# Red Hat OpenShift Serverless 1.33

## About OpenShift Serverless

Introduction to OpenShift Serverless

Last Updated: 2024-08-12

# Red Hat OpenShift Serverless 1.33 About OpenShift Serverless

Introduction to OpenShift Serverless

## Legal Notice



## Abstract

This document provides an overview of OpenShift Serverless features, including Functions, Serving, and Eventing. It also includes release notes and details about how to get support.

# Table of Contents

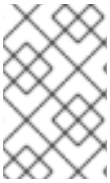
# CHAPTER 1. RELEASE NOTES

**NOTE**

For additional information about the OpenShift Serverless life cycle and supported platforms, refer to the OpenShift Operator Life Cycles.

Release notes contain information about new and deprecated features, breaking changes, and known issues. The following release notes apply for the most recent OpenShift Serverless releases on OpenShift Container Platform.

For an overview of OpenShift Serverless functionality, see About OpenShift Serverless.



**NOTE**

OpenShift Serverless is based on the open source Knative project.

For details about the latest Knative component releases, see the Knative blog.

## 1.1. ABOUT API VERSIONS

API versions are an important measure of the development status of certain features and custom resources in OpenShift Serverless. Creating resources on your cluster that do not use the correct API version can cause issues in your deployment.

The OpenShift Serverless Operator automatically upgrades older resources that use deprecated versions of APIs to use the latest version. For example, if you have created resources on your cluster that use older versions of the **ApiServerSource** API, such as **v1beta1**, the OpenShift Serverless Operator automatically updates these resources to use the **v1** version of the API when this is available and the **v1beta1** version is deprecated.

After they have been deprecated, older versions of APIs might be removed in any upcoming release. Using deprecated versions of APIs does not cause resources to fail. However, if you try to use a version of an API that has been removed, it will cause resources to fail. Ensure that your manifests are updated to use the latest version to avoid issues.

## 1.2. GENERALLY AVAILABLE AND TECHNOLOGY PREVIEW FEATURES

Features that are Generally Available (GA) are fully supported and are suitable for production use. Technology Preview (TP) features are experimental features and are not intended for production use. See the Technology Preview scope of support on the Red Hat Customer Portal for more information about TP features.

The following table provides information about which OpenShift Serverless features are GA and which are TP:

**Table 1.1. Generally Available and Technology Preview features tracker**

| Feature | 1.31 | 1.32 | 1.33 |
|---|---|---|---|
| OpenShift Serverless Logic | TP | TP | GA |

| Feature | 1.31 | 1.32 | 1.33 |
|---|---|---|---|
| ARM64 support | - | - | TP |
| Custom Metrics Autoscaler Operator (KEDA) | - | - | TP |
| kn event plugin | - | TP | TP |
| Pipelines-as-code | - | TP | TP |
| **new-trigger-filters** | - | TP | TP |
| Go function using S2I builder | - | TP | TP |
| Installing and using Serverless on single-node OpenShift | GA | GA | GA |
| Using Service Mesh to isolate network traffic with Serverless | TP | TP | TP |
| Serverless Logic | TP | TP | TP |
| Overriding **liveness** and **readiness** in functions | GA | GA | GA |
| **kn func** | GA | GA | GA |
| Quarkus functions | GA | GA | GA |
| Node.js functions | GA | GA | GA |
| TypeScript functions | GA | GA | GA |
| Python functions | TP | TP | TP |
| Service Mesh mTLS | GA | GA | GA |
| **emptyDir** volumes | GA | GA | GA |
| HTTPS redirection | GA | GA | GA |
| Kafka broker | GA | GA | GA |
| Kafka sink | GA | GA | GA |

| Feature | 1.31 | 1.32 | 1.33 |
|---|---|---|---|
| Init containers support for Knative services | GA | GA | GA |
| PVC support for Knative services | GA | GA | GA |
| Namespace-scoped brokers | TP | TP | TP |
| **multi-container support** | GA | GA | GA |

## 1.3. DEPRECATED AND REMOVED FEATURES

Some features that were Generally Available (GA) or a Technology Preview (TP) in previous releases have been deprecated or removed. Deprecated functionality is still included in OpenShift Serverless and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

For the most recent list of major functionality deprecated and removed within OpenShift Serverless, refer to the following table:

**Table 1.2. Deprecated and removed features tracker**

| Feature | 1.27 | 1.28 | 1.29 | 1.30 | 1.31 | 1.32 | 1.33 |
|---|---|---|---|---|---|---|---|
| Serving TLS for internal traffic | - | - | - | - | - | Removed | Removed |
| EventTypes **v1beta1** API | - | - | - | - | - | Deprecated | Deprecated |
| **domain-mapping** and **domain-mapping-webhook** deployments | - | - | - | - | - | Removed | Removed |
| Red Hat OpenShift Service Mesh with Serverless when Kourier is enabled | - | - | - | - | - | Deprecated | Deprecated |
| **NamespacedKafka** annotation | - | - | - | Deprecated | Deprecated | Deprecated | Deprecated |
| **enable-secret-informer-filtering** annotation | - | Deprecated | Deprecated | Deprecated | Deprecated | Deprecated | Deprecated |
| Serving and Eventing **v1alpha1** API | Deprecated | Deprecated | Removed | Removed | Removed | Removed | Removed |

| Feature | 1.27 | 1.28 | 1.29 | 1.30 | 1.31 | 1.32 | 1.33 |
|---|---|---|---|---|---|---|---|
| **kn func emit** (**kn func invoke** in 1.21+) | Removed | Removed | Removed | Removed | Removed | Removed | Removed |
| **KafkaBinding** API | Removed | Removed | Removed | Removed | Removed | Removed | Removed |

## 1.4. RED HAT OPENSHIFT SERVERLESS 1.33.2

OpenShift Serverless 1.33.2 is now available. Fixed issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes:

### 1.4.1. Fixed issues

- Previously, creating Knative installation resources like **KnativeServing** or **KnativeEventing** in a user namespace triggered an infinite reconciliation loop in the OpenShift Serverless Operator. This issue has been resolved by reintroducing an admission webhook that prevents the creation of Knative installation resources in any namespace other than **knative-serving** or **knative-eventing**.
- Previously, post-install batch jobs were removed after a certain period, leaving privileged service accounts unbound. This caused compliance systems to flag the issue. The problem has been resolved by retaining completed jobs, ensuring that service accounts remain bound.

## 1.5. RED HAT OPENSHIFT SERVERLESS 1.33

OpenShift Serverless 1.33 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes:

### 1.5.1. New features

- OpenShift Serverless now uses Knative Serving 1.12.
- OpenShift Serverless now uses Knative Eventing 1.12.
- OpenShift Serverless now uses Kourier 1.12.
- OpenShift Serverless now uses Knative (**kn**) CLI 1.12.
- OpenShift Serverless now uses Knative for Apache Kafka 1.12.
- The **kn func** CLI plugin now uses **func** 1.14.
- OpenShift Serverless Logic is now generally available (GA). This release includes an overview of OpenShift Serverless Logic; instructions on creating, running, and deploying workflows; and guidelines for the installation and uninstallation of OpenShift Serverless Logic Operator. Additionally, it includes steps for configuring OpenAPI services and endpoints, and techniques for troubleshooting the services. For more information, see OpenShift Serverless Logic overview.
  You can also refer to the additional documentation. For more details, see the Serverless Logic documentation.

- OpenShift Serverless on ARM64 is now available as Technology Preview.
- The **NamespacedKafka** annotation is now deprecated. Use the standard Kafka broker without data plane isolation instead.
- When enabling the automatic **EventType** auto-creation, you can now easily discover events available within the cluster. This functionality is available as a Developer Preview.
- You can now explore the Knative Eventing monitoring dashboards directly within the **Observe** tab of the developer view in the OpenShift Developer Console.
- You can now use the Custom Metrics Autoscaler Operator to autoscale Knative Eventing sources for Apache Kafka sources, defined by a **KafkaSource** object. This functionality is available as a Technology Preview feature, offering enhanced scalability and efficiency for Kafka-based event sources within Knative Eventing.
- You can now customize the internal Kafka topic properties when creating a Knative Broker with Kafka implementation. This improves efficiency and simplifies management.
- The new trigger filters feature is now available as a Technology Preview. These filters are enabled by default and allows users to specify a set of filter expressions, where each expression evaluates to either true or false for each event.

### 1.5.2. Known issues

- Due to different mount point permissions, direct upload on a cluster build does not work on IBM zSystems (s390x) and IBM Power (ppc64le).
- Building and deploying a function using Podman version 4.6 fails with the **invalid pull policy "1"** error.
  To work around this issue, use Podman version 4.5.

## 1.6. RED HAT OPENSHIFT SERVERLESS 1.32

OpenShift Serverless 1.32 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

### 1.6.1. New features

- OpenShift Serverless now uses Knative Serving 1.11.
- OpenShift Serverless now uses Knative Eventing 1.11.
- OpenShift Serverless now uses Kourier 1.11.
- OpenShift Serverless now uses Knative (**kn**) CLI 1.11.
- OpenShift Serverless now uses Knative for Apache Kafka 1.11.
- The **kn func** CLI plugin now uses **func** 1.13.
- Serverless Logic, which is available as a Technology Preview (TP) feature, has been updated. See the Serverless Logic documentation for usage instructions.
- You can configure the OpenShift Serverless Functions readiness and liveness probe settings for the **user** container and **queue-proxy** container.

- OpenShift Serverless Functions now supports OpenShift Pipelines versions from **1.10** till **1.14** (latest). The older versions of OpenShift Pipelines are no longer compatible with OpenShift Serverless Functions.
- On-cluster function building, including using Pipelines as Code is now supported on IBM zSystems (s390x) and IBM Power (ppc64le) on OpenShift Data Foundation storage only.
- You can now subscribe a function to a set of events by using the **func subscribe** command. This links your function to **CloudEvent** objects defined by your filters and enables automated responses.
- The Knative Serving TLS encryption feature for internal traffic is now deprecated. It was a Tech Preview feature. The functionality with the **internal-encryption** configuration flag is no longer available and it will be replaced by new configuration flags in a future release.
- The secret filtering is enabled by default on the OpenShift Serverless Operator side. An environment variable **ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID=true**, is added by default to the **net-istio** and **net-kourier** controller pods.
- The **domain-mapping** and **domain-mapping-webhook** deployments functionality in the **knative-serving** namespace is now removed. They are now integrated with Serving Webhook and Serving Controller.
- If you set **spec.config.domain** field in the **KnativeServing** custom resource (CR), the default external domain will no longer auto-populates the **config-domain** config map in the **knative-serving** namespace. Now, you must configure the **config-domain** config map manually to ensure accurate domain settings.
- You can now use the gRPC health probe for **net-kourier** deployments. The the Kourier Controller now uses a standard Kubernetes gRPC health probe for both readiness and liveness, replacing its previous use of exec and custom commands. The **timeoutSeconds** value has been adjusted from 100 milliseconds to 1 second to ensure more reliable probe responses.
- The new trigger filters feature is now available as a Technology Preview. The new trigger filters are now enabled by default. It allows users to specify a set of filter expressions, where each expression evaluates to either true or false for each event.
- Knative Eventing now offers support for data in transit encryption (Eventing TLS) as a developer preview. You can configure Knative Eventing components to expose HTTPS addresses as well as add user-provided CA trust bundles to clients.
- OpenShift Serverless now supports custom OpenShift CA bundle injection for system components. For more information, see Configuring a custom PKI.
- You can now use the Custom Metrics Autoscaler Operator to autoscale Knative Eventing sources for Apache Kafka sources. This functionality is available as a developer preview, offering enhanced scalability and efficiency for Kafka-based event sources within Knative Eventing.
- You can now explore the Knative Eventing monitoring dashboards directly within the **Observe** tab of the Developer view in the OpenShift Developer Console.
- The support for EventTypes **v1beta1** in Knative shipped is deprecated in OpenShift Serverless 1.32. In OpenShift Serverless 1.32, the Knative CLI uses the EventType **v1beta2** API to facilitate the new reference model. In previous releases, the **kn** CLI is not backward compatible with the EventType API **v1beta1** and is limited to the **kn eventtypes** sub-commands group. Therefore, it is recommended to use a matching **kn** version for the best user experience.

### 1.6.2. Fixed issues

- The default CPU limit is now increased for **3scale-kourier-gateways** from **500m** to **1s**. When more than 500 Knative Service instances are created, it could lead to readiness and liveness probe failures in the **3scale-kourier-gateways** pod due to CPU resource exhaustion. This adjustment aims to reduce such failures and ensures smoother operation even under heavy loads.

### 1.6.3. Known issues

- Due to different mount point permissions, direct upload on a cluster build does not work on IBM zSystems (s390x) and IBM Power (ppc64le).
- Building and deploying a function using Podman version 4.6 fails with the **invalid pull policy "1"** error.
  To work around this issue, use Podman version 4.5.

## 1.7. RED HAT OPENSHIFT SERVERLESS 1.31

OpenShift Serverless 1.31 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

### 1.7.1. New features

- OpenShift Serverless now uses Knative Serving 1.10.
- OpenShift Serverless now uses Knative Eventing 1.10.
- OpenShift Serverless now uses Kourier 1.10.
- OpenShift Serverless now uses Knative (**kn**) CLI 1.10.
- OpenShift Serverless now uses Knative for Apache Kafka 1.10.
- The **kn func** CLI plug-in now uses **func** 1.11.
- OpenShift Serverless multi-tenancy with Service Mesh is now available as a Technology Preview (TP) feature.
- Serverless Logic, which is available as a Technology Preview (TP) feature, has been updated. See the Serverless Logic documentation for usage instructions.
- OpenShift Serverless can now be installed and used on single-node OpenShift.
- You can now configure a persistent volume claim (PVC) for an existing **PersistentVolume** object to use with a Serverless function.
- When specifying Kourier for Ingress and using **DomainMapping**, the TLS for OpenShift Route is set to passthrough, and TLS is handled by the Kourier Gateway. Beginning with Serverless 1.31, it is possible to specify the enabled cipher suites on the side of the Kourier Gateway.
- Integrating Red Hat OpenShift Service Mesh with Serverless when Kourier is enabled is now deprecated. Use **net-istio** instead of **net-kourier** for Service Mesh integration.
  See the "Integrating Red Hat OpenShift Service Mesh with Serverless" section for details.

- The **PodDistruptionBudget** and **HorizontalPodAutoscaler** objects have been added for the **3scale-kourier-gateway** deployment.
    - **PodDistruptionBudget** is used to define the minimum availability requirements for pods in a deployment.
    - **HorizontalPodAutoscaler** is used to automatically scale the number of pods in the deployment based on demand or on your custom metrics.
- Now you can change the pattern for Apache Kafka topic names used by Knative brokers and channels for Apache Kafka.
- The **DomainMapping v1alpha1** custom resource definition (CRD) is now deprecated. Use **v1beta1** CRD instead.
- The **NamespacedKafka** annotation, which was a Technology Preview (TP) feature, is now deprecated in favor of the standard Kafka broker with no data plane isolation.

### 1.7.2. Fixed issues

- Previously, when deploying Knative Eventing with full Red Hat OpenShift Service Mesh integration and with **STRICT** peer authentication, the **PingSource** adapter metrics were unavailable.
  This has been fixed, and the **PingSource** adapter metrics are now collected using a different **job** and **service** label value. The previous value was **pingsource-mt-adapter**, the new value is **pingsource-mt-adapter-sm-service**.

## 1.8. RED HAT OPENSHIFT SERVERLESS 1.30.2

OpenShift Serverless 1.30.2 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

This release of OpenShift Serverless addresses Common Vulnerabilities and Exposures (CVEs), contains bug fixes, and is supported on OpenShift Container Platform 4.11 and later versions. Notably, this update addresses CVE-2023-44487 - HTTP/2 Rapid Stream Reset by disabling HTTP/2 transport on Serving, Eventing webhooks, and RBAC proxy containers.

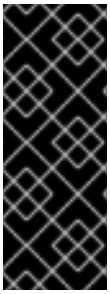
## 1.9. RED HAT OPENSHIFT SERVERLESS 1.30.1

OpenShift Serverless 1.30.1 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

This release of OpenShift Serverless addresses Common Vulnerabilities and Exposures (CVEs), contains bug fixes, and is supported on OpenShift Container Platform 4.11 and later versions.

## 1.10. RED HAT OPENSHIFT SERVERLESS 1.30

OpenShift Serverless 1.30 is now available. New features, updates, and known issues that relate to OpenShift Serverless on OpenShift Container Platform are included in the following.



**IMPORTANT**

OpenShift Container Platform 4.13 is based on Red Hat Enterprise Linux (RHEL) 9.2. RHEL 9.2 has not been submitted for Federal Information Processing Standards (FIPS) validation. Although Red Hat cannot commit to a specific timeframe, we expect to obtain FIPS validation for RHEL 9.0 and RHEL 9.2 modules, and later even minor releases of RHEL 9.x. Information on updates will be available in the Compliance Activities and Government Standards Knowledgebase article.

### 1.10.1. New features

- OpenShift Serverless now uses Knative Serving 1.9.
- OpenShift Serverless now uses Knative Eventing 1.9.
- OpenShift Serverless now uses Kourier 1.9.
- OpenShift Serverless now uses Knative (**kn**) CLI 1.9.
- OpenShift Serverless now uses Knative for Apache Kafka 1.9.
- The **kn func** CLI plug-in now uses **func** 1.10.1.
- OpenShift Serverless now runs on HyperShift-hosted clusters.
- OpenShift Serverless now runs on single-node OpenShift.
- Developer Experience around OpenShift Serverless is now available through OpenShift Toolkit, an OpenShift IDE Extension for the Visual Studio Code (VSCode). The extension can be installed from the VSCode Extension Tab and VSCode Marketplace. See the Marketplace page for the Visual Studio Code OpenShift Toolkit extension.
- OpenShift Serverless Functions nows supports Red Hat OpenShift Pipelines versions 1.10 and 1.11. Older versions of Red Hat OpenShift Pipelines are no longer compatible with OpenShift Serverless Functions.
- Serverless Logic is now available as a Technology Preview (TP) feature.
  See the Serverless Logic documentation for details.
- Beginning with OpenShift Serverless 1.30.0, the following runtime environments are supported on IBM zSystems using the s2i builder:
  - NodeJS
  - Python
  - TypeScript
  - Quarkus
- Eventing integration with Red Hat OpenShift Service Mesh is now available as a Technology Preview (TP) feature.
  The integration includes the following:
  - **PingSource**
  - **ApiServerSource**

    - Knative Source for Apache Kafka
    - Knative Broker for Apache Kafka
    - Knative Sink for Apache Kafka
    - **ContainerSource**
    - **SinkBinding**
    - **InMemoryChannel**
    - **KafkaChannel**
    - Channel-based Knative Broker

- Pipelines-as-code for OpenShift Serverless Functions is now available as a Technology Preview (TP).
- You can now configure the burst and queries per second (QPS) values for **net-kourier**.
- OpenShift Serverless Functions users now have the ability to override the **readiness** and **liveness** probe values in the **func.yaml** file for individual Quarkus functions.
  See "Functions development reference guide" for guidance on Quarkus, TypeScript, and Node.js functions.
- Beginning with OpenShift Serverless 1.30.0, Kourier controller and gateway manifests have the following limits and requests by default:
    - requests:
        - cpu: 200m
        - memory: 200Mi
    - limits:
        - cpu: 500m
        - memory: 500Mi
          See the "Overriding Knative Serving system deployment configurations" section of OpenShift Serverless documentation.
- The **NamespacedKafka** annotation, which was a Technology Preview (TP) feature, is now deprecated in favor of the standard Kafka broker with no data plane isolation.

### 1.10.2. Fixed issues

- Previously, the **3scale-kourier-gateway** pod was sending thousands of **net-kourier-controller** DNS queries daily. New queries were being sent for each **NXDOMAIN** reply. This continued until the correct DNS query was produced.
  The query now has the **net-kourier-controller.knative-serving-ingress.svc.<cluster domain>.** fully-qualified domain name (FQDN) by default, which solves the problem.

### 1.10.3. Known issues

- Building and deploying a function using Podman version 4.6 fails with the **invalid pull policy "1"** error.
  To work around this issue, use Podman version 4.5.
- On-cluster function building, including using Pipelines-as-code, is not supported on IBM zSystems and IBM Power.
- Buildpack builder is not supported on IBM zSystems and IBM Power.

**Additional resources**

- Overriding system deployment configurations

## 1.11. RED HAT OPENSHIFT SERVERLESS 1.29.1

OpenShift Serverless 1.29.1 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

This release of OpenShift Serverless addresses Common Vulnerabilities and Exposures (CVEs), contains bug fixes, and is supported on OpenShift Container Platform 4.10 and later versions.

### 1.11.1. Fixed issues

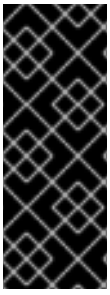
- Previously, the **net-kourier-controller** sometimes failed to start due to the liveness probe error. This has been fixed.

**Additional resources**

- Knowledgebase solution for the net-kourier-controller liveness probe error

## 1.12. RED HAT OPENSHIFT SERVERLESS 1.29

OpenShift Serverless 1.29 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.



**IMPORTANT**

OpenShift Container Platform 4.13 is based on Red Hat Enterprise Linux (RHEL) 9.2. RHEL 9.2 is yet to be submitted for Federal Information Processing Standards (FIPS) validation. Although Red Hat cannot commit to a specific timeframe, we expect to obtain FIPS validation for RHEL 9.0 and RHEL 9.2 modules, and later even minor releases of RHEL 9.x. Information on updates will be available in the Compliance Activities and Government Standards Knowledgebase article.

### 1.12.1. New features

- OpenShift Serverless now uses Knative Serving 1.8.
- OpenShift Serverless now uses Knative Eventing 1.8.
- OpenShift Serverless now uses Kourier 1.8.
- OpenShift Serverless now uses Knative (**kn**) CLI 1.8.

- OpenShift Serverless now uses Knative for Apache Kafka 1.8.
- The **kn func** CLI plug-in now uses **func** 1.10.
- Beginning with OpenShift Serverless 1.29, the different product versions are available as follows:
  - The latest release is available through the **stable** channel.
  - Releases older than the latest are available through the version-based channels. To use these, update the channel parameter in the subscription object YAML file from **stable** to the corresponding version-based channel, such as **stable-1.29**.

    This change allows you to receive updates not only for the latest release, but also for releases in the Maintenance phase.

    Additionally, you can lock the version of the Knative (**kn**) CLI. For details, see section "Installing the Knative CLI".
- You can now create OpenShift Serverless functions through developer console using OpenShift Container Platform Pipelines.
- Multi-container support for Knative Serving is now generally available (GA). This feature allows you to use a single Knative service to deploy a multi-container pod.
- OpenShift Serverless functions can now override the **readiness** and **liveness** probe values in the **func.yaml** file for individual Node.js and TypeScript functions.
- You can now configure your function to re-deploy automatically to the cluster when its source code changes in the GitHub repository. This allows for more seamless CI/CD integration.
- Eventing integration with Service Mesh is now available as developer preview feature. The integration includes: **PingSource**, **ApiServerSource**, Knative Source for Apache Kafka, Knative Broker for Apache Kafka, Knative Sink for Apache Kafka, **ContainerSource**, and **SinkBinding**.
- This release includes the upgraded Developer Preview for OpenShift Serverless Logic.
- API version **v1alpha1** of the Knative Operator Serving and Eventings CRDs has been removed. You need to use the **v1beta1** version instead. This does not affect the existing installations, because CRDs are updated automatically when upgrading the Serverless Operator.

### 1.12.2. Known issues

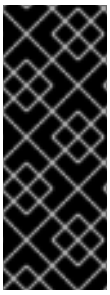
- When updating the secret specified in DomainMapping, simply updating the secret does not trigger the reconcile loop. You need to either rename the secret or delete the Knative Ingress resource to trigger the reconcile loop.
- Webhook Horizontal Pod Autoscaler (HPA) settings are overridden by the OpenShift Serverless Operator. As a result, it fails to scale for higher workloads. To work around this issue, manually set the initial replica value that corresponds to your workload.
- **KafkaSource** resources created before Red Hat OpenShift Serverless 1.27 get stuck when being deleted. To work around the issue, after starting to delete a **KafkaSource**, remove the finalizer from the resource.
- The **net-kourier-controller** might not be able to start due to the liveness probe error. You can work around the problem using the Knowledgebase solution.

**Additional resources**

- Knowledgebase solution for the net-kourier-controller liveness probe error
- Red Hat OpenShift Serverless Logic documentation

## 1.13. RED HAT OPENSHIFT SERVERLESS 1.28

OpenShift Serverless 1.28 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.



**IMPORTANT**

OpenShift Container Platform 4.13 is based on Red Hat Enterprise Linux (RHEL) 9.2. RHEL 9.2 is yet to be submitted for Federal Information Processing Standards (FIPS) validation. Although Red Hat cannot commit to a specific timeframe, we expect to obtain FIPS validation for RHEL 9.0 and RHEL 9.2 modules, and later even minor releases of RHEL 9.x. Information on updates will be available in the Compliance Activities and Government Standards Knowledgebase article.

### 1.13.1. New features

- OpenShift Serverless now uses Knative Serving 1.7.
- OpenShift Serverless now uses Knative Eventing 1.7.
- OpenShift Serverless now uses Kourier 1.7.
- OpenShift Serverless now uses Knative (**kn**) CLI 1.7.
- OpenShift Serverless now uses Knative broker implementation for Apache Kafka 1.7.
- The **kn func** CLI plug-in now uses **func** 1.9.1 version.
- Node.js and TypeScript runtimes for OpenShift Serverless Functions are now Generally Available (GA).
- Python runtime for OpenShift Serverless Functions is now available as a Technology Preview.
- Multi-container support for Knative Serving is now available as a Technology Preview. This feature allows you to use a single Knative service to deploy a multi-container pod.
- In OpenShift Serverless 1.29 or later, the following components of Knative Eventing will be scaled down from two pods to one:
  - **imc-controller**
  - **imc-dispatcher**
  - **mt-broker-controller**
  - **mt-broker-filter**
  - **mt-broker-ingress**

- The **serverless.openshift.io/enable-secret-informer-filtering** annotation for the Serving CR is now deprecated. The annotation is valid only for Istio, and not for Kourier.
  With OpenShift Serverless 1.28, the OpenShift Serverless Operator allows injecting the environment variable **ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID** for both **net-istio** and **net-kourier**.

  If you enable secret filtering, all of your secrets need to be labeled with **networking.internal.knative.dev/certificate-uid: "<id>"**. Otherwise, Knative Serving does not detect them, which leads to failures. You must label both new and existing secrets.

  In one of the following OpenShift Serverless releases, secret filtering will become enabled by default. To prevent failures, label your secrets in advance.

### 1.13.2. Known issues

- Currently, runtimes for Python are not supported for OpenShift Serverless Functions on IBM Power, IBM zSystems, and IBM® LinuxONE.
  Node.js, TypeScript, and Quarkus functions are supported on these architectures.

- On the Windows platform, Python functions cannot be locally built, run, or deployed using the Source-to-Image builder due to the **app.sh** file permissions.
  To work around this problem, use the Windows Subsystem for Linux.

- **KafkaSource** resources created before Red Hat OpenShift Serverless 1.27 get stuck when being deleted. To work around the issue, after starting to delete a **KafkaSource**, remove the finalizer from the resource.

## 1.14. RED HAT OPENSHIFT SERVERLESS 1.27

OpenShift Serverless 1.27 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

IMPORTANT

OpenShift Serverless 1.26 is the earliest release that is fully supported on OpenShift Container Platform 4.12. OpenShift Serverless 1.25 and older does not deploy on OpenShift Container Platform 4.12.

For this reason, before upgrading OpenShift Container Platform to version 4.12, first upgrade OpenShift Serverless to version 1.26 or 1.27.

### 1.14.1. New features

- OpenShift Serverless now uses Knative Serving 1.6.
- OpenShift Serverless now uses Knative Eventing 1.6.
- OpenShift Serverless now uses Kourier 1.6.
- OpenShift Serverless now uses Knative (**kn**) CLI 1.6.
- OpenShift Serverless now uses Knative Kafka 1.6.
- The **kn func** CLI plug-in now uses **func** 1.8.1.

- Namespace-scoped brokers are now available as a Technology Preview. Such brokers can be used, for instance, to implement role-based access control (RBAC) policies.
- **KafkaSink** now uses the **CloudEvent** binary content mode by default. The binary content mode is more efficient than the structured mode because it uses headers in its body instead of a **CloudEvent**. For example, for the HTTP protocol, it uses HTTP headers.
- You can now use the gRPC framework over the HTTP/2 protocol for external traffic using the OpenShift Route on OpenShift Container Platform 4.10 and later. This improves efficiency and speed of the communications between the client and server.
- API version **v1alpha1** of the Knative Operator Serving and Eventings CRDs is deprecated in 1.27. It will be removed in future versions. Red Hat strongly recommends to use the **v1beta1** version instead. This does not affect the existing installations, because CRDs are updated automatically when upgrading the Serverless Operator.
- The delivery timeout feature is now enabled by default. It allows you to specify the timeout for each sent HTTP request. The feature remains a Technology Preview.

### 1.14.2. Fixed issues

- Previously, Knative services sometimes did not get into the **Ready** state, reporting waiting for the load balancer to be ready. This issue has been fixed.

### 1.14.3. Known issues

- Integrating OpenShift Serverless with Red Hat OpenShift Service Mesh causes the **net-kourier** pod to run out of memory on startup when too many secrets are present on the cluster.
- Namespace-scoped brokers might leave **ClusterRoleBindings** in the user namespace even after deletion of namespace-scoped brokers.
  If this happens, delete the **ClusterRoleBinding** named **rbac-proxy-reviews-prom-rb-knative-kafka-broker-data-plane-{{.Namespace}}** in the user namespace.

## 1.15. RED HAT OPENSHIFT SERVERLESS 1.26

OpenShift Serverless 1.26 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

### 1.15.1. New features

- OpenShift Serverless Functions with Quarkus is now GA.
- OpenShift Serverless now uses Knative Serving 1.5.
- OpenShift Serverless now uses Knative Eventing 1.5.
- OpenShift Serverless now uses Kourier 1.5.
- OpenShift Serverless now uses Knative (**kn**) CLI 1.5.
- OpenShift Serverless now uses Knative Kafka 1.5.
- OpenShift Serverless now uses Knative Operator 1.3.

- The **kn func** CLI plugin now uses **func** 1.8.1.
- Persistent volume claims (PVCs) are now GA. PVCs provide permanent data storage for your Knative services.
- The new trigger filters feature is now available as a Developer Preview. It allows users to specify a set of filter expressions, where each expression evaluates to either true or false for each event.
  To enable new trigger filters, add the **new-trigger-filters: enabled** entry in the section of the **KnativeEventing** type in the operator config map:

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
...
...
spec:
  config:
    features:
      new-trigger-filters: enabled
...
```

- Knative Operator 1.3 adds the updated **v1beta1** version of the API for **operator.knative.dev**. To update from **v1alpha1** to **v1beta1** in your **KnativeServing** and **KnativeEventing** custom resource config maps, edit the **apiVersion** key:

  **Example KnativeServing custom resource config map**

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
...
```

  **Example KnativeEventing custom resource config map**

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
...
```

### 1.15.2. Fixed issues

- Previously, Federal Information Processing Standards (FIPS) mode was disabled for Kafka broker, Kafka source, and Kafka sink. This has been fixed, and FIPS mode is now available.

**Additional resources**

- Knative documentation on new trigger filters

## 1.16. RED HAT OPENSHIFT SERVERLESS 1.25.0

OpenShift Serverless 1.25.0 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

### 1.16.1. New features

- OpenShift Serverless now uses Knative Serving 1.4.
- OpenShift Serverless now uses Knative Eventing 1.4.
- OpenShift Serverless now uses Kourier 1.4.
- OpenShift Serverless now uses Knative (**kn**) CLI 1.4.
- OpenShift Serverless now uses Knative Kafka 1.4.
- The **kn func** CLI plugin now uses **func** 1.7.0.
- Integrated development environment (IDE) plugins for creating and deploying functions are now available for Visual Studio Code and IntelliJ.
- Knative Kafka broker is now GA. Knative Kafka broker is a highly performant implementation of the Knative broker API, directly targeting Apache Kafka.
  It is recommended to not use the MT-Channel-Broker, but the Knative Kafka broker instead.
- Knative Kafka sink is now GA. A **KafkaSink** takes a **CloudEvent** and sends it to an Apache Kafka topic. Events can be specified in either structured or binary content modes.
- Enabling TLS for internal traffic is now available as a Technology Preview.

### 1.16.2. Fixed issues

- Previously, Knative Serving had an issue where the readiness probe failed if the container was restarted after a liveness probe fail. This issue has been fixed.

### 1.16.3. Known issues

- The Federal Information Processing Standards (FIPS) mode is disabled for Kafka broker, Kafka source, and Kafka sink.
- The **SinkBinding** object does not support custom revision names for services.
- The Knative Serving Controller pod adds a new informer to watch secrets in the cluster. The informer includes the secrets in the cache, which increases memory consumption of the controller pod.
  If the pod runs out of memory, you can work around the issue by increasing the memory limit for the deployment.

**Additional resources for OpenShift Container Platform**

- Configuring TLS authentication

## 1.17. RED HAT OPENSHIFT SERVERLESS 1.24.0

OpenShift Serverless 1.24.0 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

### 1.17.1. New features

- OpenShift Serverless now uses Knative Serving 1.3.

- OpenShift Serverless now uses Knative Eventing 1.3.
- OpenShift Serverless now uses Kourier 1.3.
- OpenShift Serverless now uses Knative **kn** CLI 1.3.
- OpenShift Serverless now uses Knative Kafka 1.3.
- The **kn func** CLI plugin now uses **func** 0.24.
- Init containers support for Knative services is now generally available (GA).
- OpenShift Serverless logic is now available as a Developer Preview. It enables defining declarative workflow models for managing serverless applications.
- For OpenShift Container Platform, you can now use the cost management service with OpenShift Serverless.

### 1.17.2. Fixed issues

- Integrating OpenShift Serverless with Red Hat OpenShift Service Mesh causes the **net-istio-controller** pod to run out of memory on startup when too many secrets are present on the cluster.
  It is now possible to enable secret filtering, which causes **net-istio-controller** to consider only secrets with a **networking.internal.knative.dev/certificate-uid** label, thus reducing the amount of memory needed.
- The OpenShift Serverless Functions Technology Preview now uses Cloud Native Buildpacks by default to build container images.

### 1.17.3. Known issues

- The Federal Information Processing Standards (FIPS) mode is disabled for Kafka broker, Kafka source, and Kafka sink.
- In OpenShift Serverless 1.23, support for KafkaBindings and the **kafka-binding** webhook were removed. However, an existing **kafkabindings.webhook.kafka.sources.knative.dev MutatingWebhookConfiguration** might remain, pointing to the **kafka-source-webhook** service, which no longer exists.
  For certain specifications of KafkaBindings on the cluster, **kafkabindings.webhook.kafka.sources.knative.dev MutatingWebhookConfiguration** might be configured to pass any create and update events to various resources, such as Deployments, Knative Services, or Jobs, through the webhook, which would then fail.

  To work around this issue, manually delete **kafkabindings.webhook.kafka.sources.knative.dev MutatingWebhookConfiguration** from the cluster after upgrading to OpenShift Serverless 1.23:

  ```
  $ oc delete mutatingwebhookconfiguration kafkabindings.webhook.kafka.sources.knative.dev
  ```

## 1.18. RED HAT OPENSHIFT SERVERLESS 1.23.0

OpenShift Serverless 1.23.0 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

### 1.18.1. New features

- OpenShift Serverless now uses Knative Serving 1.2.
- OpenShift Serverless now uses Knative Eventing 1.2.
- OpenShift Serverless now uses Kourier 1.2.
- OpenShift Serverless now uses Knative (**kn**) CLI 1.2.
- OpenShift Serverless now uses Knative Kafka 1.2.
- The **kn func** CLI plugin now uses **func** 0.24.
- It is now possible to use the **kafka.eventing.knative.dev/external.topic** annotation with the Kafka broker. This annotation makes it possible to use an existing externally managed topic instead of the broker creating its own internal topic.
- The **kafka-ch-controller** and **kafka-webhook** Kafka components no longer exist. These components have been replaced by the **kafka-webhook-eventing** component.
- The OpenShift Serverless Functions Technology Preview now uses Source-to-Image (S2I) by default to build container images.

### 1.18.2. Known issues

- The Federal Information Processing Standards (FIPS) mode is disabled for Kafka broker, Kafka source, and Kafka sink.
- If you delete a namespace that includes a Kafka broker, the namespace finalizer may fail to be removed if the broker's **auth.secret.ref.name** secret is deleted before the broker.
- Running OpenShift Serverless with a large number of Knative services can cause Knative activator pods to run close to their default memory limits of 600MB. These pods might be restarted if memory consumption reaches this limit. Requests and limits for the activator deployment can be configured by modifying the **KnativeServing** custom resource:

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  deployments:
  - name: activator
    resources:
    - container: activator
      requests:
        cpu: 300m
        memory: 60Mi
      limits:
        cpu: 1000m
        memory: 1000Mi
```

- If you are using Cloud Native Buildpacks as the local build strategy for a function, **kn func** is unable to automatically start podman or use an SSH tunnel to a remote daemon. The

workaround for these issues is to have a Docker or podman daemon already running on the local development computer before deploying a function.

- On-cluster function builds currently fail for Quarkus and Golang runtimes. They work correctly for Node, Typescript, Python, and Springboot runtimes.

**Additional resources for OpenShift Container Platform**

- Source-to-Image

## 1.19. RED HAT OPENSHIFT SERVERLESS 1.22.0

OpenShift Serverless 1.22.0 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

### 1.19.1. New features

- OpenShift Serverless now uses Knative Serving 1.1.
- OpenShift Serverless now uses Knative Eventing 1.1.
- OpenShift Serverless now uses Kourier 1.1.
- OpenShift Serverless now uses Knative (**kn**) CLI 1.1.
- OpenShift Serverless now uses Knative Kafka 1.1.
- The **kn func** CLI plugin now uses **func** 0.23.
- Init containers support for Knative services is now available as a Technology Preview.
- Persistent volume claim (PVC) support for Knative services is now available as a Technology Preview.
- The **knative-serving**, **knative-serving-ingress**, **knative-eventing** and **knative-kafka** system namespaces now have the **knative.openshift.io/part-of: "openshift-serverless"** label by default.
- The **Knative Eventing - Kafka Broker/Trigger** dashboard has been added, which allows visualizing Kafka broker and trigger metrics in the web console.
- The **Knative Eventing - KafkaSink** dashboard has been added, which allows visualizing KafkaSink metrics in the web console.
- The **Knative Eventing - Broker/Trigger** dashboard is now called **Knative Eventing - Channel-based Broker/Trigger**.
- The **knative.openshift.io/part-of: "openshift-serverless"** label has substituted the **knative.openshift.io/system-namespace** label.
- Naming style in Knative Serving YAML configuration files changed from camel case (**ExampleName**) to hyphen style (**example-name**). Beginning with this release, use the hyphen style notation when creating or editing Knative Serving YAML configuration files.

### 1.19.2. Known issues

- The Federal Information Processing Standards (FIPS) mode is disabled for Kafka broker, Kafka source, and Kafka sink.

## 1.20. RED HAT OPENSHIFT SERVERLESS 1.21.0

OpenShift Serverless 1.21.0 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

### 1.20.1. New features

- OpenShift Serverless now uses Knative Serving 1.0
- OpenShift Serverless now uses Knative Eventing 1.0.
- OpenShift Serverless now uses Kourier 1.0.
- OpenShift Serverless now uses Knative (**kn**) CLI 1.0.
- OpenShift Serverless now uses Knative Kafka 1.0.
- The **kn func** CLI plugin now uses **func** 0.21.
- The Kafka sink is now available as a Technology Preview.
- The Knative open source project has begun to deprecate camel-cased configuration keys in favor of using kebab-cased keys consistently. As a result, the **defaultExternalScheme** key, previously mentioned in the OpenShift Serverless 1.18.0 release notes, is now deprecated and replaced by the **default-external-scheme** key. Usage instructions for the key remain the same.

### 1.20.2. Fixed issues

- In OpenShift Serverless 1.20.0, there was an event delivery issue affecting the use of **kn event send** to send events to a service. This issue is now fixed.
- In OpenShift Serverless 1.20.0 (**func** 0.20), TypeScript functions created with the **http** template failed to deploy on the cluster. This issue is now fixed.
- In OpenShift Serverless 1.20.0 (**func** 0.20), deploying a function using the **gcr.io** registry failed with an error. This issue is now fixed.
- In OpenShift Serverless 1.20.0 (**func** 0.20), creating a Springboot function project directory with the **kn func create** command and then running the **kn func build** command failed with an error message. This issue is now fixed.
- In OpenShift Serverless 1.19.0 (**func** 0.19), some runtimes were unable to build a function by using podman. This issue is now fixed.

### 1.20.3. Known issues

- Currently, the domain mapping controller cannot process the URI of a broker, which contains a path that is currently not supported.
  This means that, if you want to use a **DomainMapping** custom resource (CR) to map a custom domain to a broker, you must configure the **DomainMapping** CR with the broker's ingress service, and append the exact path of the broker to the custom domain:

**Example DomainMapping CR**

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
  name: <domain-name>
  namespace: knative-eventing
spec:
  ref:
    name: broker-ingress
    kind: Service
    apiVersion: v1
```

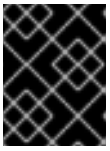
The URI for the broker is then **<domain-name>/<broker-namespace>/<broker-name>**.

## 1.21. RED HAT OPENSHIFT SERVERLESS 1.20.0

OpenShift Serverless 1.20.0 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

### 1.21.1. New features

- OpenShift Serverless now uses Knative Serving 0.26.
- OpenShift Serverless now uses Knative Eventing 0.26.
- OpenShift Serverless now uses Kourier 0.26.
- OpenShift Serverless now uses Knative (**kn**) CLI 0.26.
- OpenShift Serverless now uses Knative Kafka 0.26.
- The **kn func** CLI plugin now uses **func** 0.20.
- The Kafka broker is now available as a Technology Preview.



IMPORTANT

The Kafka broker, which is currently in Technology Preview, is not supported on FIPS.

- The **kn event** plugin is now available as a Technology Preview.
- The **--min-scale** and **--max-scale** flags for the **kn service create** command have been deprecated. Use the **--scale-min** and **--scale-max** flags instead.

### 1.21.2. Known issues

- OpenShift Serverless deploys Knative services with a default address that uses HTTPS. When sending an event to a resource inside the cluster, the sender does not have the cluster certificate authority (CA) configured. This causes event delivery to fail, unless the cluster uses globally accepted certificates.
  For example, an event delivery to a publicly accessible address works:

```
$ kn event send --to-url https://ce-api.foo.example.com/
```

On the other hand, this delivery fails if the service uses a public address with an HTTPS certificate issued by a custom CA:

```
$ kn event send --to Service:serving.knative.dev/v1:event-display
```

Sending an event to other addressable objects, such as brokers or channels, is not affected by this issue and works as expected.

- The Kafka broker currently does not work on a cluster with Federal Information Processing Standards (FIPS) mode enabled.
- If you create a Springboot function project directory with the **kn func create** command, subsequent running of the **kn func build** command fails with this error message:

```
[analyzer] no stack metadata found at path ''
[analyzer] ERROR: failed to : set API for buildpack 'paketo-buildpacks/ca-certificates@3.0.2':
buildpack API version '0.7' is incompatible with the lifecycle
```

As a workaround, you can change the **builder** property to **gcr.io/paketo-buildpacks/builder:base** in the function configuration file **func.yaml**.

- Deploying a function using the **gcr.io** registry fails with this error message:

```
Error: failed to get credentials: failed to verify credentials: status code: 404
```

As a workaround, use a different registry than **gcr.io**, such as **quay.io** or **docker.io**.

- TypeScript functions created with the **http** template fail to deploy on the cluster.
As a workaround, in the **func.yaml** file, replace the following section:

```
buildEnvs: []
```

with this:

```
buildEnvs:
- name: BP_NODE_RUN_SCRIPTS
  value: build
```

- In **func** version 0.20, some runtimes might be unable to build a function by using podman. You might see an error message similar to the following:

```
ERROR: failed to image: error during connect: Get
"http://%2Fvar%2Frun%2Fdocker.sock/v1.40/info": EOF
```

  - The following workaround exists for this issue:

    a. Update the podman service by adding **--time=0** to the service **ExecStart** definition:

    **Example service configuration**

    ```
    ExecStart=/usr/bin/podman $LOGGING system service --time=0
    ```

b. Restart the podman service by running the following commands:

```
$ systemctl --user daemon-reload
```

```
$ systemctl restart --user podman.socket
```

- Alternatively, you can expose the podman API by using TCP:

```
$ podman system service --time=0 tcp:127.0.0.1:5534 &
export DOCKER_HOST=tcp://127.0.0.1:5534
```

## 1.22. RED HAT OPENSHIFT SERVERLESS 1.19.0

OpenShift Serverless 1.19.0 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

### 1.22.1. New features

- OpenShift Serverless now uses Knative Serving 0.25.
- OpenShift Serverless now uses Knative Eventing 0.25.
- OpenShift Serverless now uses Kourier 0.25.
- OpenShift Serverless now uses Knative (**kn**) CLI 0.25.
- OpenShift Serverless now uses Knative Kafka 0.25.
- The **kn func** CLI plugin now uses **func** 0.19.
- The **KafkaBinding** API is deprecated in OpenShift Serverless 1.19.0 and will be removed in a future release.
- HTTPS redirection is now supported and can be configured either globally for a cluster or per each Knative service.

### 1.22.2. Fixed issues

- In previous releases, the Kafka channel dispatcher waited only for the local commit to succeed before responding, which might have caused lost events in the case of an Apache Kafka node failure. The Kafka channel dispatcher now waits for all in-sync replicas to commit before responding.

### 1.22.3. Known issues

- In **func** version 0.19, some runtimes might be unable to build a function by using podman. You might see an error message similar to the following:

```
ERROR: failed to image: error during connect: Get
"http://%2Fvar%2Frun%2Fdocker.sock/v1.40/info": EOF
```

  - The following workaround exists for this issue:

a. Update the podman service by adding **--time=0** to the service **ExecStart** definition:

**Example service configuration**

```
ExecStart=/usr/bin/podman $LOGGING system service --time=0
```

b. Restart the podman service by running the following commands:

```
$ systemctl --user daemon-reload
```

```
$ systemctl restart --user podman.socket
```

- Alternatively, you can expose the podman API by using TCP:

```
$ podman system service --time=0 tcp:127.0.0.1:5534 &
export DOCKER_HOST=tcp://127.0.0.1:5534
```

## 1.23. RED HAT OPENSHIFT SERVERLESS 1.18.0

OpenShift Serverless 1.18.0 is now available. New features, updates, and known issues that pertain to OpenShift Serverless on OpenShift Container Platform are included in the following notes.

### 1.23.1. New features

- OpenShift Serverless now uses Knative Serving 0.24.0.
- OpenShift Serverless now uses Knative Eventing 0.24.0.
- OpenShift Serverless now uses Kourier 0.24.0.
- OpenShift Serverless now uses Knative (**kn**) CLI 0.24.0.
- OpenShift Serverless now uses Knative Kafka 0.24.7.
- The **kn func** CLI plugin now uses **func** 0.18.0.
- In the upcoming OpenShift Serverless 1.19.0 release, the URL scheme of external routes will default to HTTPS for enhanced security.
  If you do not want this change to apply for your workloads, you can override the default setting before upgrading to 1.19.0, by adding the following YAML to your **KnativeServing** custom resource (CR):

```
...
spec:
  config:
    network:
      defaultExternalScheme: "http"
...
```

If you want the change to apply in 1.18.0 already, add the following YAML:

```
...
spec:
```

```
  config:
    network:
      defaultExternalScheme: "https"
...
```

- In the upcoming OpenShift Serverless 1.19.0 release, the default service type by which the Kourier Gateway is exposed will be **ClusterIP** and not **LoadBalancer**.
  If you do not want this change to apply to your workloads, you can override the default setting before upgrading to 1.19.0, by adding the following YAML to your **KnativeServing** custom resource (CR):

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

- You can now use **emptyDir** volumes with OpenShift Serverless. See the OpenShift Serverless documentation about Knative Serving for details.

- Rust templates are now available when you create a function using **kn func**.

### 1.23.2. Fixed issues

- The prior 1.4 version of Camel-K was not compatible with OpenShift Serverless 1.17.0. The issue in Camel-K has been fixed, and Camel-K version 1.4.1 can be used with OpenShift Serverless 1.17.0.

- Previously, if you created a new subscription for a Kafka channel, or a new Kafka source, a delay was possible in the Kafka data plane becoming ready to dispatch messages after the newly created subscription or sink reported a ready status.
  As a result, messages that were sent during the time when the data plane was not reporting a ready status, might not have been delivered to the subscriber or sink.

  In OpenShift Serverless 1.18.0, the issue is fixed and the initial messages are no longer lost. For more information about the issue, see Knowledgebase Article #6343981.

### 1.23.3. Known issues

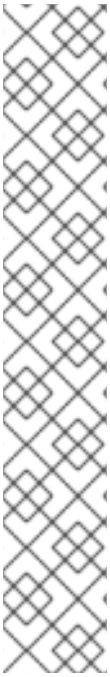
- Older versions of the Knative **kn** CLI might use older versions of the Knative Serving and Knative Eventing APIs. For example, version 0.23.2 of the **kn** CLI uses the **v1alpha1** API version. On the other hand, newer releases of OpenShift Serverless might no longer support older API versions. For example, OpenShift Serverless 1.18.0 no longer supports version **v1alpha1** of the **kafkasources.sources.knative.dev** API.

  Consequently, using an older version of the Knative **kn** CLI with a newer OpenShift Serverless might produce an error because the **kn** cannot find the outdated API. For example, version 0.23.2 of the **kn** CLI does not work with OpenShift Serverless 1.18.0.

  To avoid issues, use the latest **kn** CLI version available for your OpenShift Serverless release. For OpenShift Serverless 1.18.0, use Knative **kn** CLI 0.24.0.

# CHAPTER 2. OPENSHIFT SERVERLESS OVERVIEW

OpenShift Serverless provides Kubernetes native building blocks that enable developers to create and deploy serverless, event-driven applications on OpenShift Container Platform. OpenShift Serverless is based on the open source Knative project, which provides portability and consistency for hybrid and multi-cloud environments by enabling an enterprise-grade serverless platform.



**NOTE**

Because OpenShift Serverless releases on a different cadence from OpenShift Container Platform, the OpenShift Serverless documentation is now available as separate documentation sets for each minor version of the product.

The OpenShift Serverless documentation is available at https://docs.openshift.com/serverless/.

Documentation for specific versions is available using the version selector dropdown, or directly by adding the version to the URL, for example, https://docs.openshift.com/serverless/1.28.

In addition, the OpenShift Serverless documentation is also available on the Red Hat Portal at https://access.redhat.com/documentation/en-us/red_hat_openshift_serverless/.

For additional information about the OpenShift Serverless life cycle and supported platforms, refer to the Platform Life Cycle Policy.

## 2.1. ADDITIONAL RESOURCES

- Extending the Kubernetes API with custom resource definitions
- Managing resources from custom resource definitions
- What is serverless?

# CHAPTER 3. KNATIVE SERVING

Knative Serving supports developers who want to create, deploy, and manage cloud-native applications. It provides 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.

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.

## 3.1. KNATIVE SERVING RESOURCES

**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 custom resource. 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.

# CHAPTER 4. KNATIVE EVENTING

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 and event consumers.

Event producers create events, and event *sinks*, or consumers, receive events. Knative Eventing uses standard HTTP POST requests to send and receive events between event producers and sinks. These events conform to the CloudEvents specifications, which enables creating, parsing, sending, and receiving events in any programming language.

Knative Eventing supports the following use cases:

**Publish an event without creating a consumer**

You can send events to a broker as an HTTP POST, and use binding to decouple the destination configuration from your application that produces events.

**Consume an event without creating a publisher**

You can use a trigger to consume events from a broker based on event attributes. The application receives events as an HTTP POST.

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

**Addressable resources**

Able to receive and acknowledge an event delivered over HTTP to an address defined in the **status.address.url** field of the event. The Kubernetes **Service** resource also satisfies the addressable interface.

**Callable resources**

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.

## 4.1. USING THE KNATIVE BROKER FOR APACHE KAFKA

The Knative broker implementation for Apache Kafka provides integration options for you to use supported versions of the Apache Kafka message streaming platform with OpenShift Serverless. Kafka provides options for event source, channel, broker, and event sink capabilities.

Knative broker for Apache Kafka provides additional options, such as:

- Kafka source
- Kafka channel
- Kafka broker
- Kafka sink

## 4.2. ADDITIONAL RESOURCES

- Installing the **KnativeKafka** custom resource.
- Red Hat AMQ Streams documentation

- Red Hat AMQ Streams TLS and SASL on Apache Kafka documentation
- Event delivery

# CHAPTER 5. ABOUT OPENSHIFT SERVERLESS FUNCTIONS

OpenShift Serverless Functions enables developers to create and deploy stateless, event-driven functions as a Knative service on OpenShift Container Platform. The **kn func** CLI is provided as a plugin for the Knative **kn** CLI. You can use the **kn func** CLI to create, build, and deploy the container image as a Knative service on the cluster.

## 5.1. INCLUDED RUNTIMES

OpenShift Serverless Functions provides templates that can be used to create basic functions for the following runtimes:

- Quarkus
- Node.js
- TypeScript

## 5.2. NEXT STEPS

- Getting started with functions.

# CHAPTER 6. OPENSHIFT SERVERLESS LOGIC OVERVIEW

OpenShift Serverless Logic enables developers to define declarative workflow models that orchestrate event-driven, serverless applications.

You can write the workflow models in YAML or JSON format, which are ideal for developing and deploying serverless applications in cloud or container environments.

To deploy the workflows in your OpenShift Container Platform, you can use the OpenShift Serverless Logic Operator.

The following sections provide an overview of the various OpenShift Serverless Logic concepts.

## 6.1. EVENTS

An event state consists of one or more event definitions. Event definitions are combined to designate the **CloudEvent** types that the event state listens to. You can use the event state to start a new workflow instance upon the reception of a designated **CloudEvent**, or to pause the execution of an existing workflow instance until a designated **CloudEvent** is received.

In an event state definition, the **onEvents** property is used to group the **CloudEvent** types that might trigger the same set of **actions**. The **exclusive** property in an event definition indicates how an event match is calculated. If the value of **exclusive** property is **false**, then all **CloudEvent** types in the **eventRefs** array must be received for a match to occur. Otherwise, the reception of any referenced **CloudEvent** type is considered a match.

The following example shows event definitions, consisting of two CloudEvent types, including **noisy** and **silent**:

### Example of event definition

```
"events": [
   {
    "name": "noisyEvent",
    "source": "",
    "type": "noisy",
    "dataOnly" : "false"
   },
   {
    "name": "silentEvent",
    "source": "",
    "type": "silent"
   }
  ]
```

To indicate that an event match occurs when both **noisy** and **silent** CloudEvent types are received and to execute different actions for both CloudEvent types, define an event state containing both event definitions in separate **onEvent** items and set the **exclusive** property to false.

### Example of event state definition with multiple onEvent items

```
{
   "name": "waitForEvent",
   "type": "event",
   "onEvents": [
```

```
    {
      "eventRefs": [
        "noisyEvent"
      ],
      "actions": [
        {
          "functionRef": "letsGetLoud"
        }
      ]
    },
    {
      "eventRefs": [
        "silentEvent"
      ],
      "actions": [
        {
          "functionRef": "beQuiet"
        }
      ]
    }
  ]
  ,
  "exclusive": false
}
```

## 6.2. CALLBACKS

The Callback state performs an action and waits for an event that is produced as a result of the action before resuming the workflow. The action performed by a Callback state is an asynchronous external service invocation. Therefore, the Callback state is suitable to perform **fire&wait-for-result** operations.

From a workflow perspective, asynchronous service indicates that the control is returned to the caller immediately without waiting for the action to be completed. After the action is completed, a **CloudEvent** is published to resume the workflow.

### Example of Callback state in JSON format

```
{
      "name": "CheckCredit",
      "type": "callback",
      "action": {
          "functionRef": {
              "refName": "callCreditCheckMicroservice",
              "arguments": {
                  "customer": "${ .customer }"
              }
          }
      },
      "eventRef": "CreditCheckCompletedEvent",
      "timeouts": {
        "stateExecTimeout": "PT15M"
      },
      "transition": "EvaluateDecision"
}
```

### Example of Callback state in YAML format

```
name: CheckCredit
type: callback
action:
  functionRef:
    refName: callCreditCheckMicroservice
    arguments:
      customer: "${ .customer }"
eventRef: CreditCheckCompletedEvent
timeouts:
  stateExecTimeout: PT15M
transition: EvaluateDecision
```

The **action** property defines a function call that triggers an external activity or service. After the action executes, the Callback state waits for a **CloudEvent**, which indicates the completion of the manual decision by the called service.

After the completion callback event is received, the Callback state completes its execution and transitions to the next defined workflow state or completes workflow execution if it is an end state.

## 6.3. JQ EXPRESSIONS

Each workflow instance is associated with a data model. A data model consists of a **JSON** object regardless of whether the workflow file contains **YAML** or **JSON**. The initial content of the JSON object depends on how the workflow is started. If the workflow is created using the **CloudEvent**, then the workflow content is taken from the **data** property. If the workflow is started through an **HTTP POST** request, then the workflow content is taken from the request body.

The JQ expressions are used to interact with the data model. The supported expression languages include JsonPath and JQ. The JQ expression language is the default language. You can change the expression language to JsonPath using the **expressionLang** property.

### Example of JQ expression in functions

```
{
     "name": "max",
     "type": "expression",
     "operation": "{max: .numbers | max_by(.x), min: .numbers | min_by(.y)}"
   }
```

## 6.4. ERROR HANDLING

OpenShift Serverless Logic allows you to define **explicit** error handling. You can define inside of your workflow model what should happen if errors occur rather than some generic error handling entity. Explicit error handling enables you to handle the errors that might happen during the interactions between the workflow and external systems. When an error occurs, it changes the regular workflow sequence. In these cases, a workflow state transitions to an alternative state that can potentially handle the error, instead of transitioning to the predefined state.

Each workflow state can define error handling, which is related only to errors that might arise during its execution. Error handling defined in one state cannot be used to handle errors that happened during execution of another state during workflow execution.

Unknown errors that may arise during workflow state execution that are not explicitly handled within the workflow definition should be reported by runtime implementations and halt workflow execution.

### 6.4.1. Error definition

An error definition in a workflow is composed of the **name** and **code** parameters. The **name** is a short and natural language description of an error, such as **wrong parameter**. The **code** parameter helps the implementation to identify the error.

The **code** parameter is mandatory and the engine uses different strategies to map the provided value to an exception encountered at runtime. The available strategies include FQCN, error message, and status code.

During workflow execution, you must handle the the known workflow errors in the workflow top-level **errors** property. This property can be either a **string** type, meaning it can reference a reusable **JSON** or **YAML** definition file including the error definitions, or it can have an **array** type where you can define these checked errors inline in your workflow definition.

The following examples show definitions for both types:

#### Example of referencing a reusable JSON error definition file

```
{
"errors": "file://documents/reusable/errors.json"
}
```

#### Example of referencing a reusable YAML error definition file

```
errors: file://documents/reusable/errors.json
```

#### Example of defining workflow errors inline using a JSON file

```
{
"errors": [
  {
    "name": "Service not found error",
    "code": "404",
    "description": "Server has not found anything matching the provided service endpoint information"
  }
]
}
```

#### Example of defining workflow errors inline using a YAML file

```
errors:
  - name: Service not found error
    code: '404'
    description: Server has not found anything matching the provided service endpoint
      information
```

## 6.5. SCHEMA DEFINITIONS

OpenShift Serverless Logic supports two types of schema definitions: input schema definition and output schema definition.

### 6.5.1. Input schema definition

The **dataInputSchema** parameter validates the workflow data input against a defined **JSON** Schema. It is important to provide **dataInputSchema**, as it is used to verify if the provided workflow data input is correct before any workflow states are executed.

You can define a **dataInputSchema** as follows:

**Example of dataInputSchema definition**

```
"dataInputSchema": {
  "schema": "URL_to_json_schema",
  "failOnValidationErrors": false
}
```

The schema property is a URI, which holds the path to the JSON schema used to validate the workflow data input. The URI can be a classpath URI, a file, or an HTTP URL. If a classpath URI is specified, then the JSON schema file must be placed in the resources section of the project or any other directory included in the classpath.

The **failOnValidationErrors** is an optional flag that indicates the behavior adopted when the input data does not match the specified JSON schema. If not specified or set to **true**, an exception is thrown and flow execution fails. If set to **false**, the flow is executed and a log of level WARN with the validation errors is printed.

### 6.5.2. Output schema definition

Output schema definition is applied after workflow execution to verify that the output model has the expected format. It is also useful for Swagger generation purposes.

Similar to Input schema definition, you must specify the URL to the JSON schema, using **outputSchema** as follows:

**Example of outputSchema definition**

```
"extensions" : [ {
     "extensionid": "workflow-output-schema",
     "outputSchema": {
       "schema" : "URL_to_json_schema",
        "failOnValidationErrors": false
    }
 } ]
```

The same rules described for **dataInputSchema** are applicable for **schema** and **failOnValidationErrors**. The only difference is that the latter flag is applied after workflow execution.

## 6.6. CUSTOM FUNCTIONS

OpenShift Serverless Logic supports the **custom** function type, which enables the implementation to extend the function definitions capability. By combining with the **operation** string, you can use a list of predefined function types.

NOTE

Custom function types might not be portable across other runtime implementations.

### 6.6.1. Sysout custom function

You can use the **sysout** function for logging, as shown in the following example:

#### Example of sysout function definition

```
{
  "functions": [
    {
      "name": "logInfo",
      "type": "custom",
      "operation": "sysout:INFO"
    }
  ]
}
```

The string after the **:** is optional and is used to indicate the log level. The possible values are **TRACE**, **DEBUG**, **INFO**, **WARN**, and **ERROR**. If the value is not present, **INFO** is the default.

In the **state** definition, you can call the same **sysout** function as shown in the following example:

#### Example of a sysout function reference within a state

```
{
  "states": [
    {
      "name": "myState",
      "type": "operation",
      "actions": [
        {
          "name": "printAction",
          "functionRef": {
            "refName": "logInfo",
            "arguments": {
              "message": "\"Workflow model is \\(.)\""
            }
          }
        }
      ]
    }
  ]
}
```

In the previous example, the **message** argument can be a jq expression or a jq string using interpolation.

### 6.6.2. Java custom function

OpenShift Serverless Logic supports the **java** functions within an Apache Maven project, in which you define your workflow service.

The following example shows the declaration of a **java** function:

**Example of a java function declaration**

```
{
  "functions": [
    {
      "name": "myFunction", (1)
      "type": "custom", (2)
      "operation": "service:java:com.acme.MyInterfaceOrClass::myMethod" (3)
    }
  ]
}
```

1. **myFunction** is the function name.
2. **custom** is the function type.
3. **service:java:com.acme.MyInterfaceOrClass::myMethod** is the custom operation definition. In the custom operation definition, **service** is the reserved operation keyword, followed by the **java** keyword. **com.acme.MyInterfaceOrClass** is the FQCN (Fully Qualified Class Name) of the interface or implementation class, followed by the method name **myMethod**.

### 6.6.3. Knative custom function

OpenShift Serverless Logic provides an implementation of a custom function through the **knative-serving** add-on to invoke Knative services. It allows you to have a static URI, defining a Knative service, that is used to perform HTTP requests. The Knative service defined in the URI is queried in the current Knative cluster and translated to a valid URL.

The following example uses a deployed Knative service:

```
$ kn service list
NAME                          URL                                                 LATEST
AGE     CONDITIONS   READY   REASON
custom-function-knative-service   http://custom-function-knative-
service.default.10.109.169.193.sslip.io   custom-function-knative-service-00001   3h16m   3 OK / 3
True
```

You can declare a OpenShift Serverless Logic custom function using the Knative service name, as shown in the following example:

```
  "functions": [
    {
      "name": "greet", (1)
      "type": "custom", (2)
      "operation": "knative:services.v1.serving.knative.dev/custom-function-knative-service?
path=/plainJsonFunction", (3)
    }
  ]
```

1. **greet** is the function name.

2 **custom** is the function type.

3 In **operation**, you set the coordinates of the Knative service.

NOTE

This function sends a **POST** request. If you do not specify a path, OpenShift Serverless Logic uses the root path (/). You can also send **GET** requests by setting **method=GET** in the operation. In this case, the arguments are forwarded over a query string.

### 6.6.4. REST custom function

OpenShift Serverless Logic offers the **REST** custom type as a shortcut. When using custom rest, in the function definition, you specify the HTTP URI to be invoked and the HTTP method (get, post, patch, or put) to be used. This is done by using the **operation** string. When the function is invoked, you pass the request arguments as you do when using an OpenAPI function.

The following example shows the declaration of a **rest** function:

```
{
"functions": [
  {
    "name": "multiplyAllByAndSum", 1
    "type": "custom", 2
    "operation": "rest:post:/numbers/{multiplier}/multiplyByAndSum" 3
  }
 ]
}
```

1 **multiplyAllAndSum** is the function name.

2 **custom** is the function type.

3 **rest:post:/numbers/{multiplier}/multiplyByAndSum** is the custom operation definition. In the custom operation definition, **rest** is the reserved operation keyword that indicates this is a REST call, **post** is the HTTP method, and **/numbers/{multiplier}/multiplyByAndSum** is the relative endpoint.

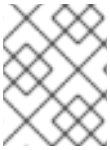
When using the relative endpoints, you must specify the host as a property. The format of the host property is **kogito.sw.functions.<function_name>**.host. In this example, **kogito.sw.functions.multiplyAllByAndSum.host** is the host property key. You can override the default port (80) if needed by specifying the **kogito.sw.functions.multiplyAllAndSum.port** property.

This endpoint expects as body a JSON object whose field **numbers** is an array of integers, multiplies each item in the array by **multiplier** and returns the sum of all the multiplied items.

## 6.7. TIMEOUTS

OpenShift Serverless Logic defines several timeouts configurations that you can use to configure maximum times for the workflow execution in different scenarios. You can configure how long a workflow can wait for an event to arrive when it is in a given state or the maximum execution time for the workflow.

Regardless of where it is defined, a timeout must be configured as an amount of time or duration, which starts when the referred workflow element becomes active. Timeouts use the **ISO 8601 data and time standard** to specify a duration of time and follow the format **PnDTnHnMn.nS**, with days considered to be exactly 24 hours. For example, **PT15M** configures 15 minutes, and **P2DT3H4M** defines 2 days, 3 hours, and 4 minutes.



NOTE

Month-based timeouts like **P2M**, or period of two months, are not valid since the month duration might vary. In that case, use **PT60D** instead.

### 6.7.1. Workflow timeout

To configure the maximum amount of time that a workflow can be running before being canceled, you can use the workflow timeouts. Once canceled, the workflow is considered to be finished, and is not accessible through a GET request anymore. Therefore, it behaves as if the interrupt was **true** by default.

Workflow timeouts are defined with the top-level **timeouts** property. It can have two types, **string** and **object**. The **string** type defines an URI that points to a JSON or YAML file containing the workflow timeout definitions. The **object** type, is used to define the timeout definitions inline.

For example, to cancel the workflow after an hour of execution, use the following configuration:

#### Example of workflow timeout

```
{
"id": "workflow_timeouts",
"version": "1.0",
"name": "Workflow Timeouts",
"description": "Simple workflow to show the workflowExecTimeout working",
"start": "PrintStartMessage",
"timeouts": {
  "workflowExecTimeout": "PT1H"
} ...
}
```

### 6.7.2. Event timeout

When you define a state in a workflow, you can use the **timeouts** property to configure the maximum time to complete this state. When that time is overdue, the state is considered timed-out, and the engine continues the execution from this state. The execution flow depends on the state type, for instance, a transition to a next state might be executed.

Event-based states can use the sub-property **eventTimeout** to configure the maximum time to wait for an event to arrive. This is the only property that is supported in current implementation.

Event timeouts support callback state timeout, switch state timeout, and event state timeout.

#### 6.7.2.1. Callback state timeout

The **Callback** state can be used when you must execute an action in general to call an external service, and wait for an asynchronous response in the form of an event.

Once the response event is consumed, the workflow continues the execution, in general moving to the next state defined in the transition property.

Since the **Callback** state halts the execution until the event is consumed, you can configure an eventTimeout for it, and in case the event does not arrive in the configured time duration, the workflow continues the execution moving to the next state defined in the transition.

**Example of Callback state with timeout**

```
{
 "name": "CallbackState",
 "type": "callback",
 "action": {
   "name": "callbackAction",
   "functionRef": {
     "refName": "callbackFunction",
     "arguments": {
       "input": "${\"callback-state-timeouts: \" + $WORKFLOW.instanceId + \" has executed the
callbackFunction.\"}"
     }
   }
 },
 "eventRef": "callbackEvent",
 "transition": "CheckEventArrival",
 "onErrors": [
   {
     "errorRef": "callbackError",
     "transition": "FinalizeWithError"
   }
 ],
 "timeouts": {
   "eventTimeout": "PT30S"
 }
}
```

### 6.7.2.2. Switch state timeout

You can use the **Switch** state when you need to take an action depending on certain conditions. These conditions can be based on the workflow data, **dataConditions**, or on events, **eventConditions**.

When you use the **eventConditions**, the workflow execution waits to make a decision until any of the configured events arrives and matches a condition. In this situation, you can configure an event timeout, that controls the maximum time to wait for an event to match the conditions.

If this time expires, the workflow moves to the state defined in the **defaultCondition** property.

**Example of Switch state with timeout**

```
{
   "name": "ChooseOnEvent",
   "type": "switch",
   "eventConditions": [
   {
       "eventRef": "visaApprovedEvent",
       "transition": "ApprovedVisa"
```

```
    },
    {
        "eventRef": "visaDeniedEvent",
        "transition": "DeniedVisa"
    }
    ],
        "defaultCondition": {
        "transition": "HandleNoVisaDecision"
    },
        "timeouts": {
        "eventTimeout": "PT5S"
    }
}
```

#### 6.7.2.3. Event state timeout

The **Event** state is used to wait for one or more events to be received by the workflow, execute a set of actions, and then continue the execution. If the Event state is a starting state, a new workflow instance is created.

The **timeouts** property is used for this state to configure the maximum time the workflow should wait for the configured events to arrive.

If this time is exceeded and the events are not received, the workflow moves to the state defined in the transition property or ends the workflow instance, in case of an end state, without performing any actions.

#### Example of Event state with timeout

```
{
  "name": "WaitForEvent",
  "type": "event",
  "onEvents": [
    {
      "eventRefs": [
        "event1"
      ],
      "eventDataFilter": {
        "data": "${ \"The event1 was received.\" }",
        "toStateData": "${ .exitMessage }"
      },
      "actions": [
        {
          "name": "printAfterEvent1",
          "functionRef": {
            "refName": "systemOut",
            "arguments": {
              "message": "${\"event-state-timeouts: \" + $WORKFLOW.instanceId + \" executing actions for
event1.\"}"
            }
          }
        }
      ]
    },
    {
```

```
      "eventRefs": [
        "event2"
      ],
      "eventDataFilter": {
        "data": "${ \"The event2 was received.\" }",
        "toStateData": "${ .exitMessage }"
      },
      "actions": [
        {
          "name": "printAfterEvent2",
          "functionRef": {
            "refName": "systemOut",
            "arguments": {
              "message": "${\"event-state-timeouts: \" + $WORKFLOW.instanceId + \" executing actions for
event2.\"}"
            }
          }
        }
      ]
    }
  ],
  "timeouts": {
    "eventTimeout": "PT30S"
  },
  "transition": "PrintExitMessage"
}
```

## 6.8. PARALLELISM

OpenShift Serverless Logic serializes the execution of parallel tasks. The word **parallel** does not indicate simultaneous execution, but it means that there is no logical dependency between the execution of branches. An inactive branch can start or resume the execution of a task without waiting for an active branch to be completed if the active branch suspends its execution. For example, an active branch may suspend its execution while waiting for an event reception.

A parallel state is a state that splits up the current workflow instance execution path into multiple paths, one for each branch. These execution paths are performed in parallel and are joined back into the current execution path depending on the defined **completionType** parameter value.

### Example of parallel workflow in JSON format

```
{
    "name":"ParallelExec",
    "type":"parallel",
    "completionType": "allOf",
    "branches": [
      {
        "name": "Branch1",
        "actions": [
          {
              "functionRef": {
                 "refName": "functionNameOne",
                 "arguments": {
                    "order": "${ .someParam }"
                 }
```

```
            }
          }
        ]
        },
        {
          "name": "Branch2",
          "actions": [
            {
              "functionRef": {
                "refName": "functionNameTwo",
                "arguments": {
                  "order": "${ .someParam }"
                }
              }
            }
          ]
        }
      ],
      "end": true
}
```

### Example of parallel workflow in YAML format

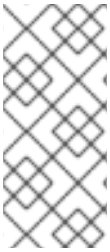
```
name: ParallelExec
type: parallel
completionType: allOf
branches:
- name: Branch1
  actions:
  - functionRef:
      refName: functionNameOne
      arguments:
        order: "${ .someParam }"
- name: Branch2
  actions:
  - functionRef:
      refName: functionNameTwo
      arguments:
        order: "${ .someParam }"
end: true
```

In the previous examples, the **allOf** defines all branches must complete execution before the state can transition or end. This is the default value if this parameter is not set.

# CHAPTER 7. OPENSHIFT SERVERLESS SUPPORT

If you experience difficulty with a procedure described in this documentation, visit the Red Hat Customer Portal at http://access.redhat.com. You can use the Red Hat Customer Portal to search or browse through the Red Hat Knowledgebase of technical support articles about Red Hat products. You can also submit a support case to Red Hat Global Support Services (GSS), or access other product documentation.

If you have a suggestion for improving this guide or have found an error, you can submit a Jira issue for the most relevant documentation component. Provide specific details, such as the section number, guide name, and OpenShift Serverless version so we can easily locate the content.



**NOTE**

The following section on defining cluster size requirements applies to these distributions:

- OpenShift Container Platform
- OpenShift Dedicated

## 7.1. ABOUT THE RED HAT KNOWLEDGEBASE

The Red Hat Knowledgebase provides rich content aimed at helping you make the most of Red Hat's products and technologies. The Red Hat Knowledgebase consists of articles, product documentation, and videos outlining best practices on installing, configuring, and using Red Hat products. In addition, you can search for solutions to known issues, each providing concise root cause descriptions and remedial steps.

## 7.2. SEARCHING THE RED HAT KNOWLEDGEBASE

In the event of an OpenShift Container Platform issue, you can perform an initial search to determine if a solution already exists within the Red Hat Knowledgebase.

**Prerequisites**

- You have a Red Hat Customer Portal account.

**Procedure**

1. Log in to the Red Hat Customer Portal .
2. In the main Red Hat Customer Portal search field, input keywords and strings relating to the problem, including:
   - OpenShift Container Platform components (such as **etcd**)
   - Related procedure (such as **installation**)
   - Warnings, error messages, and other outputs related to explicit failures
3. Click **Search**.
4. Select the **OpenShift Container Platform** product filter.
5. Select the **Knowledgebase** content type filter.

## 7.3. SUBMITTING A SUPPORT CASE

**Prerequisites**

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).
- You have a Red Hat Customer Portal account.
- You have a Red Hat standard or premium Subscription.

**Procedure**

1. Log in to the Red Hat Customer Portal and select **SUPPORT CASES → Open a case**.
2. Select the appropriate category for your issue (such as **Defect / Bug**), product (**OpenShift Container Platform**), and product version if this is not already autofilled).
3. Review the list of suggested Red Hat Knowledgebase solutions for a potential match against the problem that is being reported. If the suggested articles do not address the issue, click **Continue**.
4. Enter a concise but descriptive problem summary and further details about the symptoms being experienced, as well as your expectations.
5. Review the updated list of suggested Red Hat Knowledgebase solutions for a potential match against the problem that is being reported. The list is refined as you provide more information during the case creation process. If the suggested articles do not address the issue, click **Continue**.
6. Ensure that the account information presented is as expected, and if not, amend accordingly.
7. Check that the autofilled OpenShift Container Platform Cluster ID is correct. If it is not, manually obtain your cluster ID.
   - To manually obtain your cluster ID using the OpenShift Container Platform web console:
     a. Navigate to **Home → Dashboards → Overview**.
     b. Find the value in the **Cluster ID** field of the **Details** section.
   - Alternatively, it is possible to open a new support case through the OpenShift Container Platform web console and have your cluster ID autofilled.
     a. From the toolbar, navigate to **(?) Help → Open Support Case**.
     b. The **Cluster ID** value is autofilled.
   - To obtain your cluster ID using the OpenShift CLI (**oc**), run the following command:

     ```
     $ oc get clusterversion -o jsonpath='{.items[].spec.clusterID}{"\n"}'
     ```
8. Complete the following questions where prompted and then click **Continue**:
   - Where are you experiencing the behavior? What environment?

- When does the behavior occur? Frequency? Repeatedly? At certain times?
- What information can you provide around time-frames and the business impact?

9. Upload relevant diagnostic data files and click **Continue**.

It is recommended to include data gathered using the **oc adm must-gather** command as a starting point, plus any issue specific data that is not collected by that command.

1. Input relevant case management details and click **Continue**.
2. Preview the case details and click **Submit**.

## 7.4. GATHERING DIAGNOSTIC INFORMATION FOR SUPPORT

When you open 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.

### 7.4.1. 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, including:

- Resource definitions
- Service logs

By default, the **oc adm must-gather** command uses the default plugin image and writes into **./must-gather.local**.

Alternatively, you can collect specific information by running the command with the appropriate arguments as described in the following sections:

- To collect data related to one or more specific features, use the **--image** argument with an image, as listed in a following section.
  For example:

```
$ oc adm must-gather  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.13.0
```

- To collect the audit logs, use the **-- /usr/bin/gather_audit_logs** argument, as described in a following section.
  For example:

```
$ oc adm must-gather -- /usr/bin/gather_audit_logs
```

NOTE

Audit logs are not collected as part of the default set of information to reduce the size of the files.

When you run **oc adm must-gather**, a new pod with a random name is created in a new project 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.

For example:

```
NAMESPACE                  NAME             READY   STATUS      RESTARTS      AGE
...
openshift-must-gather-5drcj    must-gather-bklx4    2/2     Running     0          72s
openshift-must-gather-5drcj    must-gather-s8sdh    2/2     Running     0          72s
...
```

Optionally, you can run the **oc adm must-gather** command in a specific namespace by using the **--run-namespace** option.

For example:

```
$ oc adm must-gather --run-namespace <namespace> --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.13.0
```

### 7.4.2. 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.

#### Prerequisites

- Install the OpenShift CLI (**oc**).

#### 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-rhel8:1.14.0
```