingSource

Periodically sends ping events with a constant payload. It can be used as a timer.

SinkBinding is also supported, which allows you to connect core Kubernetes resources such as Deployment, Job, or StatefulSet with a sink.

Examples of sinks are Knative services and channels. Events can also be sent to:

- A broker, where they can be filtered using triggers before being sent to a sink. Using the broker and trigger together enables an event delivery mechanism that hides the details of event routing from the event producer and event consumer.

- A channel, where Knative services can "subscribe" to receive events of a certain type.

10.1.2. Additional resources

- For more information about brokers, see Using brokers.
10.2. USING THE `kn` CLI TO LIST EVENT SOURCES AND EVENT SOURCE TYPES

You can use the `kn` CLI to list and manage available event sources or event source types for use with Knative Eventing.

Currently, `kn` supports management of the following event source types:

**ApiServerSource**
- Connects a sink to the Kubernetes API server.

**PingSource**
- Periodically sends ping events with a constant payload. It can be used as a timer.

10.2.1. Listing available event source types using `kn`

You can list the available event source types in the terminal by using the following command:

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

The default output for this command will look like:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApiServerSource</td>
<td>apiserversources.sources.knative.dev</td>
<td>Watch and send Kubernetes API events to a sink</td>
</tr>
<tr>
<td>PingSource</td>
<td>pingsources.sources.knative.dev</td>
<td>Periodically send ping events to a sink</td>
</tr>
<tr>
<td>SinkBinding</td>
<td>sinkbindings.sources.knative.dev</td>
<td>Binding for connecting a PodSpecable to a sink</td>
</tr>
</tbody>
</table>

It is also possible to list available event source types in YAML format:

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

10.2.2. Listing available event sources using `kn`

You can list available event sources by using the following command:

```bash
$ kn source list
```

Here is an example output:

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>RESOURCE</th>
<th>SINK</th>
<th>READY</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>ApiServerSource</td>
<td>apiserversources.sources.knative.dev</td>
<td>svc:eshow2</td>
<td>True</td>
</tr>
<tr>
<td>b1</td>
<td>SinkBinding</td>
<td>sinkbindings.sources.knative.dev</td>
<td>svc:eshow3</td>
<td>False</td>
</tr>
<tr>
<td>p1</td>
<td>PingSource</td>
<td>pingsources.sources.knative.dev</td>
<td>svc:eshow1</td>
<td>True</td>
</tr>
</tbody>
</table>

10.2.2.1. Listing event sources of a specific type only
You can list event sources of a specific type only, by using the --type flag. For example, to list all event sources of type PingSource:

```
$ kn source list --type PingSource
```

Here is an example output:

```
NAME   TYPE              RESOURCE                               SINK         READY
p1     PingSource        pingsources.sources.knative.dev        svc:eshow1   True
```

Next steps

- See the documentation on Using ApiServerSource.
- See the documentation on Using PingSource.

## 10.3. USING APISERVERSOURCE

ApiServerSource is an event source that can be used to connect an event sink, such as a Knative service, to the Kubernetes API server. ApiServerSource watches for Kubernetes events and forwards them to the Knative Eventing broker.

**NOTE**

Both of the following procedures require you to create YAML files.

If you change the names of the YAML files from those used in the examples, you must ensure that you also update the corresponding CLI commands.

### 10.3.1. Using the ApiServerSource with the Knative CLI (kn)

This guide describes the steps required to create, manage, and delete an ApiServerSource using kn commands.

**Prerequisites**

- You will need to have a Knative Serving and Eventing installation.
- You will need to have created the default broker in the same namespace that the ApiServerSource will be installed in.
- You will need to have the kn CLI installed.

**Procedure**

1. Create a service account, role, and role binding for the ApiServerSource.
   You can do this by creating a file named `authentication.yaml` and copying the following sample code into it:

   ```yaml
   apiVersion: v1
   kind: ServiceAccount
   metadata:
   name: events-sa
   ```
namespace: default

---

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: event-watcher
  namespace: default
rules:
  - apiGroups:
    - ""
    resources:
    - events
    verbs:
    - get
    - list
    - watch

---

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: k8s-ra-event-watcher
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: event-watcher
subjects:
  - kind: ServiceAccount
    name: events-sa
    namespace: default

1. Change this namespace to the namespace that you have selected for installing ApiServerSource.

**NOTE**

If you want to re-use an existing service account with the appropriate permissions, you must modify the `authentication.yaml` for that service account.

Create the service account, role binding and cluster binding by entering the following command:

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

2. Create an ApiServerSource event source by entering the following command:

```
$ kn source apiserver create testevents --sink broker:default --resource "event:v1" --service-account events-sa --mode Resource
```  

3. To check that the ApiServerSource is set up correctly, create a Knative service that dumps incoming messages to its log by entering the following command:
Create a trigger from the default broker to filter events to the service created in the previous step by entering the following `kn` command:

```bash
$ kn trigger create event-display-trigger --sink svc:event-display
```

Create events by launching a pod in the default namespace. You can do this by entering the following command:

```bash
$ oc create deployment hello-node --image=gcr.io/hello-minikube-zero-install/hello-node
```

Check that the controller is mapped correctly by entering the following command and inspecting the output:

```bash
$ kn source apiserver describe testevents
```

The output should be similar to:

```yaml
Name:                testevents
Namespace:           default
Annotations:         sources.knative.dev/creator=developer,
sources.knative.dev/lastModifier=developer
Age:                 3m
ServiceAccountName:  events-sa
Mode:                Resource
Sink:
  Name:               default
  Namespace:          default
  Kind:               Broker (eventing.knative.dev/v1alpha1)
Resources:
  Kind:               event (v1)
  Controller:         false
Conditions:
  OK TYPE                     AGE REASON
  ++ Ready                     3m
  ++ Deployed                  3m
  ++ SinkProvided              3m
  ++ SufficientPermissions     3m
  ++ EventTypesProvided        3m
```

**Verification steps**

You can verify that the Kubernetes events were sent into the Knative eventing system by looking at the message dumper function logs.

You can view the message dumper function logs by entering the following commands:

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

The logs should contain lines similar to the following:


10.3.1. Deleting the ApiServerSource

You can delete the ApiServerSource, trigger, service, service account, cluster role, and cluster binding created in this guide by entering the following `kn` and `oc` commands:

```bash
$ kn trigger delete event-display-trigger
$ kn service delete event-display
$ kn source apiserver delete testevents
$ oc delete -f authentication.yaml
```

10.3.2. Using the ApiServerSource with the YAML method

This guide describes the steps required to create, manage, and delete an ApiServerSource using YAML files.

**Prerequisites**

- You will need to have a Knative Serving and Eventing installation.
- You will need to have created the `default` broker in the same namespace as the one defined in the ApiServerSource YAML file.

**Procedure**

1. Create a service account, role, and role binding for the ApiServerSource.
You can do this by creating a file named `authentication.yaml` and copying the following sample code into it:

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: events-sa
  namespace: default

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: event-watcher
  namespace: default
rules:
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - get
  - list
  - watch

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: k8s-ra-event-watcher
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: event-watcher
subjects:
- kind: ServiceAccount
  name: events-sa
  namespace: default
```

Change this namespace to the namespace that you have selected for installing `ApiServerSource`.

**NOTE**

If you want to re-use an existing service account with the appropriate permissions, you must modify the `authentication.yaml` for that service account.

After you have created the `authentication.yaml` file, apply it by entering the following command:

```bash
$ oc apply --filename authentication.yaml
```
2. Create an ApiServerSource event source. You can do this by creating a file named `k8s-events.yaml` and copying the following sample code into it:

```yaml
apiVersion: sources.knative.dev/v1alpha1
group: ApiServerSource
metadata:
  name: testevents
spec:
serviceAccountName: events-sa
mode: Resource
resources:
- apiVersion: v1
  kind: Event
sink:
  ref:
    apiVersion: eventing.knative.dev/v1beta1
    kind: Broker
    name: default
```

After you have created the `k8s-events.yaml` file, apply it by entering the following command:

```
$ oc apply --filename k8s-events.yaml
```

3. To check that the ApiServerSource is set up correctly, create a Knative service that dumps incoming messages to its log. Copy the following sample YAML into a file named `service.yaml`:

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
namespace: default
spec:
template:
  spec:
    containers:
    - image: quay.io/openshift-knative/knative-eventing-sources-event-display:v0.13.2
```

After you have created the `service.yaml` file, apply it by entering the following command:

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

4. Create a trigger from the `default` broker to filter events to the service created in the previous step. You can create the trigger by creating a file named `trigger.yaml` and copying the following sample code into it:

```yaml
apiVersion: eventing.knative.dev/v1alpha1
kind: Trigger
metadata:
  name: event-display-trigger
namespace: default
spec:
```

After you have created the `trigger.yaml` file, apply it by entering the following command:

```
$ oc apply --filename trigger.yaml
```
After you have created the `trigger.yaml` file, apply it by entering the following command:

```bash
$ oc apply --filename trigger.yaml
```

5. Create events by launching a pod in the default namespace. You can do this by entering the following command:

```bash
$ oc create deployment hello-node --image=gcr.io/hello-minikube-zero-install/hello-node
```

6. Check that the controller is mapped correctly by entering the following command and inspecting the output:

```
$ oc get apiserversource.sources.knative.dev testevents -o yaml
```

The output should be similar to:

```json
apiVersion: sources.knative.dev/v1alpha1
kind: ApiServerSource
metadata:
  annotations:
  creationTimestamp: "2020-04-07T17:24:54Z"
  generation: 1
  name: testevents
  namespace: default
  resourceVersion: "62868"
  selfLink: /apis/sources.knative.dev/v1alpha1/namespaces/default/apiserversources/testevents2
uid: 1603d863-bb06-4d1c-b371-f580b4db99fa
spec:
  mode: Resource
  resources:
  - apiVersion: v1
    controller: false
    controllerSelector:
      apiVersion: ""
      kind: ""
      name: ""
      uid: ""
    kind: Event
    labelSelector: {}
    serviceAccountName: events-sa
  sink:
    ref:
      apiVersion: eventing.knative.dev/v1beta1
      kind: Broker
      name: default
```

**Verification steps**
You can verify that the Kubernetes events were sent into the Knative eventing system by looking at the message dumper function logs.

You can view the message dumper function logs by entering the following commands:

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

The logs should contain lines similar to the following:

```json
~ cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.apiserver.resource.update
  datacontenttype: application/json
...
Data,
{
  "apiVersion": "v1",
  "involvedObject": {
    "apiVersion": "v1",
    "fieldPath": "spec.containers\{hello-node\}",
    "kind": "Pod",
    "name": "hello-node",
    "namespace": "default",
    ...
  },
  "kind": "Event",
  "message": "Started container",
  "metadata": {
    "name": "hello-node.159d7608e3a3572c",
    "namespace": "default",
    ...
  },
  "reason": "Started",
  ...
}
```

### 10.3.2.1. Deleting the ApiServerSource

You can delete the ApiServerSource, trigger, service, service account, cluster role, and cluster binding created in this guide by entering the following `oc` commands:

```bash
$ oc delete --filename trigger.yaml
$ oc delete --filename service.yaml
$ oc delete --filename k8s-events.yaml
$ oc delete --filename authentication.yaml
```

### 10.4. USING A PINGSOURCE

A PingSource is used to periodically send ping events with a constant payload to an event consumer.

A PingSource can be used to schedule sending events, similar to a timer, as shown in the example:
apiVersion: sources.knative.dev/v1alpha2
kind: PingSource
metadata:
  name: test-ping-source
spec:
  schedule: "*/2 * * * *"  # 1
  jsonData: '{"message": "Hello world!"}'  # 2
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display

1. The schedule of the event specified using CRON expression.
2. The event message body expressed as a JSON encoded data string.
3. These are the details of the event consumer. In this example, we are using a Knative service named event-display.

### 10.4.1. Using a PingSource with the kn CLI

The following sections describe how to create, verify and remove a basic PingSource using the kn CLI.

**Prerequisites**

- You have Knative Serving and Eventing installed.
- You have the kn CLI installed.

**Procedure**

1. To verify that the PingSource is working, create a simple Knative service that dumps incoming messages to the service’s logs:

   ```sh
   $ kn service create event-display \
     --image quay.io/openshift-knative/knative-eventing-sources-event-display:v0.13.2
   ```

2. For each set of ping events that you want to request, create a PingSource in the same namespace as the event consumer:

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

3. Check that the controller is mapped correctly by entering the following command and inspecting the output:

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

   The output should be similar to:
Name: test-ping-source
Namespace: default
Annotations: sources.knative.dev/creator=developer,
sources.knative.dev/lastModifier=developer
Age: 15s
Schedule: */2 * * * *
Data: {"message": "Hello world!"}

Sink:
Name: event-display
Namespace: default
Resource: Service (serving.knative.dev/v1)

Conditions:
OK TYPE AGE REASON
++ Ready 8s
++ Deployed 8s
++ SinkProvided 15s
++ ValidSchedule 15s
++ EventTypeProvided 15s
++ ResourcesCorrect 15s

Verification steps
You can verify that the Kubernetes events were sent to the Knative event sink by looking at the sink pod's logs.

By default, Knative services terminate their pods if no traffic is received within a 60 second period. The example shown in this guide creates a PingSource that sends a message every 2 minutes, so each message should be observed in a newly created pod.

1. Watch for new pods created:

   $ watch oc get pods

2. Cancel watching the pods using Ctrl+C, then look at the logs of the created pod:

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

The logs should contain lines similar to the following:

```
☁ cloudevents.Event
Validation: valid
Context Attributes,
   specversion: 1.0
type: dev.knative.sources.ping
source: /apis/v1/namespaces/default/pingsources/test-ping-source
id: 99e4f4f6-08ff-4bff-acf1-47f61ded68c9
time: 2020-04-07T16:16:00.000601161Z
datacontenttype: application/json
Data,
{
   "message": "Hello world!"
}
```
10.4.1. Remove the PingSource

You can delete the PingSource and the service that you created by entering the following commands:

```bash
$ kn delete pingsources.sources.knative.dev test-ping-source
$ kn delete service.serving.knative.dev event-display
```

10.4.2. Using a PingSource with YAML

The following sections describe how to create, verify and remove a basic PingSource using YAML files.

**Prerequisites**

- You have Knative Serving and Eventing installed.

**NOTE**

The following procedure requires you to create YAML files.

If you change the names of the YAML files from those used in the examples, you must ensure that you also update the corresponding CLI commands.

**Procedure**

1. To verify that the PingSource is working, create a simple Knative service that dumps incoming messages to the service’s logs.
   a. Copy the example YAML into a file named `service.yaml`:

   ```yaml
   apiVersion: serving.knative.dev/v1
   kind: Service
   metadata:
     name: event-display
   spec:
     template:
       spec:
         containers:
           - image: quay.io/openshift-knative/knative-eventing-sources-event-display:v0.13.2
   
   b. Create the service:

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

2. For each set of ping events that you want to request, create a PingSource in the same namespace as the event consumer.
   a. Copy the example YAML into a file named `ping-source.yaml`:

   ```yaml
   apiVersion: sources.knative.dev/v1alpha2
   kind: PingSource
   metadata:
     name: test-ping-source
   spec:
     schedule: "*/2 * * * *"
   ```
jsonData: `{"message": "Hello world!"}`
sink:
  ref:
    apiVersion: serving.knative.dev/v1
    kind: Service
    name: event-display

b. Create the PingSource:

```$ oc apply --filename ping-source.yaml```

3. Check that the controller is mapped correctly by entering the following command and inspecting the output:

```$ oc get pingsource.sources.knative.dev test-ping-source -oyaml```

The output should be similar to:

```
apiVersion: sources.knative.dev/v1alpha2
kind: PingSource
metadata:
  annotations:
    sources.knative.dev/creator: developer
    sources.knative.dev/lastModifier: developer
  creationTimestamp: "2020-04-07T16:11:14Z"
  generation: 1
  name: test-ping-source
  namespace: default
  resourceVersion: "55257"
  selfLink: /apis/sources.knative.dev/v1alpha2/namespaces/default/pingsources/test-ping-source
  uid: 3d80d50b-f8c7-4c1b-99f7-3ec00e0a8164
spec:
  jsonData: '{ value: "hello" }'
  schedule: "*/2 * * * *
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
      namespace: default
```

**Verification steps**

You can verify that the Kubernetes events were sent to the Knative event sink by looking at the sink pod's logs.

By default, Knative services terminate their pods if no traffic is received within a 60 second period. The example shown in this guide creates a PingSource that sends a message every 2 minutes, so each message should be observed in a newly created pod.

1. Watch for new pods created:

```$ watch oc get pods```
2. Cancel watching the pods using Ctrl+C, then look at the logs of the created pod:

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

The logs should contain lines similar to the following:

```
☁ cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.sources.ping
  source: /apis/v1/namespaces/default/pingsources/test-ping-source
  id: 042ff529-240e-45ee-b40c-3a908129853e
  time: 2020-04-07T16:22:00.000791674Z
  datacontenttype: application/json
Data,
  {
    "message": "Hello world!"
  }
```

10.4.2.1. Remove the PingSource

You can delete the PingSource and the service that you created by entering the following commands:

```
$ oc delete --filename service.yaml
$ oc delete --filename ping-source.yaml
```

10.5. USING SINKBINDING

SinkBinding is used to connect event producers, or event sources, to an event consumer, or event sink, for example, a Knative service or application.

**NOTE**

Both of the following procedures require you to create YAML files.

If you change the names of the YAML files from those used in the examples, you must ensure that you also update the corresponding CLI commands.

10.5.1. Using SinkBinding with the Knative CLI (kn)

This guide describes the steps required to create, manage, and delete a SinkBinding instance using kn commands.

**Prerequisites**

- You have Knative Serving and Eventing installed.
- You have the kn CLI installed.

**Procedure**
1. To check that SinkBinding is set up correctly, create a Knative event display service, or event sink, that dumps incoming messages to its log:

   ```bash
   $ kn service create event-display --image quay.io/openshift-knative/knative-eventing-sources-event-display:v0.13.2
   ```

2. Create a SinkBinding that directs events to the service:

   ```bash
   $ kn source binding create bind-heartbeat --subject Job:batch/v1:app=heartbeat-cron --sink svc:event-display
   ```

3. Create a CronJob.

   a. Create a file named `heartbeats-cronjob.yaml` and copy the following sample code into it:

   ```yaml
   apiVersion: batch/v1beta1
   kind: CronJob
   metadata:
     name: heartbeat-cron
   spec:
     schedule: "* * * * *"
     jobTemplate:
       metadata:
         labels:
           app: heartbeat-cron
       spec:
         template:
           spec:
             restartPolicy: Never
           containers:
             - name: single-heartbeat
               image: quay.io/openshift-knative/knative-eventing-sources-heartbeats:v0.13.2
               args:
                 - --period=1
               env:
                 - name: ONE_SHOT
                   value: "true"
                 - name: POD_NAME
                   valueFrom:
                     fieldRef:
                       fieldPath: metadata.name
                 - name: POD_NAMESPACE
                   valueFrom:
                     fieldRef:
                       fieldPath: metadata.namespace
   ```

   b. After you have created the `heartbeats-cronjob.yaml` file, apply it by entering:

   ```bash
   $ oc apply --filename heartbeats-cronjob.yaml
   ```

4. Check that the controller is mapped correctly by entering the following command and inspecting the output:
$ kn source binding describe bind-heartbeat

The output should be similar to:

Name:         bind-heartbeat
Namespace:    demo-2
Annotations:  sources.knative.dev/creator=minikube-user,
sources.knative.dev/lastModifier=minikube ...
Age:          2m
Subject:      
Resource:   job (batch/v1)
Selector:    
app:      heartbeat-cron
Sink:        Name:       event-display
Resource:   Service (serving.knative.dev/v1)

Conditions:
OK TYPE     AGE REASON
++ Ready     2m

Verification steps

You can verify that the Kubernetes events were sent to the Knative event sink by looking at the
message dumper function logs.

You can view the message dumper function logs by entering:

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

The logs should contain lines similar to the following:

```plaintext
☁ cloudevents.Event
Validation: valid
Context Attributes,
specversion: 1.0
  type: dev.knative.eventing.samples.heartbeat
source: https://knative.dev/eventing-contrib/cmd/heartbeats/#event-test/mypod
id: 2b72d7bf-c38f-4a98-a433-608fbcdd2596
contenttype: application/json
Extensions,
  beats: true
  heart: yes
the: 42
Data,
  {
    "id": 1,
    "label": ""
  }
```

10.5.2. Using SinkBinding with the YAML method
This guide describes the steps required to create, manage, and delete a SinkBinding instance using YAML files.

Prerequisites

- You have Knative Serving and Eventing installed.

Procedure

1. To check that SinkBinding is set up correctly, create a Knative event display service, or event sink, that dumps incoming messages to its log.
   a. Copy the following sample YAML into a file named `service.yaml`:

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
spec:
template:
  spec:
    containers:
      - image: quay.io/openshift-knative/knative-eventing-sources-event-display:v0.13.2
```
   b. After you have created the `service.yaml` file, apply it by entering:

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

2. Create a SinkBinding that directs events to the service.
   a. Create a file named `sinkbinding.yaml` and copy the following sample code into it:

```
apiVersion: sources.knative.dev/v1alpha1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  subject:
    apiVersion: batch/v1
    kind: Job
    selector:
      matchLabels:
        app: heartbeat-cron
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

   1. In this example, any Job with the label `app: heartbeat-cron` will be bound to the event sink.
b. After you have created the `sinkbinding.yaml` file, apply it by entering:

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

3. Create a CronJob.

a. Create a file named `heartbeats-cronjob.yaml` and copy the following sample code into it:

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: heartbeat-cron
spec:
  spec:
    # Run every minute
    schedule: "* * * * *"
    jobTemplate:
      metadata:
        labels:
        - app: heartbeat-cron
      spec:
        template:
          spec:
            restartPolicy: Never
            containers:
              - name: single-heartbeat
                image: quay.io/openshift-knative/knative-eventing-sources-heartbeats:v0.13.2
                args:
                  - --period=1
                env:
                  - name: ONE_SHOT
                    value: "true"
                  - name: POD_NAME
                    valueFrom:
                      fieldRef:
                        fieldPath: metadata.name
                  - name: POD_NAMESPACE
                    valueFrom:
                      fieldRef:
                        fieldPath: metadata.namespace
```

b. After you have created the `heartbeats-cronjob.yaml` file, apply it by entering:

```
$ oc apply --filename heartbeats-cronjob.yaml
```

4. Check that the controller is mapped correctly by entering the following command and inspecting the output:

```
$ oc get sinkbindings.sources.knative.dev bind-heartbeat -oyaml
```

The output should be similar to:

```yaml
spec:
sink:
```
ref:
apiVersion: serving.knative.dev/v1
kind: Service
name: event-display
namespace: default
subject:
apiVersion: batch/v1
kind: Job
namespace: default
selector:
  matchLabels:
    app: heartbeat-cron

Verification steps

You can verify that the Kubernetes events were sent to the Knative event sink by looking at the message dumper function logs.

You can view the message dumper function logs by entering:

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

The logs should contain lines similar to the following:

---
cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
type: dev.knative.eventing.samples.heartbeat
source: https://knative.dev/eventing-contrib/cmd/heartbeats/#event-test/mypod
id: 2b72d7bf-c38f-4a98-a433-608fbccd2596
contenttype: application/json
Extensions,
  beats: true
  heart: yes
  the: 42
Data,
  {
    "id": 1,
    "label": ""
  }
---

10.6. USING TRIGGERS

All events which are sent to a channel or broker will be sent to all subscribers of that channel or broker by default.

Using triggers allows you to filter events from a channel or broker, so that subscribers will only receive a subset of events based on your defined criteria.

The Knative CLI provides a set of `kn trigger` commands that can be used to create and manage triggers.

Prerequisites
Before you can use triggers, you will need:

- Knative Eventing and `kn` installed.
- An available broker, either the default broker or one that you have created. You can create the default broker either by following the instructions on Using brokers with Knative Eventing, or by using the `--inject-broker` flag while creating a trigger. Use of this flag is described in the procedure below.
- An available event consumer, for example, a Knative service.

10.6.1. Creating a trigger using `kn`

**Procedure**

To create a trigger, enter the following command:

```
$ kn trigger create <TRIGGER-NAME> --broker <BROKER-NAME> --filter <KEY=VALUE> --sink <SINK>
```

To create a trigger and also create the default broker using broker injection, enter the following command:

```
$ kn trigger create <TRIGGER-NAME> --inject-broker --filter <KEY=VALUE> --sink <SINK>
```

**Example trigger YAML:**

```yaml
apiVersion: eventing.knative.dev/v1alpha1
kind: Trigger
metadata:
  name: trigger-example
spec:
  broker: default
  subscriber:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: my-service
```

1. The name of the trigger.
2. The name of the broker where events will be filtered from. If the broker is not specified, the trigger will revert to using the default broker.
3. The name of the service that will consumer filtered events.

10.6.2. Listing triggers using `kn`

The `kn trigger list` command prints a list of available triggers.

**Procedure**
To print a list of available triggers, enter the following command:

```
$ kn trigger list
```

Example output:

```
$ kn trigger list
NAME    BROKER    SINK           AGE   CONDITIONS   READY   REASON
 email   default   svc:edisplay   4s    5 OK / 5     True
 ping    default   svc:edisplay   32s   5 OK / 5     True
```

To print a list of triggers in JSON format, enter the following command:

```
$ kn trigger list -o json
```

10.6.3. Describing a trigger using `kn`

The `kn trigger describe` command prints information about a trigger.

**Procedure**

To print information about a trigger, enter the following command:

```
$ kn trigger describe <TRIGGER-NAME>
```

Example output:

```
$ kn trigger describe ping
Name:         ping
Namespace:    default
Labels:       eventing.knative.dev/broker=default
Annotations:  eventing.knative.dev/creator=kube:admin,
              eventing.knative.dev/lastModifier=kube:admin
Age:          2m
Broker:       default
Filter:
  type:       dev.knative.event
Sink:
  Name:       edisplay
  Namespace:  default
  Resource:   Service (serving.knative.dev/v1)
Conditions:
  OK TYPE      AGE REASON
  ++ Ready     2m
  ++ BrokerReady 2m
  ++ DependencyReady 2m
  ++ Subscribed 2m
  ++ SubscriberResolved 2m
```

10.6.4. Deleting a trigger using `kn`
Procedure
To delete a trigger, enter the following command:

$ kn trigger delete <TRIGGER-NAME>

10.6.5. Updating a trigger using kn
You can use the **kn trigger update** command with certain flags to quickly update attributes of a trigger.

Procedure
To update a trigger, enter the following command:

$ kn trigger update NAME --filter KEY=VALUE --sink SINK [flags]

You can update a trigger to filter exact event attributes that match incoming events, such as **type=knative.dev.event**. For example:

$ kn trigger update mytrigger --filter type=knative.dev.event

You can also remove a filter attribute from a trigger. For example, you can remove the filter attribute with key **type**:

$ kn trigger update mytrigger --filter type-

The following example shows how to update the sink of a trigger to **svc:new-service**:

$ kn trigger update mytrigger --sink svc:new-service

10.6.6. Filtering events using triggers
In the following trigger example, only events with attribute **type: dev.knative.samples.helloworld** will reach the event consumer.

$ kn trigger create foo --broker default --filter type=dev.knative.samples.helloworld --sink svc:mysvc

You can also filter events using multiple attributes. The following example shows how to filter events using the type, source, and extension attributes.

$ kn trigger create foo --broker default --sink svc:mysvc \  --filter type=dev.knative.samples.helloworld \  --filter source=dev.knative.samples/helloworldsource \  --filter myextension=my-extension-value

10.7. USING BROKERS WITH KNATIVE EVENTING
Knative Eventing uses the **default** broker unless otherwise specified.

If you have cluster administrator permissions, you can create the **default** broker automatically using namespace annotation.
All other users must create a broker using the manual process as described in this guide.

10.7.1. Creating a broker manually

To create a broker, you must create a ServiceAccount for each namespace and give that ServiceAccount the required RBAC permissions.

Prerequisites

- Knative Eventing installed, which includes the ClusterRole.

Procedure

1. Create the ServiceAccount objects:

```
$ oc -n default create serviceaccount eventing-broker-ingress
$ oc -n default create serviceaccount eventing-broker-filter
```

2. Give those objects RBAC permissions:

```
$ oc -n default create rolebinding eventing-broker-ingress --clusterrole=eventing-broker-ingress --serviceaccount=default:eventing-broker-ingress
$ oc -n default create rolebinding eventing-broker-filter --clusterrole=eventing-broker-filter --serviceaccount=default:eventing-broker-filter
```

3. Create the broker:

```
cat << EOF | oc apply -f -
apiVersion: eventing.knative.dev/v1beta1
kind: Broker
metadata:
  namespace: default
  name: default
EOF
```

1 This example uses the name `default`, but you can replace this with any other valid name.

10.7.2. Creating a broker automatically using namespace annotation

If you have cluster administrator permissions, you can create a broker automatically by annotating a namespace.

Prerequisites

- Knative Eventing installed.
- Cluster administrator permissions for OpenShift Container Platform.

Procedure
1. Annotate your namespace by entering the following commands:

```
$ oc label namespace default knative-eventing-injection=enabled
$ oc -n default get broker default
```

Replace `default` with the desired namespace.

The line shown in this example will automatically create a broker named `default` in the `default` namespace.

**NOTE**

Brokers created due to annotation will not be removed if you remove the annotation. You must manually delete them.

### 10.7.3. Deleting a broker that was created using namespace annotation

1. Delete the injected broker from the selected namespace (in this example, the `default` namespace):

```
$ oc -n default delete broker default
```

### 10.8. USING CHANNELS

It is possible to sink events from an event source to a Knative Eventing channel. Channels are Custom Resources (CRs) that define a single event-forwarding and persistence layer. After events have been sent to a channel, these events can be sent to multiple Knative services by using a subscription.

The default configuration for channel instances is defined in the `default-ch-webhook` ConfigMap. However, developers can still create their own channels directly by instantiating a supported channel object.

#### 10.8.1. Supported channel types

Currently, OpenShift Serverless only supports the use of InMemoryChannel channels as part of the Knative Eventing Technology Preview.

The following are limitations of InMemoryChannel channels:

- No event persistence is available. If a Pod goes down, events on that Pod will be lost.
- InMemoryChannel channels do not implement event ordering, so two events that are received in the channel at the same time may be delivered to a subscriber in any order.
- If a subscriber rejects an event, there are no redelivery attempts. Instead, the rejected event is sent to a `dead letter sink` if this sink exists, or is otherwise dropped.

#### 10.8.2. Creating a channel with cluster default configuration

To create a channel using the cluster default configuration set by the cluster administrator, you must create a generic `Channel` custom object.

**Example Channel object**
When the above `Channel` object is created, a mutating admission webhook adds a set of `spec.channelTemplate` properties for the `Channel` object based on the default channel implementation chosen by the cluster administrator.

**Example Channel object with spec.channelTemplate properties**

```yaml
apiVersion: messaging.knative.dev/v1beta1
kind: Channel
metadata:
  name: example-channel
  namespace: default

spec:
  channelTemplate:
    apiVersion: messaging.knative.dev/v1beta1
    kind: InMemoryChannel
```

The channel controller will then create the backing channel instance based on that `spec.channelTemplate` configuration. The `spec.channelTemplate` properties cannot be changed after creation, because they are set by the default channel mechanism rather than by the user.

When this mechanism is used, two objects are created, a generic channel, and an `InMemoryChannel` channel.

The generic channel acts as a proxy that copies its subscriptions to the `InMemoryChannel`, and sets its status to reflect the status of the backing `InMemoryChannel` channel.

Because the channel in this example is created in the default namespace, the channel uses the cluster default, which is `InMemoryChannel`. 
CHAPTER 11. USING METERING WITH OPENSShift SERVERLESS

As a 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.

11.1. INSTALLING METERING

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

11.2. DATASOURCES FOR KNATIVE SERVING METERING

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

11.2.1. 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

```yaml
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!=""
```

11.2.2. Datasource for memory usage in Knative Serving

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

YAML file

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

    label_replace(rate(container_memory_usage_bytes{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!=""
```
11.2.3. 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

11.3. QUERIES FOR KNATIVE SERVING METERING

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

11.3.1. Query for CPU usage in Knative Serving

YAML file

```yaml
apiVersion: metering.openshift.io/v1
category: 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:
```
11.3.2. Query for memory usage in Knative Serving

YAML file

```yaml
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
```

11.3.2. Query for memory usage in Knative Serving

```
SELECT
  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 |}
GROUP BY labels['namespace'],labels['label_serving_knative_dev_service']
```
11.3.3. 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
```

11.4. 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

```yaml
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: knative-service-cpu-usage
spec:
  reportingStart: '2019-06-01T00:00:00Z'
  query: knative-service-cpu-usage
  runImmediately: true
```

1. Start date of the report, in ISO 8601 format.
2. End date of the report, in ISO 8601 format.

11.4.1. 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
```