OpenShift Serverless installation, usage, and release notes
Abstract

This document provides information on how to use OpenShift Serverless in OpenShift Container Platform.
# Table of Contents

CHAPTER 1. OPENSIGHT SERVERLESS RELEASE NOTES .................................................. 6  
1.1. RELEASE NOTES FOR RED HAT OPENSIGHT SERVERLESS 1.10.1 .......................... 6  
  1.1.1. Fixed issues ................................................................................................. 6  
1.2. RELEASE NOTES FOR RED HAT OPENSIGHT SERVERLESS 1.10.0 ...................... 6  
  1.2.1. New features ............................................................................................. 6  
  1.2.2. Fixed issues .............................................................................................. 6  
1.3. RELEASE NOTES FOR RED HAT OPENSIGHT SERVERLESS 1.9.0 ....................... 7  
  1.3.1. New features ............................................................................................. 7  
  1.3.2. Known issues ......................................................................................... 7  
1.4. RELEASE NOTES FOR RED HAT OPENSIGHT SERVERLESS 1.8.0 ....................... 7  
  1.4.1. New features ............................................................................................. 7  
  1.4.2. Known issues ......................................................................................... 7  
1.5. RELEASE NOTES FOR RED HAT OPENSIGHT SERVERLESS 1.7.2 ....................... 8  
  1.5.1. Fixed issues .............................................................................................. 9  
1.6. RELEASE NOTES FOR RED HAT OPENSIGHT SERVERLESS 1.7.1 ....................... 9  
  1.6.1. New features ............................................................................................ 9  
  1.6.2. Fixed issues ............................................................................................. 9  
1.7. RELEASE NOTES FOR RED HAT OPENSIGHT SERVERLESS 1.7.0 ....................... 9  
  1.7.1. New features ............................................................................................ 9  
  1.7.2. Fixed issues ............................................................................................. 10  
  1.7.3. Known issues ..................................................................................... 10  
1.8. ADDITIONAL RESOURCES ................................................................................ 11  

CHAPTER 2. OPENSIGHT SERVERLESS SUPPORT .......................................................... 12  
2.1. GETTING SUPPORT ......................................................................................... 12  
2.2. GATHERING DIAGNOSTIC INFORMATION FOR SUPPORT ......................... 12  
  2.2.1. About the must-gather tool ...................................................................... 12  
  2.2.2. About collecting OpenShift Serverless data .......................................... 12  

CHAPTER 3. GETTING STARTED WITH OPENSIGHT SERVERLESS ............................. 14  
3.1. HOW OPENSIGHT SERVERLESS WORKS .......................................................... 14  
3.2. SUPPORTED CONFIGURATIONS ..................................................................... 14  
3.3. NEXT STEPS .................................................................................................. 14  

CHAPTER 4. INSTALLING OPENSIGHT SERVERLESS .................................................. 15  
4.1. INSTALLING OPENSIGHT SERVERLESS .......................................................... 15  
  4.1.1. Defining cluster size requirements for an OpenShift Serverless installation 15  
  4.1.2. Additional requirements for advanced use cases .................................. 15  
  4.1.3. Scaling your cluster using machine sets ................................................. 16  
  4.1.4. Installing the OpenShift Serverless Operator ....................................... 16  
  4.1.5. Next steps ................................................................................................ 17  
4.2. INSTALLING KNATIVE SERVING ................................................................... 17  
  4.2.1. Creating the knative-serving namespace ............................................... 18  
  4.2.1.1. Creating the knative-serving namespace using the web console 18  
  4.2.1.2. Creating the knative-serving namespace using the CLI .................. 19  
  4.2.2. Prerequisites ......................................................................................... 19  
  4.2.3. Installing Knative Serving using the web console .................................. 19  
  4.2.4. Installing Knative Serving using YAMAL ............................................ 22  
  4.2.5. Next steps .............................................................................................. 24  
4.3. INSTALLING KNATIVE EVENTING ................................................................. 24  
  4.3.1. Creating the knative-eventing namespace ............................................. 24  
  4.3.1.1. Creating the knative-eventing namespace using the web console 24
4.3.1.2. Creating the knative-eventing namespace using the CLI
4.3.2. Prerequisites
4.3.3. Installing Knative Eventing using the web console
4.3.4. Installing Knative Eventing using YAML
4.3.5. Next steps
4.4. ADVANCED INSTALLATION CONFIGURATION OPTIONS
4.4.1. Knative Serving supported installation configuration options
  4.4.1.1. Controller custom certificates
  4.4.1.2. High availability
  4.4.2. Additional resources
4.5. UPGRADING OPENSIGHT SERVERLESS
  4.5.1. Upgrading the Subscription Channel
4.6. REMOVING OPENSIGHT 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. ARCHITECTURE .......................................................... 37
5.1. KNATIVE SERVING ARCHITECTURE ................................. 37
  5.1.1. Knative Serving CRDs ........................................... 37
5.2. KNATIVE EVENTING ARCHITECTURE ............................. 37
  5.2.1. Event sinks ..................................................... 38

CHAPTER 6. CREATING AND MANAGING SERVERLESS APPLICATIONS .............................. 39
6.1. SERVERLESS APPLICATIONS USING KNATIVE SERVICES ......................... 39
6.2. CREATING SERVERLESS APPLICATIONS USING THE OPENSIGHT CONTAINER PLATFORM WEB CONSOLE .................................................. 39
  6.2.1. Creating serverless applications using the Administrator perspective .... 39
  6.2.2. Creating serverless applications using the Developer perspective ....... 40
6.3. CREATING SERVERLESS APPLICATIONS USING THE KN CLI .................. 40
6.4. CREATING SERVERLESS APPLICATIONS USING YAML ....................... 41
6.5. VERIFYING YOUR SERVERLESS APPLICATION DEPLOYMENT ................. 42
6.6. INTERACTING WITH A SERVERLESS APPLICATION USING HTTP2 AND GRPC . 43

CHAPTER 7. HIGH AVAILABILITY ON OPENSIGHT SERVERLESS ...................................... 45
7.1. CONFIGURING HIGH AVAILABILITY REPLICA ON OPENSIGHT SERVERLESS .......... 45

CHAPTER 8. TRACING REQUESTS USING JAEGAR .................................................. 48
8.1. CONFIGURING JAEGAR FOR USE WITH OPENSIGHT SERVERLESS ............... 48

CHAPTER 9. KNATIVE SERVING ........................................................................... 50
9.1. USING KN TO COMPLETE SERVING TASKS .............................................. 50
  9.1.1. Basic workflow using kn ......................................... 50
  9.1.2. Autoscaling workflow using kn .................................. 52
  9.1.3. Traffic splitting using kn ......................................... 52
  9.1.3.1. Assigning tag revisions ...................................... 53
  9.1.3.2. Unassigning tag revisions .................................... 54
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1.3.3</td>
<td>Traffic flag operation precedence</td>
<td>54</td>
</tr>
<tr>
<td>9.1.3.4</td>
<td>Traffic splitting flags</td>
<td>55</td>
</tr>
<tr>
<td>9.2.1.</td>
<td>Configuring concurrent requests for Knative Serving autoscaling</td>
<td>55</td>
</tr>
<tr>
<td>9.2.1.1</td>
<td>Configuring concurrent requests using the target annotation</td>
<td>56</td>
</tr>
<tr>
<td>9.2.1.2</td>
<td>Configuring concurrent requests using the containerConcurrency field</td>
<td>57</td>
</tr>
<tr>
<td>9.2.2.</td>
<td>Configuring scale bounds Knative Serving autoscaling</td>
<td>57</td>
</tr>
<tr>
<td>9.3.1.</td>
<td>About deploying cluster logging</td>
<td>58</td>
</tr>
<tr>
<td>9.3.2.</td>
<td>About deploying and configuring cluster logging</td>
<td>58</td>
</tr>
<tr>
<td>9.3.2.1</td>
<td>Configuring and Tuning Cluster Logging</td>
<td>58</td>
</tr>
<tr>
<td>9.3.2.2</td>
<td>Sample modified ClusterLogging custom resource</td>
<td>60</td>
</tr>
<tr>
<td>9.3.3.</td>
<td>Using cluster logging to find logs for Knative Serving components</td>
<td>61</td>
</tr>
<tr>
<td>9.3.4.</td>
<td>Using cluster logging to find logs for services deployed with Knative Serving</td>
<td>62</td>
</tr>
<tr>
<td>9.4.1.</td>
<td>Splitting traffic between revisions using the Developer perspective</td>
<td>62</td>
</tr>
<tr>
<td>10.1.1.1</td>
<td>Creating a broker using the Knative CLI</td>
<td>65</td>
</tr>
<tr>
<td>10.1.1.2</td>
<td>Creating a broker by annotating a trigger</td>
<td>66</td>
</tr>
<tr>
<td>10.1.1.3</td>
<td>Creating a broker by labeling a namespace</td>
<td>67</td>
</tr>
<tr>
<td>10.1.2.1</td>
<td>Listing existing brokers using the Knative CLI</td>
<td>68</td>
</tr>
<tr>
<td>10.1.2.2</td>
<td>Describing an existing broker using the Knative CLI</td>
<td>69</td>
</tr>
<tr>
<td>10.1.2.3</td>
<td>Deleting a broker that was created by injection</td>
<td>70</td>
</tr>
<tr>
<td>10.1.3.1</td>
<td>Creating a trigger using the Developer perspective</td>
<td>71</td>
</tr>
<tr>
<td>10.1.3.2</td>
<td>Deleting a trigger using the Developer perspective</td>
<td>72</td>
</tr>
<tr>
<td>10.1.3.3</td>
<td>Creating a trigger using kn</td>
<td>73</td>
</tr>
<tr>
<td>10.1.3.4</td>
<td>Listing triggers using kn</td>
<td>73</td>
</tr>
<tr>
<td>10.1.3.4.1</td>
<td>Describing a trigger using kn</td>
<td>74</td>
</tr>
<tr>
<td>10.1.3.4.2</td>
<td>Filtering events using triggers</td>
<td>74</td>
</tr>
<tr>
<td>10.1.3.4.3</td>
<td>Updating a trigger using kn</td>
<td>75</td>
</tr>
<tr>
<td>10.1.3.4.4</td>
<td>Deleting a trigger using kn</td>
<td>75</td>
</tr>
<tr>
<td>10.2.1.1</td>
<td>Supported channel types</td>
<td>76</td>
</tr>
<tr>
<td>10.2.2.</td>
<td>Creating a development channel</td>
<td>77</td>
</tr>
<tr>
<td>10.2.3.</td>
<td>Creating a subscription</td>
<td>78</td>
</tr>
<tr>
<td>11.1.1.</td>
<td>Creating event sources</td>
<td>80</td>
</tr>
<tr>
<td>11.1.2.</td>
<td>Additional resources</td>
<td>80</td>
</tr>
<tr>
<td>11.2.1.1</td>
<td>Listing available event source types using the Knative CLI</td>
<td>80</td>
</tr>
<tr>
<td>11.2.2.</td>
<td>Listing available event sources using the Knative CLI</td>
<td>81</td>
</tr>
<tr>
<td>11.2.3.</td>
<td>Next steps</td>
<td>81</td>
</tr>
<tr>
<td>11.3.1.</td>
<td>Prerequisites</td>
<td>81</td>
</tr>
<tr>
<td>11.3.2.</td>
<td>Creating a service account, role, and binding for event sources</td>
<td>82</td>
</tr>
</tbody>
</table>
### Chapter 11.3. Networking

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.3.3. Creating an ApiServerSource event source using the Developer perspective</td>
<td>83</td>
</tr>
<tr>
<td>11.3.4. Deleting the ApiServerSource</td>
<td>84</td>
</tr>
<tr>
<td>11.3.5. Using the API server source with the Knative CLI</td>
<td>85</td>
</tr>
<tr>
<td>11.3.6. Deleting an API server source using the Knative CLI</td>
<td>88</td>
</tr>
<tr>
<td>11.3.7. Creating an API server source using YAML files</td>
<td>88</td>
</tr>
<tr>
<td>11.3.8. Deleting the API server source</td>
<td>92</td>
</tr>
<tr>
<td>11.4. USING A PING SOURCE</td>
<td>93</td>
</tr>
<tr>
<td>11.4.1. Creating a ping source using the Knative CLI</td>
<td>93</td>
</tr>
<tr>
<td>11.4.1.1. Remove the ping source</td>
<td>95</td>
</tr>
<tr>
<td>11.4.2. Creating a ping source using YAML files</td>
<td>95</td>
</tr>
<tr>
<td>11.4.2.1. Remove the PingSource</td>
<td>97</td>
</tr>
<tr>
<td>11.5. USING SINK BINDING</td>
<td>98</td>
</tr>
<tr>
<td>11.5.1. Using sink binding with the Knative CLI</td>
<td>98</td>
</tr>
<tr>
<td>11.5.2. Using sink binding with the YAML method</td>
<td>101</td>
</tr>
</tbody>
</table>

### Chapter 12. Networking

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.1. USING SERVICE MESH WITH OPENSHIFT SERVERLESS</td>
<td>105</td>
</tr>
<tr>
<td>12.1.1. Enabling sidecar injection for a Knative service</td>
<td>105</td>
</tr>
<tr>
<td>12.1.2. Additional resources</td>
<td>106</td>
</tr>
<tr>
<td>12.2. USING JSON WEB TOKEN AUTHENTICATION WITH SERVICE MESH AND OPENSHIFT SERVERLESS</td>
<td>106</td>
</tr>
<tr>
<td>12.2.1. Additional resources</td>
<td>108</td>
</tr>
<tr>
<td>12.3. USING CUSTOM DOMAINS FOR KNATIVE SERVICES WITH SERVICE MESH</td>
<td>108</td>
</tr>
<tr>
<td>12.3.1. Setting cluster availability to cluster-local</td>
<td>108</td>
</tr>
<tr>
<td>12.3.2. Creating necessary Service Mesh resources</td>
<td>109</td>
</tr>
<tr>
<td>12.3.3. Accessing a service using your custom domain</td>
<td>111</td>
</tr>
<tr>
<td>12.3.4. Additional resources</td>
<td>111</td>
</tr>
</tbody>
</table>

### Chapter 13. Using Metering with OpenShift Serverless

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.1. INSTALLING METERING</td>
<td>112</td>
</tr>
<tr>
<td>13.2. DATASOURCES FOR KNATIVE SERVING METERING</td>
<td>112</td>
</tr>
<tr>
<td>13.2.1. Datasource for CPU usage in Knative Serving</td>
<td>112</td>
</tr>
<tr>
<td>13.2.2. Datasource for memory usage in Knative Serving</td>
<td>112</td>
</tr>
<tr>
<td>13.2.3. Applying data sources for Knative Serving metering</td>
<td>113</td>
</tr>
<tr>
<td>13.3. QUERIES FOR KNATIVE SERVING METERING</td>
<td>113</td>
</tr>
<tr>
<td>13.3.1. Query for CPU usage in Knative Serving</td>
<td>113</td>
</tr>
<tr>
<td>13.3.2. Query for memory usage in Knative Serving</td>
<td>114</td>
</tr>
<tr>
<td>13.3.3. Applying queries for Knative Serving metering</td>
<td>115</td>
</tr>
<tr>
<td>13.4. METERING REPORTS FOR KNATIVE SERVING</td>
<td>115</td>
</tr>
<tr>
<td>13.4.1. Running a metering report</td>
<td>116</td>
</tr>
</tbody>
</table>

### Chapter 14. Integrations

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.1. USING NVIDIA GPU RESOURCES WITH SERVERLESS APPLICATIONS</td>
<td>117</td>
</tr>
<tr>
<td>14.1.1. Additional resources</td>
<td>117</td>
</tr>
</tbody>
</table>
CHAPTER 1. OPENSIFT SERVERLESS RELEASE NOTES

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

**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/](https://access.redhat.com/support/offerings/techpreview/).

1.1. RELEASE NOTES FOR RED HAT OPENSIFT SERVERLESS 1.10.1

This release of OpenShift Serverless addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.1.1. Fixed issues

- A universal base image (UBI) has been upgraded from `ubi8-minimal-container-8.2-349` in 1.10.0 to `ubi8-minimal-container-8.3-230` in this release.

1.2. RELEASE NOTES FOR RED HAT OPENSIFT SERVERLESS 1.10.0

1.2.1. New features

- OpenShift Serverless now uses Knative Operator 0.16.0.
- OpenShift Serverless now uses Knative Serving 0.16.0.
- OpenShift Serverless uses Knative Eventing 0.16.0.
- OpenShift Serverless now uses Kourier 0.16.0.
- OpenShift Serverless now uses Knative `kn` CLI 0.16.1.
- The annotation `knative-eventing-injection=enabled` that was previously used to label namespaces for broker creation is now deprecated. The new annotation is `eventing.knative.dev/injection=enabled`. For more information, see the documentation on Event delivery workflows using brokers and triggers.
- Multi-container support is now available on Knative as a Technology Preview feature. You can enable multi-container support in the `config-features` config map. For more information, see the Knative documentation.

1.2.2. Fixed issues

- In previous releases, Knative Serving had a fixed, minimum CPU request of 25m for `queue-proxy`. If your cluster had any value set that conflicted with this, for example, if you had set a minimum CPU request for `defaultRequest` of more than 25m, the Knative service failed to
deploy. This issue is fixed in 1.10.0.

1.3. RELEASE NOTES FOR RED HAT OPENSHIFT SERVERLESS 1.9.0

1.3.1. New features

- OpenShift Serverless now uses Knative Operator 0.15.2. The Knative Serving and Knative Eventing Operators are now merged into a common Operator.

- OpenShift Serverless now uses Knative Serving 0.15.2.

- OpenShift Serverless now uses Knative kn CLI 0.15.2.

- OpenShift Serverless uses Knative Eventing 0.15.2.

- OpenShift Serverless now uses Kourier 0.15.0.

- OpenShift Serverless now supports some integrated Red Hat OpenShift Service Mesh features, including enabling sidecars, and JSON Web Token (JWT) authentication. Supported features are documented in the Networking guide.

1.3.2. Known issues

- After deleting the KnativeEventing custom resource (CR), the v0.15.0-upgrade-xr55x and storage-version-migration-eventing-99c7q pods remain on the cluster and show a Completed status. You can delete the namespace where the KnativeEventing CR was installed to completely remove these pods.

1.4. RELEASE NOTES FOR RED HAT OPENSHIFT SERVERLESS 1.8.0

1.4.1. New features

- OpenShift Serverless now uses Knative Serving 0.14.1.

- OpenShift Serverless now uses Knative Serving Operator 0.14.0.

- OpenShift Serverless now uses Knative kn CLI 0.14.0.

- OpenShift Serverless uses Knative Eventing 0.14.2.

- OpenShift Serverless now uses Knative Eventing Operator 0.14.0.

- OpenShift Serverless now uses Kourier 0.14.1.

1.4.2. Known issues

- Knative Serving has a fixed, minimum CPU request of 25m for the queue-proxy setting. If your cluster has any value set that conflicts with this, for example, if you have set a minimum CPU request for defaultRequest of more than 25m, the Knative service fails to deploy. As a workaround, you can configure the resourcePercentage annotation individually for your Knative services.

Example resourcePercentage configuration
spec:
  template:
    metadata:
      annotations:
        queue.sidecar.serving.knative.dev/resourcePercentage: "10"  

queue.sidecar.serving.knative.dev/resourcePercentage is the percentage of user container resources to be used for queue-proxy. This can be between a range of 0.1 - 100.

- On OpenShift Container Platform 4.5 and newer versions, deploying a Knative service with traffic distribution shows an invalid URL for the general service address in the Developer perspective of the web console. The correct URL is shown in YAML resources and CLI command outputs.

- If you use a ping source with OpenShift Serverless, after you uninstall and delete all other Knative Eventing components, the pingsource-jobrunner Deployment resource is not deleted.

- If you delete a sink before you delete the sink binding connected to it, the SinkBinding object deletion might hang. As a workaround, you can edit the SinkBinding object and remove the finalizer that causes the hanging:

  finalizers:
  - sinkbindings.sources.knative.dev

- The sink binding behavior has changed in OpenShift Serverless 1.8.0, which breaks backwards compatibility. To use sink binding, cluster administrators must now label the namespace configured in the SinkBinding object with bindings.knative.dev/include:"true".

  Resources configured in the SinkBinding object must also be labeled with bindings.knative.dev/include:"true"; however, this task can be completed by any OpenShift Serverless user.

  1. As a cluster administrator, you can label the namespace by entering the following command:

     $ oc label namespace <namespace> bindings.knative.dev/include=true

  2. Users must manually add a bindings.knative.dev/include=true label to resources. For example, to add this label to a CronJob object, add the following lines to the Job resource YAML definition:

     jobTemplate:
       metadata:
         labels:
           app: heartbeat-cron
           bindings.knative.dev/include: "true"

1.5. RELEASE NOTES FOR RED HAT OPENShift SERVERLESS 1.7.2

This release of OpenShift Serverless addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.
1.5.1. Fixed issues

- In previous versions of OpenShift Serverless, KnativeServing custom resources showed a status of Ready, even if Kourier did not deploy. This bug is fixed in OpenShift Serverless 1.7.2.

1.6. RELEASE NOTES FOR RED HAT OPENSHIFT SERVERLESS 1.7.1

1.6.1. 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.

1.6.2. 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.

1.7. RELEASE NOTES FOR RED HAT OPENSHIFT SERVERLESS 1.7.0

1.7.1. 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 the OpenShift Container Platform cluster-wide proxy, and is now fully compatible with OpenShift Container Platform proxy settings.

- OpenShift Serverless now supports HTTPS by 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.

1.7.2. Fixed issues

- In previous versions, requesting `KnativeServing` CRs without specifying an API group, for example, by using the command `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.
```

The OpenShift Serverless Operator now restarts the Knative Serving controller whenever a new service CA certificate is generated, which ensures that the controller is always configured to use the current service CA certificate. For more information, see the OpenShift Container Platform documentation on Securing service traffic using service serving certificate secrets under Authentication.

1.7.3. 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 the `KnativeServing` CR becomes Ready.

The `3scale-kourier-control` controller 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`. 

OpenShift Container Platform 4.5 Serverless
1.8. 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 CLI releases page.
- For details about the latest Knative Eventing release, see the Knative Eventing releases page.
CHAPTER 2. OPENSIFT SERVERLESS SUPPORT

2.1. 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.

2.2. GATHERING DIAGNOSTIC INFORMATION FOR 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.

2.2.1. 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.

2.2.2. About collecting OpenShift 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 and the image tag for your installed version of OpenShift Serverless.
Procedure

- Collect data by using the `oc adm must-gather` command:

  ```
  $ oc adm must-gather --image=registry.redhat.io/openshift-serverless-1/svls-must-gather-rhel8:<image_version_tag>
  ```

Example command

  ```
  $ oc adm must-gather --image=registry.redhat.io/openshift-serverless-1/svls-must-gather-rhel8:1.10.0
  ```
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.

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/.

3.3. NEXT STEPS

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

- View the OpenShift Serverless release notes.
4.1. INSTALLING OPENSHIFT 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](#).

**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 version of Knative Eventing using the OpenShift Serverless Operator.

### 4.1.1. Defining cluster size requirements for an OpenShift Serverless installation

To install and use OpenShift Serverless, the OpenShift Container Platform cluster must be sized correctly. The minimum requirement for 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 approximately 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.

**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

### 4.1.2. 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](#).
4.1.3. Scaling your cluster using machine sets

You can use the OpenShift Container Platform 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 machine sets by two additional machines. See Manually scaling a machine set.

4.1.4. 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 Install Operator 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 4.5 channel as the Update Channel. The 4.5 channel will enable installation of the latest stable release of the OpenShift Serverless Operator.

   d. Select Automatic or Manual approval strategy.

5. Click Install 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**

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](https://docs.openshift.com/container-platform/).  

**4.1.5. Next steps**

- After the OpenShift Serverless Operator is installed, you can install the Knative Serving component. See the documentation on [Installing Knative Serving](https://www.knative.dev/docs/install/).

- After the OpenShift Serverless Operator is installed, you can install the Knative Eventing component. See the documentation on [Installing Knative Eventing](https://www.knative.dev/docs/install/).

**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.

   ![OpenShift Container Platform web console](image)

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
   ```

### 4.2.2. Prerequisites

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

### 4.2.3. 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**

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.

   ![Example Image]

   **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.

### 4.2.4. 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:

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

**Verification**

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

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

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:

```
$ 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-tqwvw    1/1     Running   0          9m26s
controller-7fd5655f49-9gxz5        1/1     Running   0          9m32s
controller-7fd5655f49-pncv5        1/1     Running   0          9m14s
kn-cli-downloads-8c65d4cbf-mt4t7   1/1     Running   0          9m42s
webhook-5c7d678c7c-n267j           1/1     Running   0          9m35s
```
4.2.5. 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:

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

**4.3.2. Prerequisites**

- An OpenShift Container Platform account with cluster administrator access
- Installed OpenShift Serverless Operator
- Created the `knative-eventing` namespace
4.3.3. 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

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.

   ![Example Image]

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

   ![Example Image]

   **Conditions**
   - **Type**: Ready
   - **Status**: True
   - **Updated**: 1 minute ago
   - **Reason**: -
   - **Message**: -

**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.
4.3.4. 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:

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

Verification

1. To verify the installation is complete, enter:

   ```
   $ 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:

   ```
   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:

   ```
   $ oc get pods -n knative-eventing
   ```

   The output should look similar to:

   ```
   NAME                                   READY   STATUS    RESTARTS   AGE
   broker-controller-58765d9d49-g9zp6     1/1     Running   0          7m21s
   eventing-controller-65fdd66b54-jw7bh   1/1     Running   0          7m31s
   eventing-webhook-57fd74b5bd-kvhlz       1/1     Running   0          7m31s
   imc-controller-5b75d458fc-ptvm2         1/1     Running   0          7m19s
   imc-dispatcher-64f6d5fccc-kkc4c          1/1     Running   0          7m18s
   ```
4.3.5. Next steps

- 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 OpenShift Serverless components.

4.4.1. Knative Serving supported installation configuration options

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

**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.

4.4.1.1. Controller custom certificates

If your registry uses a self-signed certificate, you must enable tag-to-digest resolution by creating a config map 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.

**IMPORTANT**

The config map or secret must reside in the same namespace as the Knative Serving custom resource definition (CRD).
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
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  controller-custom-certs:
    name: certs
    type: ConfigMap
```

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

The supported types are `ConfigMap` and `Secret`.

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

**Example default YAML**

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

**Example YAML**

```yaml
apiVersion: operator.knative.dev/v1alpha1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  high-availability:
    replicas: 2
```
4.4.2. Additional resources

- For more information about configuring high availability, see High availability on OpenShift Serverless.

4.5. UPGRADEING OPENSIFT SERVERLESS

If you have 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. Upgrading the Subscription Channel

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 4.5, and then click Save.

6. Wait until all pods have been upgraded in the knative-serving namespace and the KnativeServing custom resource (CR) reports the latest Knative Serving version.

Verification

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:

   $ 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:

   $ 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 OPENSShift 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:

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

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

   $ oc delete namespace knative-serving

#### 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:

   $ 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 custom resource definitions (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 have uninstalled Knative Serving and Knative Eventing and have removed the OpenShift Serverless Operator.

**Procedure**

- Delete the remaining OpenShift Serverless CRDs:

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

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

**NOTE**

`kn` does not have its own login mechanism. To log in to the cluster, you must install the `oc` CLI and use `oc login`.

Installation options for the `oc` CLI will vary depending on your operating system.

For more information on installing the `oc` CLI for your operating system and logging in with `oc`, see the [CLI getting started](#) documentation.

#### 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 ![Help](#) 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> 1
  # subscription-manager repos --enable="openshift-serverless-1-for-rhel-8-x86_64-rpms"
  # yum install openshift-serverless-clients
  ```

1. **Pool ID** for an active OpenShift Container Platform subscription

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. ARCHITECTURE

5.1. KNATIVE SERVING ARCHITECTURE

Knative Serving on OpenShift Container Platform enables developers to write cloud-native applications using serverless architecture. Serverless is a cloud computing model where application developers don’t need to provision servers or manage scaling for their applications. These routine tasks are abstracted away by the platform, allowing developers to push code to production much faster than in traditional models.

Knative Serving supports deploying and managing cloud-native applications by providing a set of objects as Kubernetes custom resource definitions (CRDs) that define and control the behavior of serverless workloads on an OpenShift Container Platform cluster. For more information about CRDs, see Extending the Kubernetes API with custom resource definitions.

Developers use these CRDs to create custom resource (CR) instances that can be used as building blocks to address complex use cases. For example:

- Rapidly deploying serverless containers.
- Automatically scaling pods.

For more information about CRs, see Managing resources from custom resource definitions.

5.1.1. Knative Serving CRDs

Service
The service.serving.knative.dev CRD automatically manages the life cycle of your workload to ensure that the application is deployed and reachable through the network. It creates a route, a configuration, and a new revision for each change to a user created service or CR. Most developer interactions in Knative are carried out by modifying services.

Revision
The revision.serving.knative.dev CRD 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 necessary.

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

Configuration
The configuration.serving.knative.dev CRD maintains the desired state for your deployment. It provides a clean separation between code and configuration. Modifying a configuration creates a new revision.

5.2. KNATIVE EVENTING ARCHITECTURE

Knative Eventing on OpenShift Container Platform enables developers to use an event-driven architecture with serverless applications. An event-driven architecture is based on the concept of decoupled relationships between event producers that create events, and event sinks, or consumers, that receive them.
Knative Eventing uses standard HTTP POST requests to send and receive events between event producers and consumers. These events conform to the CloudEvents specifications, which enables creating, parsing, sending, and receiving events in any programming language.

You can propagate an event from an event source to multiple event sinks by using:

- channels and subscriptions, or
- brokers and triggers.

Events are buffered if the destination sink is unavailable.

Knative Eventing supports the following scenarios:

**Publish an event without creating a consumer**
You can send events to a broker as an HTTP POST, and use a sink binding to decouple the destination configuration from your application that is producing events.

**Consume an event without creating a publisher**
You can use a trigger to consume events from a broker based on event attributes. Your application receives events as an HTTP POST.

### 5.2.1. Event sinks

To enable delivery to multiple types of sinks, Knative Eventing defines the following generic interfaces that can be implemented by multiple Kubernetes resources:

**Addressable objects**
Able to receive and acknowledge an event delivered over HTTP to an address defined in the event’s status.address.url field. The Kubernetes Service object also satisfies the addressable interface.

**Callable objects**
Able to receive an event delivered over HTTP and transform it, returning 0 or 1 new events in the HTTP response payload. These returned events may be further processed in the same way that events from an external event source are processed.
CHAPTER 6. CREATING AND MANAGING SERVERLESS APPLICATIONS

6.1. 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.

6.2. 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.

6.2.1. 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.

6.2.2. 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.

6.3. 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

6.4. CREATING SERVERLESS APPLICATIONS USING YAML

To create a serverless application by using YAML, you must create a YAML file that defines a Service, then apply it by using oc apply.

Procedure

1. Create a YAML file, then copy the following example into the file:

   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!"

2. Navigate to the directory where the YAML file is contained, and deploy the application by applying the YAML file:

   $ oc apply -f <filename>
After the Service is created and the application is deployed, Knative creates an immutable Revision for this version of the application.

Knative also performs network programming to create a Route, Ingress, Service, and load balancer for your application and automatically scales your pods up and down based on traffic, including inactive pods.

6.5. 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>` always prints URLs using the http:// format.

Procedure

1. Find the application URL by entering:

   $ oc get ksvc <service_name>

   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>
</tbody>
</table>

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

   Example HTTP request

   $ curl http://hello-default.example.com

   Example HTTPS request

   $ curl https://hello-default.example.com

   Example output

   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.
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 command

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

Example output

Hello Serverless!

6.6. INTERACTING WITH A SERVERLESS APPLICATION USING HTTP2 AND GRPC

OpenShift Serverless supports only insecure or edge-terminated routes.

Insecure or edge-terminated routes do not support HTTP2 on OpenShift Container Platform. These routes also do not support gRPC because gRPC 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. Find the ingress gateway’s public address:

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

   Example output:

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>kourier</td>
<td>LoadBalancer</td>
<td>172.30.51.103</td>
<td>a83e86291bcd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com 80:31380/TCP,443:31390/TCP 67m</td>
</tr>
</tbody>
</table>

   The public address is surfaced in the `EXTERNAL-IP` field. In this case, it would be a83e86291bcd11e993af02b7a65e514-33544245.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. The following example uses the information obtained from the steps in *Verifying your serverless application deployment*:

**Example command**

```bash
$ curl -H "Host: hello-default.example.com" a83e86291bddd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com
```

**Example output**

```
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:

```go
gRPC.Dial(
    "a83e86291bddd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com:80",
    grpc.WithAuthority("hello-default.example.com:80"),
    grpc.WithInsecure(),
)
```

**NOTE**

Ensure that you append the respective port, 80 by default, to both hosts as shown in the previous example.
CHAPTER 7. HIGH AVAILABILITY ON OPENS shifted 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.

7.1. CONFIGURING HIGH AVAILABILITY REPLICAS ON OPENS shifted 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 the KnativeServing.spec.highAvailability spec in the KnativeServing custom resource definition (CRD).

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**

```yaml
apiVersion: operator.knative.dev/v1alpha1
class: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
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 8. 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.

8.1. CONFIGURING JAEGER FOR USE WITH OPENSHIFT SERVERLESS

Prerequisites

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

- Cluster administrator permissions on an OpenShift Container Platform cluster.
- You have installed the OpenShift Serverless Operator and Knative Serving.
- You have installed the Jaeger Operator.

Procedure

1. Create and apply a Jaeger custom resource YAML file that contains the following sample YAML:

   Jaeger custom resource YAML

   ```yaml
   apiVersion: jaegertracing.io/v1
   kind: Jaeger
   metadata:
     name: jaeger
     namespace: default
   ```

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" 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

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

1. Get the hostname of the jaeger route:

   $ oc get route jaeger

   Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>HOST/PORT</th>
<th>PATH</th>
<th>SERVICES</th>
<th>PORT</th>
<th>TERMINATION</th>
</tr>
</thead>
</table>
   | WILDCARD | jaeger-default.apps.example.com jaeger-query <all> reencrypt None

2. Open the endpoint address in your browser to view the console.
CHAPTER 9. KNATIVE SERVING

9.1. USING KN TO COMPLETE SERVING TASKS

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.

9.1.1. 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

   OpenShift Container Platform 4.5 Serverless
   50
4. Update the service:

```
$ 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.

```
$ kn service describe hello
```

**Example output**

| Name:       hello |
|-------------|----------------|
| Namespace:  default |
| Age:        13m |
| URL:        http://hello-default.apps-crc.testing |

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

Conditions:
- ++ Ready 1m
- ++ ConfigurationsReady 1m
- ++ RoutesReady 1m

6. Delete the service:

```
$ kn service delete hello
```

**Example output**

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

7. Verify that the `hello` service is deleted by attempting to list it:

```
$ kn service list hello
```

CHAPTER 9. KNATIVE SERVING
9.1.2. 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</code></td>
<td>Hard limit of concurrent requests to be processed by a single replica.</td>
</tr>
<tr>
<td><code>--concurrency-target</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</code></td>
<td>Maximum number of replicas.</td>
</tr>
<tr>
<td><code>--min-scale</code></td>
<td>Minimum number of replicas.</td>
</tr>
</tbody>
</table>

9.1.3. 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:

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

9.1.3.1. Assigning tag revisions

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

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

NOTE

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

Procedure

- Use the following command:

  $ kn service update svc --tag @latest=candidate --tag svc-vwxyz=current
### 9.1.3.2. 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:

  ```bash
  $ 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.

### 9.1.3.3. 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.

### 9.1.3.4. 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>

### 9.2. 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** spec in the revision template.

### target annotation 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
```

### containerConcurrency spec used in a revision template

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: myapp
spec:
template:
  metadata:
    annotations:
      autoscaling.knative.dev/containerConcurrency: 100
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**.
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.

The `minScale` and `maxScale` annotations can be configured as follows in the revision template:

```yaml
spec:
template:
  metadata:
    annotations:
      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 the `minScale` annotation 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 OPENSOURCE SERVERLESS

9.3.1. About deploying cluster logging

OpenShift Container Platform cluster administrators can deploy cluster logging using the OpenShift Container Platform web console or CLI to install the Elasticsearch Operator and Cluster Logging Operator. When the operators are installed, you create a ClusterLogging 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.

The ClusterLogging CR defines a complete cluster logging environment that includes all the components of the logging stack to collect, store and visualize logs. The Cluster Logging Operator watches the Cluster Logging CR 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 ClusterLogging custom resource (CR), which you can use to create a cluster logging instance and configure your cluster logging environment.

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 ClusterLogging custom resource.

9.3.2.1. Configuring and Tuning Cluster Logging

You can configure your cluster logging environment by modifying the ClusterLogging custom resource deployed in the openshift-logging project.

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

Memory and CPU

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

```
spec:
  logStore:
    elasticsearch:
      resources:
        limits:
          cpu: 500m
          memory: 16Gi
        requests:
          cpu: 500m
          memory: 16Gi
        type: "elasticsearch"
```
logs:
  fluentd:
    resources:
      limits:
        cpu:
        memory:
      requests:
        cpu:
        memory:
        type: "fluentd"
visualization:
  kibana:
    resources:
      limits:
        cpu:
        memory:
      requests:
        cpu:
        memory:
        type: kibana
curation:
  curator:
    resources:
      limits:
      memory: 200Mi
      requests:
        cpu: 200m
        memory: 200Mi
      type: "curator"

Elasticsearch storage

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

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

This example specifies each data node in the cluster will be bound to a PVC that requests "200G" of "gp2" storage. Each primary shard will be backed by a single replica.
Omitting the `storage` block results in a deployment that includes ephemeral storage only.

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

**Elasticsearch replication policy**

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

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

**Curator schedule**

You specify the schedule for Curator in the **cron format**.

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

### 9.3.2.2. Sample modified ClusterLogging custom resource

The following is an example of a **ClusterLogging** custom resource modified using the options previously described.

**Sample modified ClusterLogging custom resource**

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

```bash
$ 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:

   ```bash
   $ 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.
Figure 9.2. Serverless application revisions

Revisions
- nodejs-ex2-4ic7h
- nodejs-ex2-4ic7h-deployment
- nodejs-ex2-7f75sf
- nodejs-ex2-7f75sf-deployment

Routes
- nodejs-ex2
  Location: http://nodejs-ex2.test-project.apps.gajemore.devcluster.openshift.com

Overview

Resources
10.1. EVENT DELIVERY WORKFLOWS USING BROKERS AND TRIGGERS

Brokers can be used in combination with triggers to deliver events from an event source to an event sink.

Events can be sent from an event source to a broker as an HTTP POST request.

After events have entered the broker, they can be filtered by CloudEvent attributes using triggers, and sent as an HTTP POST request to an event sink.

10.1.1. Creating a broker

OpenShift Serverless provides a default Knative broker that can be created by using the Knative CLI. You can also create the default broker by adding the `eventing.knative.dev/injection=enabled` label to a namespace if you are a cluster administrator, or by adding the `eventing.knative.dev/injection: enabled` annotation to a trigger if you are a developer.

IMPORTANT

Although both developers and cluster administrators can add a broker by injection, only cluster administrators can permanently delete brokers that were created using this method.

10.1.1.1. Creating a broker using the Knative CLI

Prerequisites

- The OpenShift Serverless Operator and Knative Eventing are installed on your OpenShift Container Platform cluster.
- You have installed the `kn` CLI.

Procedure

- Create the `default` broker:
  
  ```
  $ kn broker create default
  ```
Verification

1. Use the `kn` command to list all existing brokers:

   ```bash
   $ kn broker list
   ```

Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>URL</th>
<th>AGE</th>
<th>CONDITIONS</th>
<th>READY</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td><a href="http://broker-ingress.knative-eventing.svc.cluster.local/test/default">http://broker-ingress.knative-eventing.svc.cluster.local/test/default</a></td>
<td>45s</td>
<td>5 OK / 5</td>
<td>True</td>
</tr>
</tbody>
</table>

2. Optional: If you are using the OpenShift Container Platform web console, you can navigate to the Topology view in the Developer perspective, and observe that the broker exists:

   ![Topology view](image)

10.1.1.2. Creating a broker by annotating a trigger

You can create a broker by adding the `eventing.knative.dev/injection: enabled` annotation to a Trigger object.

**IMPORTANT**

If you create a broker by using the `eventing.knative.dev/injection: enabled` annotation, you cannot delete this broker without cluster administrator permissions. If you delete the broker without having a cluster administrator remove this annotation first, the broker is created again after deletion.

Prerequisites

- The OpenShift Serverless Operator and Knative Eventing are installed on your OpenShift Container Platform cluster.

Procedure

1. Create a Trigger object as a `.yaml` file that has the `eventing.knative.dev/injection: enabled` annotation:

   ```yaml
   apiVersion: eventing.knative.dev/v1
   kind: Trigger
   metadata:
   ```
Specify details about the event sink, or **subscriber**, that the trigger sends events to.

2. Apply the **.yaml** file:

   ```bash
   $ oc apply -f <filename>
   
   Verification
   
   You can verify that the broker has been created successfully by using the **oc** CLI, or by observing it in the **Topology** view in the web console.

   1. Use the **oc** command to get the broker:

   ```bash
   $ oc -n <namespace> get broker default
   
   Example output
   
<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>REASON</th>
<th>URL</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>True</td>
<td></td>
<td><a href="http://broker-ingress.knative-eventing.svc.cluster.local/test/default">http://broker-ingress.knative-eventing.svc.cluster.local/test/default</a></td>
<td>3m56s</td>
</tr>
</tbody>
</table>
   
   2. Navigate to the **Topology** view in the web console, and observe that the broker exists:

   ![Topology view](image)

10.1.1.3. Creating a broker by labeling a namespace

If you have cluster administrator permissions, you can create the **default** broker automatically by labeling a namespace.
Brokers created using this method will not be removed if you remove the label. You must manually delete them.

**Prerequisites**

- The OpenShift Serverless Operator and Knative Eventing are installed on your OpenShift Container Platform cluster.
- You have cluster administrator permissions for OpenShift Container Platform.

**Procedure**

- Label a namespace with `eventing.knative.dev/injection=enabled`:
  
  ```bash
  $ oc label namespace <namespace> eventing.knative.dev/injection=enabled
  ```

**Verification**

You can verify that the broker has been created successfully by using the `oc` CLI, or by observing it in the **Topology** view in the web console.

1. Use the `oc` command to get the broker:

   ```bash
   $ oc -n <namespace> get broker <broker_name>
   ```

   **Example command**

   ```bash
   $ oc -n default get broker default
   ```

   **Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>REASON</th>
<th>URL</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>True</td>
<td></td>
<td><a href="http://broker-ingress.knative-eventing.svc.cluster.local/test/default">http://broker-ingress.knative-eventing.svc.cluster.local/test/default</a></td>
<td>3m56s</td>
</tr>
</tbody>
</table>

2. Navigate to the **Topology** view in the web console, and observe that the broker exists:

**10.1.2. Managing brokers**
The `kn` CLI provides commands that can be used to list, describe, update, and delete brokers. Cluster administrators can also permanently delete a broker that was created using injection.

### 10.1.2.1. Listing existing brokers using the Knative CLI

**Prerequisites**
- The OpenShift Serverless Operator, Knative Serving and Knative Eventing are installed on your OpenShift Container Platform cluster.
- You have installed the `kn` CLI.

**Procedure**
- List all existing brokers:

  
  ```bash
  $ kn broker list
  ```

**Example output**

```
NAME      URL                                                                      AGE   CONDITIONS   READY
REASON     
default   http://broker-ingress.knative-eventing.svc.cluster.local/test/default   45s   5 OK / 5       True
```

### 10.1.2.2. Describing an existing broker using the Knative CLI

**Prerequisites**
- The OpenShift Serverless Operator, Knative Serving and Knative Eventing are installed on your OpenShift Container Platform cluster.
- You have installed the `kn` CLI.

**Procedure**
- Describe an existing broker:

  ```bash
  $ kn broker describe <broker_name>
  ```

**Example command using default broker**

```
$ kn broker describe default
```

**Example output**

```
Name:         default
Namespace:    default
Annotations:  eventing.knative.dev/broker.class=MTChannelBasedBroker,
              eventing.knative.dev/created ...
Age:          22s
```
10.1.2.3. Deleting a broker that was created by injection

Brokers created by injection, by using a namespace label or trigger annotation, are not deleted permanently if a developer removes the label or annotation. A user with cluster administrator permissions must manually delete these brokers.

Procedure

1. Remove the `eventing.knative.dev/injection=enabled` label from the namespace:

   ```bash
   $ oc label namespace <namespace> eventing.knative.dev/injection=enabled
   ```

   Removing the annotation prevents Knative from recreating the broker after you delete it.

2. Delete the broker from the selected namespace:

   ```bash
   $ oc -n <namespace> delete broker <broker_name>
   ```

Verification

- Use the `oc` command to get the broker:

  ```bash
  $ oc -n <namespace> get broker <broker_name>
  ```

Example command

```bash
$ oc -n default get broker default
```

Example output

```
No resources found.
Error from server (NotFound): brokers.eventing.knative.dev "default" not found
```

10.1.3. Filtering events using triggers

Using triggers enables you to filter events from the broker for delivery to event sinks.

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 later in this section.

• An available event consumer, such as a Knative service.

### 10.1.3.1. Creating a trigger using the Developer perspective

After you have created a broker, you can create a trigger in the web console Developer perspective.

**Prerequisites**

• The OpenShift Serverless Operator, Knative Serving, and Knative Eventing are installed on your OpenShift Container Platform cluster.

• You have logged in to the web console.

• You are in the Developer perspective.

• You have created a project or have access to a project with the appropriate roles and permissions to create applications and other workloads in OpenShift Container Platform.

• You have created a broker and a Knative service or other event sink to connect to the trigger.

**Procedure**

1. In the Developer perspective, navigate to the Topology page.

2. Hover over the broker that you want to create a trigger for, and drag the arrow. The Add Trigger option is displayed.
3. Click **Add Trigger**.

4. Select your sink as a **Subscriber** from the drop-down list.

5. Click **Add**.

**Verification**

- After the subscription has been created, it is represented as a line that connects the broker to the service in the **Topology** view:

![](image)

10.1.3.2. Deleting a trigger using the **Developer perspective**

You can delete triggers in the web console **Developer perspective**.

**Prerequisites**

- To delete a trigger using the **Developer** perspective, ensure that you have logged in to the web console.

**Procedure**

1. In the **Developer** perspective, navigate to the **Topology** page.

2. Click on the trigger that you want to delete.

3. In the **Actions** context menu, select **Delete Trigger**.
10.1.3.3. Creating a trigger using kn

You can create a trigger by using the **kn trigger create** command.

**Procedure**

- Create a trigger:

  ```bash
  $ kn trigger create <trigger_name> --broker <broker_name> --filter <key=value> --sink <sink_name>
  ``

  Alternatively, you can create a trigger and simultaneously create the **default** broker using broker injection:

  ```bash
  $ kn trigger create <trigger_name> --inject-broker --filter <key=value> --sink <sink_name>
  ``

  By default, triggers forward all events sent to a broker to sinks that are subscribed to that broker. Using the **--filter** attribute for triggers allows you to filter events from a broker, so that subscribers will only receive a subset of events based on your defined criteria.

10.1.3.4. Listing triggers using kn

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

**Procedure**

1. To print a list of available triggers, enter the following command:
Example output:

```
$ kn trigger list

NAME    BROKER    SINK           AGE   CONDITIONS   READY   REASON
email    default   ksvc:edisplay  4s    5 OK / 5     True
ping     default   ksvc:edisplay  32s   5 OK / 5     True

2. Optional: Print a list of triggers in JSON format:

$ kn trigger list -o json

10.1.3.4.1. Describing a trigger using kn

You can use the `kn trigger describe` command to print information about a trigger.

Procedure

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

  `$ kn trigger describe <trigger_name>`

Example output

```
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

10.1.3.4.2. 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 <trigger_name> --broker <broker_name> --filter type=dev.knative.samples.helloworld --sink ksvc:<service_name>

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 <trigger_name> --broker <broker_name> --sink ksvc:<service_name> \
  --filter type=dev.knative.samples.helloworld \
  --filter source=dev.knative.samples/helloworldsource \
  --filter myextension=my-extension-value

10.1.3.4.3. Updating a trigger using kn

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

**Procedure**

- Update a trigger:

  $ kn trigger update <trigger_name> --filter <key=value> --sink <sink_name> [flags]

  - You can update a trigger to filter exact event attributes that match incoming events. For example, using the **type** attribute:

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

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

    $ kn trigger update mytrigger --filter type-  

  - You can use the **--sink** parameter to change the event sink of a trigger:

    $ kn trigger update <trigger_name> --sink ksvc:my-event-sink

10.1.3.4.4. Deleting a trigger using kn

**Procedure**

- Delete a trigger:

  $ kn trigger delete <trigger_name>

**Verification**

1. List existing triggers:

   $ kn trigger list

2. Verify that the trigger no longer exists:
10.2. EVENT DELIVERY WORKFLOWS USING CHANNELS

Events can be sent from a source to a sink by using channels and subscriptions for event delivery.

Channels are custom resources 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, or other sinks, by using a subscription.

The default configuration for channel instances is defined in the `default-ch-webhook` config map. Developers can create their own channels directly by instantiating a supported `Channel` object.

10.2.1. Supported channel types

Currently, OpenShift Serverless only supports `InMemoryChannel` kind channels for development use, 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 are lost.

- `InMemoryChannel` channels do not implement event ordering, so two events that are received in the channel at the same time can be delivered to a subscriber in any order.

- If a subscriber rejects an event, there are no re-delivery attempts. Instead, the rejected event is sent to a `deadLetterSink` object if this exists, or is dropped.

10.2.1.1. Using the default development channel configuration

When you install Knative Eventing, the following `default-ch-webhook` config map is created automatically in the `knative-eventing` namespace:

```yaml
apiVersion: v1
class: ConfigMap
metadata:
  name: default-ch-webhook
```
namespace: knative-eventing
data:
default-ch-config: |
clusterDefault:
  apiVersion: messaging.knative.dev/v1
  kind: InMemoryChannel
namespaceDefaults:
  some-namespace:
    apiVersion: messaging.knative.dev/v1
    kind: InMemoryChannel
This config map can specify either a cluster-wide default channel implementation, or a namespace-specific default channel implementation. Configuring a namespace-specific default overrides any cluster-wide settings.

After you create a Channel object, a mutating admission webhook adds a set of spec.channelTemplate properties for the Channel object based on the default channel implementation.

Example Channel object with spec.channelTemplate properties

```yaml
apiVersion: messaging.knative.dev/v1
kind: Channel
metadata:
  name: example-channel
  namespace: default
spec:
  channelTemplate:
    apiVersion: messaging.knative.dev/v1
    kind: InMemoryChannel
The channel controller then creates the backing channel instance based on the spec.channelTemplate configuration.

NOTE
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 channel, 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.

10.2.2. Creating a development channel

Procedure

You can create a channel using the cluster default configuration by completing the following procedure.

1. Create a Channel object.
   a. Create a YAML file and copy the following sample code into it:
a. Create a YAML file and copy the following sample code into it:

```yaml
apiVersion: messaging.knative.dev/v1
kind: Channel
metadata:
  name: example-channel
namespace: default
```

b. Apply the YAML file by entering:

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

10.2.3. Creating a subscription

You can create a Subscription object to connect a channel to a sink. In the following procedure, the example sink is a Knative service named error-handler.

Procedure

1. Create a YAML file and copy the following sample code into it:

```yaml
apiVersion: messaging.knative.dev/v1beta1
kind: Subscription
metadata:
  name: my-subscription
namespace: default
spec:
  channel:
    apiVersion: messaging.knative.dev/v1beta1
    kind: Channel
    name: example-channel
delivery: 
  deadLetterSink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: error-handler
subscriber:
  ref:
    apiVersion: serving.knative.dev/v1
    kind: Service
    name: event-display
```

1. Name of the subscription.

2. Configuration settings for the channel that the subscription connects to.

3. Configuration settings for event delivery. This tells the subscription what happens to events that cannot be delivered to the subscriber. When this is configured, events that failed to be consumed are sent to the deadLetterSink. The event is dropped, no re-delivery of the event is attempted, and an error is logged in the system. The deadLetterSink value must be a Destination.

4. Configuration settings for the subscriber. This is the event sink that events are delivered to from the channel.
2. Apply the YAML file:

$ oc apply -f <FILENAME>
CHAPTER 11. EVENT SOURCES

11.1. GETTING STARTED WITH EVENT SOURCES

An event source is an object that links an event producer with an event sink, or consumer. A sink can be a Knative service, channel, or broker that receives events from an event source.

11.1.1. Creating event sources

Currently, OpenShift Serverless supports the following event source types:

API server source
Connects a sink to the Kubernetes API server by creating an APIServerSource object.

Ping source
Periodically sends ping events with a constant payload. A ping source can be used as a timer, and is created as a PingSource object.

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

You can create and manage Knative event sources using the Developer perspective in the OpenShift Container Platform web console, the kn CLI, or by applying YAML files.

- Create an API server source.
- Create a ping source.
- Create a sink binding.

11.1.2. Additional resources

- For more information about eventing workflows using OpenShift Serverless, see Knative Eventing architecture.

11.2. USING THE KNATIVE 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:

API server source
Connects a sink to the Kubernetes API server by creating an APIServerSource object.

Ping source
Periodically sends ping events with a constant payload. A ping source can be used as a timer, and is created as a PingSource object.

11.2.1. Listing available event source types using the Knative CLI

You can list the available event source types in the terminal by using the following command:
The default output for this command will look like:

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

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

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

### 11.2.2. Listing available event sources using the Knative CLI

You can list the available event sources in the terminal by entering the following command:

```
$ kn source list
```

#### 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>ksvc:eshow2</td>
<td>True</td>
</tr>
<tr>
<td>b1</td>
<td>SinkBinding</td>
<td>sinkbindings.sources.knative.dev</td>
<td>ksvc:eshow3</td>
<td>False</td>
</tr>
<tr>
<td>p1</td>
<td>PingSource</td>
<td>pingsources.sources.knative.dev</td>
<td>ksvc:eshow1</td>
<td>True</td>
</tr>
</tbody>
</table>

You can list event sources of a specific type only, by using the `--type` flag.

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

#### 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>p1</td>
<td>PingSource</td>
<td>pingsources.sources.knative.dev</td>
<td>ksvc:eshow1</td>
<td>True</td>
</tr>
</tbody>
</table>

### 11.2.3. Next steps

- See the documentation on [Using the API server source](#).
- See the documentation on [Using a ping source](#).

### 11.3. USING THE API SERVER SOURCE

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

#### 11.3.1. Prerequisites
You must have a current installation of OpenShift Serverless, including Knative Serving and Eventing, in your OpenShift Container Platform cluster. This can be installed by a cluster administrator.

Event sources need a service to use as an event **sink**. The sink is the service or application that events are sent to from the event source.

You must create or update a service account, role and role binding for the event source.

**NOTE**

Some 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.

### 11.3.2. Creating a service account, role, and binding for event sources

**Procedure**

1. Create a service account, role, and role binding for the event source 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
```
Change this namespace to the namespace that you have selected for installing the event source.

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.

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

   `$ oc apply --filename authentication.yaml`

11.3.3. Creating an ApiServerSource event source using the Developer perspective

Procedure

1. Navigate to the Add page and select Event Source.

2. In the Event Sources page, select ApiServerSource in the Type section.

   **Event Sources**

   Create an event source to register interest in a class of events from a particular system

   **Type**

   ![ApiServerSource, ContainerSource, CronJobSource]

   **ApiServerSource**

   Resource *

<table>
<thead>
<tr>
<th>APIVERSION</th>
<th>KIND</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>Event</td>
</tr>
</tbody>
</table>

   The list of resources to watch

3. Configure the ApiServerSource settings:

   a. Enter `v1` as the APIVERSION, and Event as the KIND.
b. Select the **Service Account Name** for the service account that you created.

```
Mode
Ref
```

The mode the receive adapter controller runs under

```
Service Account Name
Select a Service Account Name
```

The name of Service Account use to run this

---

**Sink**

```
Knative Service *
KSVC helloworld-go
```

Select a Service to sink to.

---

c. Select the targeted Knative service from the dropdown menu in **Sink → Knative Service**.

4. Click **Create**.

---

**Verification**

1. After you have created the ApiServerSource, you will see it connected to the service it is sinked to in the **Topology** view.

---

**11.3.4. Deleting the ApiServerSource**

**Procedure**

1. Navigate to the **Topology** view.

2. Right-click the ApiServerSource and select **Delete ApiServerSource**.
11.3.5. Using the API server source with the Knative CLI

This section describes the steps required to create an `ApiServerSource` object using `kn` commands.

**Prerequisites**

- Knative Serving and Eventing are installed on your cluster.
- You have created the `default` broker in the same namespace that the API server source will be installed in.
- You have the `kn` CLI installed.

**Procedure**

1. Create a service account, role, and role binding for the `ApiServerSource` object.
   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:
   ```
1 Change this namespace to the namespace that you have selected for installing the API server source.

NOTE

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

Create the service account, role binding, and cluster binding:

1. $ oc apply -f authentication.yaml

2. Create an `ApiServerSource` object that uses a broker as an event sink:

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

3. Create a Knative service that dumps incoming messages to its log:

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

4. Create a trigger to filter events from the `default` broker to the service:

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

5. Create events by launching a pod in the default namespace:
$ oc create deployment hello-node --image=quay.io/openshift-knative/knative-eventing-sources-event-display

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

$ kn source apiserver describe testevents

Example output

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/v1)
Resources:
  Kind: event (v1)
Controller: false
Conditions:
  OK TYPE AGE REASON
  ++ Ready 3m
  ++ Deployed 3m
  ++ SinkProvided 3m
  ++ SufficientPermissions 3m
  ++ EventTypesProvided 3m

Verification

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

1. Get the pods:

   $ oc get pods

2. View the message dumper function logs for the pods:

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

Example output

  cloudevents.Event
  Validation: valid
  Context Attributes,
    specversion: 1.0
    type: dev.knative.apiserver.resource.update
datacontenttype: application/json
...
11.3.6. Deleting an API server source using the Knative CLI

This section describes the steps used to delete the ApiServerSource object, trigger, service, service account, cluster role, and cluster binding using the `kn` and `oc` commands.

**Prerequisites**

- You must have the kn CLI installed.

**Procedure**

1. Delete the trigger:

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

2. Delete the service:

   ```
   $ kn service delete <service_name>
   ```

3. Delete the API server source:

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

4. Delete the service account, cluster role, and cluster binding:

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

11.3.7. Creating an API server source using YAML files

This guide describes the steps required to create an ApiServerSource object using YAML files.
### Prerequisites

- Knative Serving and Eventing are installed on your cluster.
- You have created the default broker in the same namespace as the one defined in the `ApiServerSource` object.

### Procedure

1. To create a service account, role, and role binding for the API server source, create a file named `authentication.yaml` and copy 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 the API server source.
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:

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

2. To create an `ApiServerSource` object, create a file named `k8s-events.yaml` and copy the following sample code into it:

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

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

```bash
$ oc apply -f k8s-events.yaml
```

3. To check that the API server source 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:latest
```

After you have created the `service.yaml` file, apply it:

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

4. To create a trigger from the `default` broker that filters events to the service created in the previous step, create a file named `trigger.yaml` and copy the following sample code into it:

```yaml
```
After you have created the `trigger.yaml` file, apply it:

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

5. To create events, launch a pod in the `default` namespace:

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

6. To check that the controller is mapped correctly, enter the following command and inspect the output:

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

**Example output**

```yaml
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
```
Verification

To verify that the Kubernetes events were sent to Knative, you can look at the message dumper function logs.

1. Get the pods:

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

   ```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",
   ...
   }
   ```

11.3.8. Deleting the API server source

This section describes how to delete the ApiServerSource object, trigger, service, service account, cluster role, and cluster binding by deleting their YAML files.
Procedure

1. Delete the trigger:
   $ oc delete -f trigger.yaml

2. Delete the service:
   $ oc delete -f service.yaml

3. Delete the API server source:
   $ oc delete -f k8s-events.yaml

4. Delete the service account, cluster role, and cluster binding:
   $ oc delete -f authentication.yaml

11.4. USING A PING SOURCE

A ping source is used to periodically send ping events with a constant payload to an event consumer. A ping source can be used to schedule sending events, similar to a timer, as shown in the example:

Example ping source

```yaml
apiVersion: sources.knative.dev/v1alpha2
kind: PingSource
metadata:
  name: test-ping-source
spec:
  schedule: "*/2 * * * *" 1
  jsonData: '{"message": "Hello world!"}' 2
  sink: 3
  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.

11.4.1. Creating a ping source using the Knative CLI

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

Prerequisites
You have Knative Serving and Eventing installed.

You have the `kn` CLI installed.

**Procedure**

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

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

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

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

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

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

**Example output**

- **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**

You can verify that the Kubernetes events were sent to the Knative event sink by looking at the logs of the sink pod.
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** object that sends a message every 2 minutes, so each message should be observed in a newly created pod.

1. Watch for new pods created:
   ```bash
   $ watch oc get pods
   ```
2. Cancel watching the pods using **Ctrl+C**, then look at the logs of the created pod:
   ```bash
   $ oc logs $(oc get pod -o name | grep event-display) -c user-container
   ```

**Example output**

```
$ watch oc get pods

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!"
   }
```

### 11.4.1. Remove the ping source

1. Delete the **PingSource** object:
   ```bash
   $ kn delete pingsources.sources.knative.dev test-ping-source
   ```
2. Delete the **event-display** service:
   ```bash
   $ kn delete service.serving.knative.dev event-display
   ```

### 11.4.2. Creating a ping source using YAML files

The following sections describe how to create, verify and remove a basic ping source 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 ping source is working, create a simple Knative service that dumps incoming messages to the log of the service.
   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:latest
   
   b. Apply the `service.yaml` file:

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

2. For each set of ping events that you want to request, create a `PingSource` object 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. Apply the `ping-source.yaml` file:

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

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

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

Example output

```yaml
apiVersion: sources.knative.dev/v1alpha2
kind: PingSource
metadata:
  annotations:
```
Verification

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

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 object 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

   Example output

   ```
   ☁ 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!" }
   ```

### 11.4.2.1. Remove the PingSource
1. Delete the service by entering the following command:

   $ oc delete --filename service.yaml

2. Delete the PingSource object by entering the following command:

   $ oc delete --filename ping-source.yaml

### 11.5. USING SINK BINDING

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

**IMPORTANT**

Before developers can use sink binding, cluster administrators must label the namespace that will be configured in the SinkBinding object with bindings.knative.dev/include:"true".

   $ oc label namespace <namespace> bindings.knative.dev/include=true

### 11.5.1. Using sink binding with the Knative CLI

This guide describes the steps required to create, manage, and delete a sink binding instance using the kn CLI.

**Prerequisites**

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

**IMPORTANT**

Before developers can use sink binding, cluster administrators must label the namespace that will be configured in the SinkBinding object with bindings.knative.dev/include:"true".

   $ oc label namespace <namespace> bindings.knative.dev/include=true

**Procedure**

1. To check that sink binding is set up correctly, create a Knative event display service, or event sink, that dumps incoming messages to its log:
$ kn service create event-display --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest

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

```
$ kn source binding create bind-heartbeat --subject Job:batch/v1:app=heartbeat-cron --sink ksvc: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:
  spec:
    # Run every minute
    schedule: "* * * * *"
    jobTemplate:
      metadata:
        labels:
          app: heartbeat-cron
          bindings.knative.dev/include: "true"
      spec:
        template:
          spec:
            restartPolicy: Never
            containers:
              - name: single-heartbeat
                image: quay.io/openshift-knative/knative-eventing-sources-heartbeats:latest
                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
```

```
IMPORTANT

To use sink binding, you must manually add a bindings.knative.dev/include=true label to your Knative resources.

For example, to add this label to a CronJob object, add the following lines to the Job resource YAML definition:

```
jobTemplate:
  metadata:
    labels:
      app: heartbeat-cron
      bindings.knative.dev/include: "true"
```

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

```
$ 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
```

Example output

```
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

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

- View the message dumper function logs:

```
$ oc get pods

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

```yaml
clodevents.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": ""
  }
```

11.5.2. Using sink binding with the YAML method

This guide describes the steps required to create, manage, and delete a sink binding instance 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.

**IMPORTANT**

Before developers can use sink binding, cluster administrators must label the namespace that will be configured in the `SinkBinding` object with `bindings.knative.dev/include:"true"`:

```
$ oc label namespace <namespace> bindings.knative.dev/include=true
```

Procedure

1. To check that sink binding 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`:

```yaml
apiVersion: serving.knative.dev/v1
```
After you have created the `service.yaml` file, apply it:

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

2. Create a **SinkBinding** object that directs events to the service.

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

   ```yaml
   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:

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

3. Create a **CronJob** object.

   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: "* * * * *"
   ```
To use sink binding, you must manually add a `bindings.knative.dev/include=true` label to your Knative resources.

For example, to add this label to a cron job instance, add the following lines to the `Job` resource YAML definition:

```yaml
jobTemplate:
  metadata:
    labels:
      app: heartbeat-cron
      bindings.knative.dev/include: "true"
  spec:
    template:
      spec:
        restartPolicy: Never
        containers:
        - name: single-heartbeat
          image: quay.io/openshift-knative/knative-eventing-sources-heartbeats:latest
          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:

```
$ oc apply -f 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
```

**Example output**

```yaml
spec:
  sink:
```
Verification

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

1. View the message dumper function logs:

   $ oc get pods

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

Example output

```
☁ 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
  hear: yes
  the: 42
  Data,
  {
    "id": 1,
    "label": ""
  }
```
12.1. USING SERVICE MESH WITH OPENSHIFT SERVERLESS

Using Service Mesh with OpenShift Serverless enables developers to configure additional networking and routing options that are not supported when using OpenShift Serverless with the default Kourier implementation. These options include setting custom domains, using TLS certificates, and using JSON Web Token authentication.

Prerequisites

1. Install the OpenShift Serverless Operator and Knative Serving.
2. Install Red Hat OpenShift Service Mesh.

Procedure

1. Add the default namespace to the ServiceMeshMemberRoll as a member:

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

   *IMPORTANT*
   
   Adding sidecar injection to Pods in system namespaces such as `knative-serving` and `knative-serving-ingress` is not supported.

2. Create a network policy that permits traffic flow from Knative system pods to Knative services:
   
   a. Add the `serving.knative.openshift.io/system-namespace=true` label to the `knative-serving` namespace:

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

   b. Add the `serving.knative.openshift.io/system-namespace=true` label to the `knative-serving-ingress` namespace:

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

   c. Copy the following `NetworkPolicy` resource into a YAML file:

      ```yaml
      apiVersion: networking.k8s.io/v1
      kind: NetworkPolicy
      metadata:
        name: allow-from-serving-system-namespace
      ```
12.1.1. Enabling sidecar injection for a Knative service

You can add an annotation to the Service resource YAML file to enable sidecar injection for a Knative service.

**Procedure**

1. Add the `sidecar.istio.io/inject="true"` annotation to the Service resource:

   ```yaml
   namespace: default
   spec:
     ingress:
     - from:
       - namespaceSelector:
         matchLabels:
           serving.knative.openshift.io/system-namespace: "true"
       podSelector: {}
     policyTypes:
     - Ingress
   
   1. Add the `sidecar.istio.io/inject="true"` annotation.
   
2. Apply the Service resource YAML file:

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

12.1.2. Additional resources

- For more information about Red Hat OpenShift Service Mesh, see Red Hat OpenShift Service Mesh architecture.

12.2. USING JSON WEB TOKEN AUTHENTICATION WITH SERVICE MESH AND OPENSHIFT SERVERLESS
You can enable JSON Web Token (JWT) authentication for Knative services by creating a policy in your serverless application namespace that only allows requests with valid JWTs.

**Prerequisites**

- Install **OpenShift Serverless**.
- Install **Red Hat OpenShift Service Mesh**.
- Configure **Service Mesh with OpenShift Serverless**, including enabling sidecar injection for your Knative services.

**IMPORTANT**

Adding sidecar injection to pods in system namespaces such as `knative-serving` and `knative-serving-ingress` is not supported.

**Procedure**

1. Copy the following **Policy** resource into a YAML file:

   **IMPORTANT**

   The paths `/metrics` and `/healthz` must be included in `excludedPaths` because they are accessed from system pods in the `knative-serving` namespace.

   ```yaml
   apiVersion: authentication.istio.io/v1alpha1
   kind: Policy
   metadata:
     name: default
   spec:
     origins:
     - jwt:
         issuer: testing@secure.istio.io
     triggerRules:
     - excludedPaths:
     - prefix: /metrics
       prefix: /healthz
     principalBinding: USE_ORIGIN
   ```

2. Apply the **Policy** resource YAML file:

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

**Verification**

1. If you try to use a **curl** request to get the Knative service URL, it is denied.

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

**Example output**
2. Verify the request with a valid JWT.
   a. Get the valid JWT token by entering the following command:

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

   b. Access the service by using the valid token in the `curl` request header:

   ```bash
   ```

   The request is now allowed.

   **Example output**
   ```
   Hello OpenShift!
   ```

12.2.1. Additional resources

- See [Red Hat OpenShift Service Mesh architecture](#).
- For more information about verifying Knative services and using `curl` requests, see [Verifying your serverless application deployment](#).

12.3. USING CUSTOM DOMAINS FOR KNATIVE SERVICES WITH SERVICE MESH

By default, Knative services have a fixed domain format:

```
<application_name>-<namespace>.<openshift_cluster_domain>
```

You can customize the domain for your Knative service by configuring the service as a private service and creating the required Service Mesh resources.

**Prerequisites**

- Install the [OpenShift Serverless Operator](#) and [Knative Serving](#).
- Install [Red Hat OpenShift Service Mesh](#).
- Complete the configuration steps in [Using Service Mesh with OpenShift Serverless](#).
- You can configure a custom domain for an existing Knative service, or create a new sample service. To create a new service, see [Creating and managing serverless applications](#).

12.3.1. Setting cluster availability to `cluster-local`
By default, Knative services are published to a public IP address. Being published to a public IP address means that Knative services are public applications, and have a publicly accessible URL.

Publicly accessible URLs are accessible from outside of the cluster. However, developers may need to build back-end services that are only be accessible from inside the cluster, known as private services. Developers can label individual services in the cluster with the `serving.knative.dev/visibility=cluster-local` label to make them private.

### Procedure

- Set the visibility for your service by adding the `serving.knative.dev/visibility=cluster-local` label:

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

### Verification

- Check that the URL for your service is now in the format `http://<service_name>.<namespace>.svc.cluster.local`, by entering the following command and reviewing the output:

  ```
  $ oc get ksvc
  ```

### Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>URL</th>
<th>LATESTCREATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td><a href="http://hello.default.svc.cluster.local">http://hello.default.svc.cluster.local</a></td>
<td>hello-tx2g7 hello-</td>
</tr>
<tr>
<td>tx2g7</td>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>

### 12.3.2. Creating necessary Service Mesh resources

#### Procedure

1. Create an Istio gateway to accept traffic.
   a. Create a YAML file, and copy the following YAML into it:

   ```yaml
   apiVersion: networking.istio.io/v1alpha3
   kind: Gateway
   metadata:
     name: default-gateway
   spec:
     selector:
       istio: ingressgateway
     servers:
       - port:
           number: 80
           name: http
           protocol: HTTP
     hosts:
   ```

   b. Apply the YAML file:
2. Create an Istio **VirtualService** object to rewrite the host header.
   a. Create a YAML file, and copy the following YAML into it:

   ```yaml
   apiVersion: networking.istio.io/v1alpha3
   kind: VirtualService
   metadata:
     name: hello
   spec:
     hosts:
       - custom-ksvc-domain.example.com
     gateways:
       - default-gateway
     http:
       - rewrite:
           authority: hello.default.svc
         route:
           - destination:
               host: hello.default.svc
               port:
                 number: 80
   ```

   ①② Your Knative service in the format `<service_name>..<namespace>.svc`.

   b. Apply the YAML file:

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

3. Create an Istio **ServiceEntry** object. This is required for OpenShift Serverless because Kourier is outside of the service mesh.
   a. Create a YAML file, and copy the following YAML into it:

   ```yaml
   apiVersion: networking.istio.io/v1alpha3
   kind: ServiceEntry
   metadata:
     name: hello.default.svc
   spec:
     hosts:
       - hello.default.svc
     location: MESH_EXTERNAL
     endpoints:
       - address: kourier-internal.knative-serving-ingress.svc
     ports:
       - number: 80
         name: http
         protocol: HTTP
     resolution: DNS
   ```

   ① Your Knative service in the format `<service_name>..<namespace>.svc`.

   ```bash
   $ oc apply -f <filename>
   ```
b. Apply the YAML file:

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

4. Create an OpenShift Container Platform route that points to the `VirtualService` object.
   a. Create a YAML file, and copy the following YAML into it:

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: hello
  namespace: istio-system
spec:
  host: custom-ksvc-domain.example.com
  port:
    targetPort: 8080
  to:
    kind: Service
    name: istio-ingressgateway
```

The OpenShift Container Platform route must be created in the same namespace as the ServiceMeshControlPlane. In this example, the ServiceMeshControlPlane is deployed in the `istio-system` namespace.

   a. Apply the YAML file:

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

---

12.3.3. Accessing a service using your custom domain

**Procedure**

1. Access the custom domain by using the `Host` header in a `curl` request. For example:

```bash
$ curl -H "Host: custom-ksvc-domain.example.com" http://<ip_address>
```

where `<ip_address>` is the IP address that the OpenShift Container Platform ingress router is exposed to.

**Example output**

```
Hello OpenShift!
```

12.3.4. Additional resources

- For more information about Red Hat OpenShift Service Mesh, see Understanding Red Hat OpenShift Service Mesh.
CHAPTER 13. USING METERING WITH OPENSHIFT 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.

13.1. INSTALLING METERING

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

13.2. DATASOURCES FOR KNATIVE SERVING METERING

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

13.2.1. Datasource for CPU usage in Knative Serving

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

Example 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!=""}
)
```

13.2.2. Datasource for memory usage in Knative Serving

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

Example YAML file
13.2.3. Applying data sources for Knative Serving metering

Procedure

- Apply the `ReportDataSources` resource as a YAML file:

  ```bash
  $ oc apply -f <datasource_name>.yaml
  ```

Example command

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

13.3. QUERIES FOR KNATIVE SERVING METERING

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

13.3.1. Query for CPU usage in Knative Serving

Example YAML file

```yaml
apiVersion: metering.openshift.io/v1
kind: ReportQuery
metadata:
  name: knative-service-cpu-usage
spec:
  inputs:
    - name: ReportingStart
type: time
  - name: ReportingEnd
type: time
  - default: knative-service-cpu-usage
default:
name: KnativeServiceCpuUsageDataSource
```
type: ReportDataSource
columns:
- name: period_start
type: timestamp
unit: date
- name: period_end
type: timestamp
unit: date
- name: namespace
type: varchar
unit: kubernetes_namespace
- name: service
type: varchar
- name: data_start
type: timestamp
unit: date
- name: data_end
type: timestamp
unit: date
- name: service_cpu_seconds
type: double
unit: cpu_core_seconds
query:
| SELECT
  timestamp '{| default .Report.ReportingEnd .Report Inputs.ReportingEnd | prestoTimestamp |}' AS period_end,
  labels["namespace"] as project,
  labels["label_serving_knative_dev_service"] as service,
  min("timestamp") as data_start,
  max("timestamp") as data_end,
  sum(amount * "timeprecision") AS service_cpu_seconds
FROM {| dataSourceTableName .Report Inputs.KnativeServiceCpuUsageDataSource |}
GROUP BY labels["namespace"],labels["label_serving_knative_dev_service"]

13.3.2. Query for memory usage in Knative Serving

Example 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
```
13.3.3. Applying queries for Knative Serving metering

Apply the query as a YAML file:

Example command

```bash
$ oc apply -f <query_name>.yaml
```

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

### 13.4. METERING REPORTS FOR KNATIVE SERVING

You can run metering reports against Knative Serving by creating `Report` resources. Before you run a

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

Example 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.

13.4.1. Running a metering report

1. Run the report by applying it as a YAML file:

   ```bash
   $ oc apply -f <report_name>.yaml
   
   $ oc get report
   
   NAME                        QUERY                       SCHEDULE   RUNNING    FAILED   LAST REPORT TIME       AGE
   knative-service-cpu-usage   knative-service-cpu-usage              Finished            2019-06-
   30T23:59:59Z   10h
   
   Example output
   ```
CHAPTER 14. INTEGRATIONS

14.1. USING NVIDIA GPU RESOURCES WITH SERVERLESS APPLICATIONS

NVIDIA supports experimental use of GPU resources on OpenShift Container Platform. See OpenShift Container Platform on NVIDIA GPU accelerated clusters for more information about setting up GPU resources on OpenShift Container Platform.

After GPU resources are enabled for your OpenShift Container Platform cluster, you can specify GPU requirements for a Knative service using the `kn` CLI.

**Procedure**

You can specify a GPU resource requirement when you create a Knative service using `kn`.

1. Create a service.

2. Set the GPU resource requirement limit to 1 by using `nvidia.com/gpu=1`:

   ```
   $ kn service create hello --image docker.io/knativesamples/hellocuda-go --limit nvidia.com/gpu=1
   ```

   A GPU resource requirement limit of 1 means that the service has 1 GPU resource dedicated. Services do not share GPU resources. Any other services that require GPU resources must wait until the GPU resource is no longer in use.

   A limit of 1 GPU also means that applications exceeding usage of 1 GPU resource are restricted. If a service requests more than 1 GPU resource, it is deployed on a node where the GPU resource requirements can be met.

**Updating GPU requirements for a Knative service using `kn`**

- Update the service. Change the GPU resource requirement limit to 3 by using `nvidia.com/gpu=3`:

  ```
  $ kn service update hello --limit nvidia.com/gpu=3
  ```

**14.1.1. Additional resources**

- For more information about limits, see Setting resource quotas for extended resources .