Red Hat AMQ Streams 2.6

Release Notes for AMQ Streams 2.6 on OpenShift

Highlights of what's new and what's changed with this release of AMQ Streams on OpenShift Container Platform

Last Updated: 2023-12-19

Red Hat AMQ Streams 2.6 Release Notes for AMQ Streams 2.6 on OpenShift

Highlights of what's new and what's changed with this release of AMQ Streams on OpenShift Container Platform

Legal Notice

Abstract

The release notes summarize the new features, enhancements, and fixes introduced in the AMQ Streams 2.6 release.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

CHAPTER 1. FEATURES

AMQ Streams 2.6 introduces the features described in this section.

AMQ Streams 2.6 on OpenShift is based on Apache Kafka 3.6.0 and Strimzi 0.38.x.

NOTE

To view all the enhancements and bugs that are resolved in this release, see the AMQ Streams Jira project.

1.1. OPENSHIFT CONTAINER PLATFORM SUPPORT

AMQ Streams 2.6 is supported on OpenShift Container Platform 4.11 to 4.14.

For more information, see Chapter 9, *Supported Configurations*.

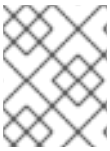
1.2. KAFKA 3.6.0 SUPPORT

AMQ Streams now supports and uses Apache Kafka version 3.6.0. Only Kafka distributions built by Red Hat are supported.

You must upgrade the Cluster Operator to AMQ Streams version 2.6 before you can upgrade brokers and client applications to Kafka 3.6.0. For upgrade instructions, see Upgrading AMQ Streams.

Refer to the Kafka 3.6.0 Release Notes for additional information.

Kafka 3.5.x is supported only for the purpose of upgrading to AMQ Streams 2.6.



NOTE

Kafka 3.6.0 provides access to KRaft mode, where Kafka runs without ZooKeeper by utilizing the Raft protocol. KRaft mode is available as a Developer Preview.

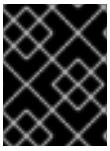
1.3. SUPPORTING THE V1BETA2 API VERSION

The **v1beta2** API version for all custom resources was introduced with AMQ Streams 1.7. For AMQ Streams 1.8, **v1alpha1** and **v1beta1** API versions were removed from all AMQ Streams custom resources apart from **KafkaTopic** and **KafkaUser**.

Upgrade of the custom resources to **v1beta2** prepares AMQ Streams for a move to Kubernetes CRD **v1**, which is required for Kubernetes 1.22.

If you are upgrading from an AMQ Streams version prior to version 1.7:

1. Upgrade to AMQ Streams 1.7
2. Convert the custom resources to **v1beta2**
3. Upgrade to AMQ Streams 1.8



IMPORTANT

You must upgrade your custom resources to use API version **v1beta2** before upgrading to AMQ Streams version 2.6.

1.3.1. Upgrading custom resources to v1beta2

To support the upgrade of custom resources to **v1beta2**, AMQ Streams provides an *API conversion tool*, which you can download from the AMQ Streams 1.8 software downloads page .

You perform the custom resources upgrades in two steps.

Step one: Convert the format of custom resources

Using the API conversion tool, you can convert the format of your custom resources into a format applicable to **v1beta2** in one of two ways:

- Converting the YAML files that describe the configuration for AMQ Streams custom resources
- Converting AMQ Streams custom resources directly in the cluster

Alternatively, you can manually convert each custom resource into a format applicable to **v1beta2**. Instructions for manually converting custom resources are included in the documentation.

Step two: Upgrade CRDs to v1beta2

Next, using the API conversion tool with the **crd-upgrade** command, you must set **v1beta2** as the *storage* API version in your CRDs. You cannot perform this step manually.

For more information, see Upgrading from an AMQ Streams version earlier than 1.7 .

1.4. STABLECONNECTIDENTITIES FEATURE GATE IS NOW ENABLED BY DEFAULT

The **StableConnectIdentities** feature gate moves to a beta level of maturity and is now enabled by default.

The feature allows you to use **StrimziPodSet** resources to manage Kafka Connect and Kafka MirrorMaker 2 pods instead of using **Deployment** resources. This helps to minimize the number of rebalances of connector tasks.

To disable the **StableConnectIdentities** feature gate, specify **-StableConnectIdentities** as a value for the **STRIMZI_FEATURE_GATES** environment variable in the Cluster Operator configuration.

Disabling the StableConnectIdentities feature gate

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: -StableConnectIdentities
```

See **StableConnectIdentities** feature gate.

1.5. BLOCK OR SKIP SCALE-DOWN OPERATIONS IF REPLICAS PRESENT

AMQ Streams now prevents scale-down of a cluster if brokers are still in use. By default, AMQ Streams performs a check to ensure that there are no partition replicas on a broker before initiating a scale-down operation on a Kafka cluster. If scale-down is blocked, you must either revert the operation or move the residual partitions as quickly as possible so that the Cluster Operator can continue to manage the cluster.

However, there may be scenarios where you want to bypass this blocking mechanism. For example, it might be useful to disable the check on busy clusters. To do this, you annotate the **Kafka** resource by setting **strimzi.io/skip-broker-scaledown-check="true"**.

See Skipping checks on scale-down operations .

1.6. SUPPORT FOR STOPPING CONNECTORS FROM RUNNING

You can now stop a Kafka Connect or Kafka MirrorMaker 2 connector from running. Compared to the paused state, where the connector and tasks stay instantiated, when stopped only the configuration of the connector is kept but nothing is actually running. Stopping a connector from running might be better suited for longer durations than just pausing. A paused connector is quicker to resume, but a stopped connector can free up memory and resources.

The **pause** property of the **KafkaConnectorSpec** schema and **KafkaMirrorMaker2ConnectorSpec** schema has been deprecated. Instead, both schemas now include a new **state** property. The **state** property allows you to configure one of the following values: **running**, **paused**, and **stopped**.

NOTE

Support for MirrorMaker 2 has a known issue . The issue will be fixed in the next release of AMQ Streams.

For example, if you want to stop a Kafka Connect connector resource, you change **state** to **stopped** in the configuration.

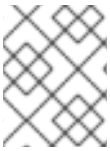
Example configuration for stopping a Kafka Connect connector

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-source-connector
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
  class: org.apache.kafka.connect.file.FileStreamSourceConnector
  tasksMax: 2
  config:
    file: "/opt/kafka/LICENSE"
    topic: my-topic
  state: stopped
    # ...
```

Example configuration for stopping a Kafka MirrorMaker 2 connector

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
```

```
  name: my-mirror-maker2
spec:
  version: 3.6.0
  replicas: 3
  connectCluster: "my-cluster-target"
  clusters:
    # ...
  mirrors:
  - sourceCluster: "my-cluster-source"
    targetCluster: "my-cluster-target"
    sourceConnector:
      tasksMax: 10
      autoRestart:
        enabled: true
      state: stopped
  # ...
```



NOTE

This feature requires at least Kafka 3.5.x to work. Only the pause operation works with older versions of Kafka.

Stop and pause operations are also supported by the Kafka Connect REST API (when the **KafkaConnector** operator is not enabled) using the following REST endpoints:

- **PUT /connectors/<connector_name>/stop**
- **PUT /connectors/<connector_name>/pause**

A **PUT /connectors/<connector_name>/resume** request restarts stopped and paused connectors.

See **KafkaConnectorSpec** schema reference and **KafkaMirrorMaker2ConnectorSpec** schema reference.

1.7. RUN PARALLEL ZOOKEEPER-BASED AND KRAFT-BASED KAFKA CLUSTERS

You can now run parallel Kafka clusters that operate in KRaft mode (using Kafka Raft metadata) or use ZooKeeper for cluster management.

As KRaft mode is a developer preview, to be able to use a cluster in KRaft mode, you must do the following:

1. Enable the **UseKRaft** and **KafkaNodePool** feature gates.
2. Ensure that the **Kafka** custom resource using KRaft mode has the annotation **strimzi.io/kraft: enabled**

When the **UseKRaft** feature gate is enabled and the annotation is set, the Kafka cluster is deployed without ZooKeeper and operates in KRaft mode. If you don't apply these settings, the Cluster Operator manages the **Kafka** resource as a ZooKeeper-based cluster.

To deploy a Kafka cluster in KRaft mode, you must now enable the **UseKRaft** and **KafkaNodePools** feature gates. As previews, both of these features are not intended for production. KRaft mode is supported only by using **KafkaNodePool** resources to manage the configuration of Kafka nodes. If you

are using KRaft mode, you can specify roles for all nodes in a node pool to operate as brokers, controllers, or both. If you are using ZooKeeper, you can still use node pools, but nodes must be set as brokers only.

See Section 4.1, "**UseKRaft** feature gate".

CHAPTER 2. ENHANCEMENTS

AMQ Streams 2.6 adds a number of enhancements.

2.1. KAFKA 3.6.0 ENHANCEMENTS

For an overview of the enhancements introduced with Kafka 3.6.0, refer to the Kafka 3.6.0 Release Notes.

2.2. PAUSE RECONCILIATION OF KAFKATOPIC RESOURCES IN UNIDIRECTIONAL MODE

It is now possible to pause reconciliation of **KafkaTopic** resources when the Topic Operator is operating in unidirectional mode (developer preview). To pause the reconciliation, you annotate the **KafkaTopic** resource with **strimzi.io/pause-reconciliation="true"**.

See Pausing reconciliation of custom resources.

2.3. UNIDIRECTIONAL TOPIC OPERATOR METRICS

Metrics are now available for the Topic Operator running in unidirectional mode (technology preview). Strimzi operators automatically expose Prometheus metrics by default.

In addition to the standard JVM metrics, we provide the following custom metrics:

- **strimzi.resources** (gauge): Number of custom resources the operator sees
- **strimzi.reconciliations** (counter): Number of reconciliations done by the operator for individual resources
- **strimzi.reconciliations.failed** (counter): Number of reconciliations done by the operator for individual resources which failed
- **strimzi.reconciliations.successful** (counter): Number of reconciliations done by the operator for individual resources which were successful
- **strimzi.reconciliations.duration** (timer): The time the reconciliation takes to complete
- **strimzi.reconciliations.paused** (gauge): Number of custom resources the operator sees but does not reconcile due to paused reconciliations
- **strimzi.reconciliations.locked** (counter): Number of reconciliations skipped because another reconciliation for the same resource was still running
- **strimzi.reconciliations.max.queue.size** (gauge): Max size recorded for the shared event queue
- **strimzi.reconciliations.max.batch.size** (gauge): Max size recorded for a single event batch

See Introducing metrics.

2.4. SUPPORT FOR MANUAL ROLLING UPDATE OF KAFKA CONNECT AND KAFKA MIRRORMAKER 2 PODS

With the **StableConnectIdentities** feature gate enabled, you can now trigger manual rolling updates of Kafka Connect and Kafka MirrorMaker 2 pods.

You can annotate the **StrimziPodSet** resource that manages the pods you want to manually update with **strimzi.io/manual-rolling-update=true** to perform the rolling update:

Annotating a StrimziPodSet resource

```
oc annotate strimzipodset <cluster_name>-connect strimzi.io/manual-rolling-update=true

oc annotate strimzipodset <cluster_name>-mirrormaker2 strimzi.io/manual-rolling-update=true
```

Or you can update the actual **Pod** resource:

Annotating a Pod resource

```
oc annotate pod <cluster_name>-connect-<index_number> strimzi.io/manual-rolling-update=true

oc annotate pod <cluster_name>-mirrormaker2-<index_number> strimzi.io/manual-rolling-update=true
```

See Performing a rolling update using a pod management annotation and Performing a rolling update using a pod annotation.

2.5. INFINITE AUTOMATIC RESTARTS OF CONNECTORS

When the auto-restart feature is enabled in **KafkaConnector** or **KafkaMirrorMaker2** custom resources, it now continues to restart the connectors indefinitely rather than stopping after 7 restarts, as previously.

If you want to use the original behavior, use the **.spec.autoRestart.maxRestarts** option to configure the maximum number of restarts.

Limiting the number of connector restarts

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-source-connector
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
  class: org.apache.kafka.connect.file.FileStreamSourceConnector
  tasksMax: 2
  autoRestart:
      enabled: true
      maxRestarts: 7
    # ...
```

See Deploying KafkaConnector resources and Configuring Kafka MirrorMaker 2.

2.6. CPU CAPACITY CONFIGURATION LOGIC IN CRUISE CONTROL

CPU capacity in Cruise Control is now determined using configuration values in the following order of precedence, with the highest priority first:

1. **Kafka.spec.cruiseControl.brokerCapacity.overrides.cpu** that define custom CPU capacity limits for individual brokers
2. **Kafka.cruiseControl.brokerCapacity.cpu** that defines custom CPU capacity limits for all brokers in the kafka cluster
3. **Kafka.spec.kafka.resources.requests.cpu** that defines the CPU resources that are reserved for each broker in the Kafka cluster.
4. **Kafka.spec.kafka.resources.limits.cpu** that defines the maximum CPU resources that can be consumed by each broker in the Kafka cluster.

This order of precedence is the sequence in which different configuration values are considered when determining the actual capacity limit for a Kafka broker. For example, broker-specific overrides take precedence over capacity limits for all brokers. If none of the CPU capacity configurations are specified, the default CPU capacity for a Kafka broker is set to 1 CPU core.

See **CruiseControlSpec** schema reference.

2.7. OAUTH 2.0 ACCEPT HEADER EXCLUDE OPTION

The **includeAcceptHeader** configuration property has been added as an OAuth 2.0 server-side and client-side configuration option. Some authorization servers have issues with clients sending **Accept: application/json** headers. By setting **includeAcceptHeader: false**, the header is not sent. The default setting is **true**.

See Configuring OAuth 2.0 support for Kafka brokers, Configuring OAuth 2.0 for Kafka components, and Configuring OAuth 2.0 authorization support.

2.8. ENHANCED FIPS SUPPORT

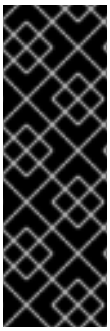
FIPS is now supported on the ppc64le (IBM Power) and s390x (IBM Z and IBM® LinuxONE) architectures.

2.9. ENHANCED SUPPORT FOR RUNNING RED HAT OPENSHIFT CONTAINER PLATFORM IN DISCONNECTED ENVIRONMENTS

You can now run Red Hat OpenShift Container Platform in disconnected environments on the ppc64le (IBM Power) and s390x (IBM Z and IBM® LinuxONE) architectures.

CHAPTER 3. TECHNOLOGY PREVIEWS

Technology Preview features included with AMQ Streams 2.6.



IMPORTANT

Technology Preview features are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete; therefore, Red Hat does not recommend implementing any Technology Preview features in production environments. This Technology Preview feature provides early access to upcoming product innovations, enabling you to test functionality and provide feedback during the development process. For more information about the support scope, see Technology Preview Features Support Scope.

3.1. KAFKANODEPOOLS FEATURE GATE

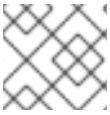
The **KafkaNodePools** feature gate and the new **KafkaNodePool** custom resource enables the configuration of different pools of Apache Kafka nodes. This feature gate is at an alpha level of maturity, which means that it is disabled by default, and should be treated as a technology preview.

A node pool refers to a distinct group of Kafka nodes within a Kafka cluster. The **KafkaNodePool** custom resource represents the configuration for nodes only in the node pool. Each pool has its own unique configuration, which includes mandatory settings such as the number of replicas, storage configuration, and a list of assigned roles. As you can assign roles to the nodes in a node pool, you can try the feature with a Kafka cluster that uses ZooKeeper for cluster management or KRaft mode.

To enable the **KafkaNodePools** feature gate, specify **+KafkaNodePools** in the **STRIMZI_FEATURE_GATES** environment variable in the Cluster Operator configuration.

Enabling the KafkaNodePools feature gate

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: +KafkaNodePools
```



NOTE

Drain Cleaner is not supported for the node pools feature.

See Configuring node pools.

3.2. UNIDIRECTIONALTOPICOPERATOR FEATURE GATE

The **UnidirectionalTopicOperator** feature gate introduces a unidirectional topic management mode. With unidirectional mode, you create Kafka topics using the **KafkaTopic** resource, which are then managed by the Topic Operator. This feature gate is at an alpha level of maturity, and should be treated as a technology preview.

To enable the **UnidirectionalTopicOperator** feature gate, specify **+UnidirectionalTopicOperator** in the **STRIMZI_FEATURE_GATES** environment variable in the Cluster Operator configuration.

Enabling the UnidirectionalTopicOperator feature gate

-

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: +UnidirectionalTopicOperator
```

Up to this release, the only way to use the Topic Operator to manage topics was in bidirectional mode, which is compatible with using ZooKeeper for cluster management. Unidirectional mode does not require ZooKeeper for cluster management, which is an important development as Kafka moves to using KRaft mode for managing clusters.

See Using the Topic Operator.

3.3. KAFKA STATIC QUOTA PLUGIN CONFIGURATION

Use the technology preview of the *Kafka Static Quota* plugin to set throughput and storage limits on brokers in your Kafka cluster. You enable the plugin and set limits by configuring the **Kafka** resource. You can set a byte-rate threshold and storage quotas to put limits on the clients interacting with your brokers.

Example Kafka Static Quota plugin configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    config:
      client.quota.callback.class: io.strimzi.kafka.quotas.StaticQuotaCallback
      client.quota.callback.static.produce: 1000000
      client.quota.callback.static.fetch: 1000000
      client.quota.callback.static.storage.soft: 400000000000
      client.quota.callback.static.storage.hard: 500000000000
      client.quota.callback.static.storage.check-interval: 5
```

See Setting limits on brokers using the Kafka Static Quota plugin .

CHAPTER 4. DEVELOPER PREVIEWS

Developer preview features included with AMQ Streams 2.6.

As a Kafka cluster administrator, you can toggle a subset of features on and off using feature gates in the Cluster Operator deployment configuration. The feature gates available as developer previews are at an alpha level of maturity and disabled by default.

IMPORTANT

Developer Preview features are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete; therefore, Red Hat does not recommend implementing any Developer Preview features in production environments. This Developer Preview feature provides early access to upcoming product innovations, enabling you to test functionality and provide feedback during the development process. For more information about the support scope, see Developer Preview Support Scope.

4.1. USEKRAFT FEATURE GATE

Apache Kafka is in the process of phasing out the need for ZooKeeper. With the **UseKRaft** feature gate enabled, you can try deploying a Kafka cluster in KRaft (Kafka Raft metadata) mode without ZooKeeper.

CAUTION

This feature gate is experimental, intended **only** for development and testing, and must not be enabled for a production environment.

To use KRaft mode, you must also use **KafkaNodePool** resources to manage the configuration of groups of nodes. To enable the **UseKRaft** feature gate, specify **+UseKRaft,+KafkaNodePools** as values for the **STRIMZI_FEATURE_GATES** environment variable in the Cluster Operator configuration.

Enabling the UseKRaft feature gate

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: +UseKRaft,+KafkaNodePools
```

Currently, the KRaft mode in AMQ Streams has the following major limitations:

- Moving from Kafka clusters with ZooKeeper to KRaft clusters or the other way around is not supported.
- Controller-only nodes cannot undergo rolling updates or be updated individually.
- Upgrades and downgrades of Apache Kafka versions or the Strimzi operator are not supported. Users might need to delete the cluster, upgrade the operator and deploy a new Kafka cluster.
- Only the *Unidirectional* Topic Operator is supported in KRaft mode. You can enable it using the **UnidirectionalTopicOperator** feature gate. The *Bidirectional* Topic Operator is not supported and when the **UnidirectionalTopicOperator** feature gate is not enabled, the **spec.entityOperator.topicOperator** property **must be removed** from the **Kafka** custom resource.

- JBOD storage is not supported. The **type: jbod** storage can be used, but the JBOD array can contain only one disk.

See **UseKRaft** feature gate and Feature gate releases.

CHAPTER 5. KAFKA BREAKING CHANGES

This section describes any changes to Kafka that required a corresponding change to AMQ Streams to continue to work.

5.1. USING KAFKA'S EXAMPLE FILE CONNECTORS

Kafka no longer includes the example file connectors **FileStreamSourceConnector** and **FileStreamSinkConnector** in its **CLASSPATH** and **plugin.path** by default. AMQ Streams has been updated so that you can still use these example connectors. The examples now have to be added to the plugin path like any connector.

AMQ Streams provides an example connector configuration file with the configuration required to deploy the file connectors as **KafkaConnector** resources:

- **examples/connect/source-connector.yaml**

See Deploying example KafkaConnector resources and Extending Kafka Connect with connector plugins.

CHAPTER 6. DEPRECATED FEATURES

The features deprecated in this release, and that were supported in previous releases of AMQ Streams, are outlined below.

6.1. ENVIRONMENT VARIABLE CONFIGURATION PROVIDER IS DEPRECATED

You can use configuration providers to load configuration data from external sources for all Kafka components, including producers and consumers.

Previously, you could enable the **secrets.io.strimzi.kafka.EnvVarConfigProvider** environment variable configuration provider using the **config.providers** properties in the **spec** configuration of a component. However, this provider is now deprecated and will be removed in the future. Therefore, it is recommended to update your implementation to use Kafka's own environment variable configuration provider (**org.apache.kafka.common.config.provider.EnvVarConfigProvider**) to provide configuration properties as environment variables.

Example configuration to enable the environment variable configuration provider

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  # ...
  config:
    # ...
    config.providers: env
    config.providers.env.class: org.apache.kafka.common.config.provider.EnvVarConfigProvider
  # ...
```

6.2. LISTENER STATUS TYPE PROPERTY REPLACED WITH NAME PROPERTY

The **type** property in the **ListenerStatus** schema has been deprecated and is no longer used. It has been replaced with the **name** property. The **name** property provides the name of a listener in the status of a **Kafka** resource.

Example Kafka status showing the name of a listener

```
status:
  clusterId: Y_RJQDGKRXmNF7fEcWldJQ
  conditions:
    - lastTransitionTime: '2023-11-29T14:59:37.113630Z'
      status: 'True'
      type: Ready
  kafkaVersion: 3.6.0
  listeners:
    # ...
    - addresses:
```

```
      - host: >-
          a8d4a6fb363bf44...
        port: 9094
    bootstrapServers: >-
      a8d4a6fb363bf447fb6e475...
    certificates:
      - |
        -----BEGIN CERTIFICATE-----

        -----END CERTIFICATE-----
    name: external3
  observedGeneration: 2
  operatorLastSuccessfulVersion: 2.6
# ...
```

6.3. PAUSE PROPERTY OF THE KAFKACONNECTORSPEC SCHEMA

The **pause** property of the **KafkaConnectorSpec** schema is now deprecated. Instead, a new **state** property allows you to choose a state value.

See Section 1.6, “Support for stopping connectors from running” .

6.4. STATEFULSET SUPPORT REMOVED

The **UseStrimziPodSets** feature gate is now permanently enabled and cannot be disabled. For this reason, support for **StatefulSet** resources to manage pods is no longer available.

The **StatefulSet** template properties in the **Kafka** custom resource (**.spec.zookeeper.template.statefulSet** and **.spec.kafka.template.statefulSet**) are deprecated and ignored. You should remove them from your custom resources.

6.5. JAVA 8 SUPPORT REMOVED IN AMQ STREAMS 2.4.0

Support for Java 8 was deprecated in Kafka 3.0.0 and AMQ Streams 2.0. Support for Java 8 was removed in AMQ Streams 2.4.0. This applies to all AMQ Streams components, including clients.

AMQ Streams supports Java 11 and Java 17. Use Java 11 or 17 when developing new applications. Plan to migrate any applications that currently use Java 8 to Java 11 or 17.

If you want to continue using Java 8 for the time being, AMQ Streams 2.2 provides Long Term Support (LTS). For information on the LTS terms and dates, see the AMQ Streams LTS Support Policy .

6.6. OPENTRACING

Support for **type: jaeger** tracing is now removed.

The Jaeger clients are now retired and the OpenTracing project archived. As such, we cannot guarantee their support for future Kafka versions.

OpenTelemetry has replaced OpenTracing for distributed tracing.

See Introducing distributed tracing.

6.7. ACL RULE CONFIGURATION

The **operation** property for configuring operations for ACL rules is deprecated. A new, more-streamlined configuration format using the **operations** property is now available.

New format for configuring ACL rules

```
authorization:
  type: simple
  acls:
    - resource:
        type: topic
        name: my-topic
      operations:
        - Read
        - Describe
        - Create
        - Write
```

The **operation** property for the old configuration format is deprecated, but still supported.

6.8. KAFKA MIRRORMAKER 2 IDENTITY REPLICATION POLICY

Identity replication policy is a feature used with MirrorMaker 2 to override the automatic renaming of remote topics. Instead of prepending the name with the source cluster's name, the topic retains its original name. This setting is particularly useful for active/passive backups and data migration scenarios.

To implement an identity replication policy, you must specify a replication policy class (**replication.policy.class**) in the MirrorMaker 2 configuration. Previously, you could specify the **io.strimzi.kafka.connect.mirror.IdentityReplicationPolicy** class included with the AMQ Streams **mirror-maker-2-extensions** component. However, this component is now deprecated and will be removed in the future. Therefore, it is recommended to update your implementation to use Kafka's own replication policy class (**org.apache.kafka.connect.mirror.IdentityReplicationPolicy**).

See Configuring Kafka MirrorMaker 2.

6.9. KAFKA MIRRORMAKER 1

Kafka MirrorMaker replicates data between two or more active Kafka clusters, within or across data centers. Kafka MirrorMaker 1 was deprecated in Kafka 3.0.0 and will be removed in Kafka 4.0.0. MirrorMaker 2 will be the only version available. MirrorMaker 2 is based on the Kafka Connect framework, connectors managing the transfer of data between clusters.

As a consequence, the AMQ Streams **KafkaMirrorMaker** custom resource which is used to deploy Kafka MirrorMaker 1 has been deprecated. The **KafkaMirrorMaker** resource will be removed from AMQ Streams when Kafka 4.0.0 is adopted.

If you are using MirrorMaker 1 (referred to as just *MirrorMaker* in the AMQ Streams documentation), use the **KafkaMirrorMaker2** custom resource with the **IdentityReplicationPolicy** class. MirrorMaker 2 renames topics replicated to a target cluster. **IdentityReplicationPolicy** configuration overrides the automatic renaming. Use it to produce the same active/passive unidirectional replication as MirrorMaker 1.

See Configuring Kafka MirrorMaker 2.

6.10. LISTENERSTATUS TYPE PROPERTY

The **type** property of **ListenerStatus** has been deprecated and will be removed in the future. **ListenerStatus** is used to specify the addresses of internal and external listeners. Instead of using the **type**, the addresses are now specified by **name**.

See ListenerStatus schema reference.

6.11. CRUISE CONTROL TLS SIDECAR PROPERTIES

The Cruise Control TLS sidecar has been removed. As a result, the **.spec.cruiseControl.tlsSidecar** and **.spec.cruiseControl.template.tlsSidecar** properties are now deprecated. The properties are ignored and will be removed in the future.

6.12. CRUISE CONTROL CAPACITY CONFIGURATION

The Cruise Control **disk** and **cpuUtilization** capacity configuration properties have been deprecated, are ignored, and will be removed in the future. The properties were used in setting capacity limits in optimization proposals to determine if resource-based optimization goals are being broken. Disk and CPU capacity limits are now automatically generated by AMQ Streams.

See Configuring and deploying Cruise Control with Kafka.

CHAPTER 7. FIXED ISSUES

The issues fixed in AMQ Streams 2.6 on OpenShift.

For details of the issues fixed in Kafka 3.6.0, refer to the Kafka 3.5.0 Release Notes.

Table 7.1. Fixed issues

Issue Number	Description
ENTMQST-4708	Recreate watcher when closed with an exception
ENTMQST-4932	Improve CPU estimation in the Cruise Control capacity configuration
ENTMQST-5132	Cluster Operator allows multiple JBOD disks for KRaft with node pools
ENTMQST-5133	Delete ClusterRoleBinding when MirrorMaker 2 with rack awareness is deleted
ENTMQST-5197	Allow use of default user quota while User Operator is enabled
ENTMQST-5333	Missing pause reconciliation for Unidirectional Topic Operator
ENTMQST-5341	Cluster Operator allows multiple JBOD disks for KRaft with node pools
ENTMQST-5372	Fix missing CO replicaset and pod YAMLs in report.sh
ENTMQST-5394	UTO NPE with missing partitions and/or replicas
ENTMQST-5575	BackPort fix of ZOOKEEPER-4708 to AMQ Streams

Table 7.2. Fixed common vulnerabilities and exposures (CVEs)

Issue Number	Description
ENTMQST-4990	jackson-databind: denial of service via cylic dependencies
ENTMQST-4999	CVE-2023-33201 bouncycastle: potential blind LDAP injection attack using a self-signed certificate
ENTMQST-5023	netty: io.netty:netty-handler: SniHandler 16MB allocation
ENTMQST-5047	CVE-2023-2976 guava: insecure temporary directory creation
ENTMQST-5385	CVE-2023-44981 [2.6] CVE-2023-44981 zookeeper: zookeeper: Authorization Bypass in Apache ZooKeeper
ENTMQST-5139	CVE-2023-20873 spring-boot: Security Bypass With Wildcard Pattern Matching on Cloud Foundry

Issue Number	Description
ENTMQST-5160	CVE-2022-46751 apache-ivy: XML External Entity vulnerability
ENTMQST-5173	CVE-2023-41080 tomcat: Open Redirect vulnerability in FORM authentication
ENTMQST-5293	CVE-2023-40167 jetty-http: jetty: Improper validation of HTTP/1 content-length
ENTMQST-5352	CVE-2023-42445 gradle: Possible local text file exfiltration by XML External entity injection
ENTMQST-5353	CVE-2023-44387 gradle: Incorrect permission assignment for symlinked files used in copy or archiving operations
ENTMQST-5398	CVE-2023-44981 zookeeper: Authorization Bypass in Apache ZooKeeper
ENTMQST-5427	CVE-2023-31582 jose4j: Insecure iteration count setting
ENTMQST-5437	CVE-2023-5072 in cruise-control

CHAPTER 8. KNOWN ISSUES

This section lists the known issues for AMQ Streams 2.6 on OpenShift.

8.1. STOPPING MIRRORMAKER 2 CONNECTORS

The new support for stopping connectors does not work in MirrorMaker 2 because the state is not passed to the connector operator. This will be fixed in the next release of AMQ Streams.

8.2. KAFKA BRIDGE SENDING MESSAGES WITH CORS ENABLED

If Cross-Origin Resource Sharing (CORS) is enabled for the Kafka Bridge, a *400 bad request error* is returned when sending a HTTP request to produce messages.

Workaround

To avoid this error, disable CORS in the Kafka Bridge configuration. HTTP requests to produce messages must have CORS disabled in the Kafka Bridge. This issue will be fixed in a future release of AMQ Streams.

To use CORS, you can deploy Red Hat 3scale for the Kafka Bridge.

- For information on deploying 3scale see, Using 3scale API Management with the AMQ Streams Kafka Bridge.
- For information on CORS request handling by 3scale, see Administering the API Gateway.

8.3. AMQ STREAMS CLUSTER OPERATOR ON IPV6 CLUSTERS

The AMQ Streams Cluster Operator does not start on Internet Protocol version 6 (IPv6) clusters.

Workaround

There are two workarounds for this issue.

Workaround one: Set the KUBERNETES_MASTER environment variable

1. Display the address of the Kubernetes master node of your OpenShift Container Platform cluster:

```
oc cluster-info
Kubernetes master is running at <master_address>
# ...
```

   Copy the address of the master node.

2. List all Operator subscriptions:

```
oc get subs -n <operator_namespace>
```

3. Edit the **Subscription** resource for AMQ Streams:

```
oc edit sub amq-streams -n <operator_namespace>
```

4. In **spec.config.env**, add the **KUBERNETES_MASTER** environment variable, set to the address of the Kubernetes master node. For example:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amq-streams
  namespace: <operator_namespace>
spec:
  channel: amq-streams-1.8.x
  installPlanApproval: Automatic
  name: amq-streams
  source: mirror-amq-streams
  sourceNamespace: openshift-marketplace
  config:
    env:
    - name: KUBERNETES_MASTER
      value: MASTER-ADDRESS
```

5. Save and exit the editor.

6. Check that the **Subscription** was updated:

```
oc get sub amq-streams -n <operator_namespace>
```

7. Check that the Cluster Operator **Deployment** was updated to use the new environment variable:

```
oc get deployment <cluster_operator_deployment_name>
```

Workaround two: Disable hostname verification

1. List all Operator subscriptions:

```
oc get subs -n <operator_namespace>
```

2. Edit the **Subscription** resource for AMQ Streams:

```
oc edit sub amq-streams -n <operator_namespace>
```

3. In **spec.config.env**, add the **KUBERNETES_DISABLE_HOSTNAME_VERIFICATION** environment variable, set to **true**. For example:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amq-streams
  namespace: <operator_namespace>
spec:
  channel: amq-streams-1.8.x
  installPlanApproval: Automatic
  name: amq-streams
  source: mirror-amq-streams
  sourceNamespace: openshift-marketplace
```

```
config:
  env:
  - name: KUBERNETES_DISABLE_HOSTNAME_VERIFICATION
    value: "true"
```

4. Save and exit the editor.
5. Check that the **Subscription** was updated:

```
oc get sub amq-streams -n <operator_namespace>
```

6. Check that the Cluster Operator **Deployment** was updated to use the new environment variable:

```
oc get deployment <cluster_operator_deployment_name>
```

8.4. CRUISE CONTROL CPU UTILIZATION ESTIMATION

Cruise Control for AMQ Streams has a known issue that relates to the calculation of CPU utilization estimation. CPU utilization is calculated as a percentage of the defined capacity of a broker pod. The issue occurs when running Kafka brokers across nodes with varying CPU cores. For example, node1 might have 2 CPU cores and node2 might have 4 CPU cores. In this situation, Cruise Control can underestimate and overestimate CPU load of brokers The issue can prevent cluster rebalances when the pod is under heavy load.

There are two workarounds for this issue.

Workaround one: Equal CPU requests and limits

You can set CPU requests equal to CPU limits in **Kafka.spec.kafka.resources**. That way, all CPU resources are reserved upfront and are always available. This configuration allows Cruise Control to properly evaluate the CPU utilization when preparing the rebalance proposals based on CPU goals.

Workaround two: Exclude CPU goals

You can exclude CPU goals from the hard and default goals specified in the Cruise Control configuration.

Example Cruise Control configuration without CPU goals

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    topicOperator: {}
    userOperator: {}
  cruiseControl:
    brokerCapacity:
      inboundNetwork: 10000KB/s
```

```
    outboundNetwork: 10000KB/s
  config:
    hard.goals: >
      com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.MinTopicLeadersPerBrokerGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundCapacityGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundCapacityGoal
    default.goals: >
      com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.MinTopicLeadersPerBrokerGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundCapacityGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundCapacityGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaDistributionGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.PotentialNwOutGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskUsageDistributionGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundUsageDistributionGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundUsageDistributionGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.TopicReplicaDistributionGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderReplicaDistributionGoal,
      com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderBytesInDistributionGoal
```

For more information, see Insufficient CPU capacity.

8.5. JMX AUTHENTICATION WHEN RUNNING IN FIPS MODE

When running AMQ Streams in FIPS mode with JMX authentication enabled, clients may fail authentication. To work around this issue, do not enable JMX authentication while running in FIPS mode. We are investigating the issue and working to resolve it in a future release.

CHAPTER 9. SUPPORTED CONFIGURATIONS

Supported configurations for the AMQ Streams 2.6 release.

9.1. SUPPORTED PLATFORMS

The following platforms are tested for AMQ Streams 2.6 running with Kafka on the version of OpenShift stated.

Platform	Version	Architecture
Red Hat OpenShift Container Platform (including disconnected environments)	4.11 to 4.14	x86_64, ppc64le (IBM Power), s390x (IBM Z and IBM® LinuxONE), aarch64 (64-bit ARM)
Red Hat OpenShift Dedicated	Latest	x86_64
Microsoft Azure Red Hat OpenShift	Latest	x86_64
Red Hat OpenShift Service on AWS	Latest	x86_64
Red Hat MicroShift	Latest	x86_64
Red Hat OpenShift Local	2.10-2.12 (OCP 4.11), 2.13-2.19 (OCP 4.12), 2.20-2.28 (OCP 4.13), 2.29 and newer (OCP 4.14)	x86_64

OpenShift Local is a limited version of Red Hat OpenShift Container Platform (OCP). Use only for development and evaluation on the understanding that some features may be unavailable.

Unsupported features

- Red Hat MicroShift does not support Kafka Connect's build configuration for building container images with connectors.
- IBM Z and IBM® LinuxONE s390x architecture does not support AMQ Streams OPA integration.

FIPS compliance

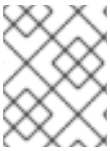
Red Hat OpenShift Container Platform is designed for FIPS. When running on RHEL or RHEL CoreOS booted in FIPS mode, OpenShift Container Platform core components use the RHEL cryptographic libraries submitted to NIST for FIPS validation on only the x86_64, ppc64le (IBM Power), s390x (IBM Z), and aarch64 (64-bit ARM) architectures. For more information about the NIST validation program, see Cryptographic Module Validation Program. For the latest NIST status for the individual versions of the RHEL cryptographic libraries submitted for validation, see Compliance Activities and Government Standards.

9.2. SUPPORTED CLIENTS

Only client libraries built by Red Hat are supported for AMQ Streams. Currently, AMQ Streams only provides a Java client library. Clients are supported for use with AMQ Streams 2.6 on the following operating systems and architectures:

Operating System	Architecture	JVM
RHEL and UBI 7	x86, amd64	Java 11
RHEL and UBI 8 and 9	x86, amd64, ppc64le (IBM Power), s390x (IBM Z and IBM® LinuxONE), aarch64 (64-bit ARM)	Java 11 and Java 17

Clients are tested with Open JDK 11 and 17. The IBM JDK is supported but not regularly tested against during each release. Open JDK 8, Oracle JDK 8 & 11, and IBM JDK 8 are not supported.



NOTE

Support for Red Hat Universal Base Image (UBI) versions correspond to the same RHEL version.

9.3. SUPPORTED APACHE KAFKA ECOSYSTEM

In AMQ Streams, only the following components released directly from the Apache Software Foundation are supported:

- Apache Kafka Broker
- Apache Kafka Connect
- Apache MirrorMaker
- Apache MirrorMaker 2
- Apache Kafka Java Producer, Consumer, Management clients, and Kafka Streams
- Apache ZooKeeper

NOTE

Apache ZooKeeper is supported solely as an implementation detail of Apache Kafka and should not be modified for other purposes. Additionally, the cores or vCPU allocated to ZooKeeper nodes are not included in subscription compliance calculations. In other words, ZooKeeper nodes do not count towards a customer's subscription.

9.4. ADDITIONAL SUPPORTED FEATURES

- Kafka Bridge
- Drain Cleaner
- Cruise Control
- Distributed Tracing

See also, Chapter 11, *Supported integration with Red Hat products* .

9.5. STORAGE REQUIREMENTS

Kafka requires block storage; file storage options like NFS are not compatible.

Additional resources

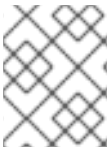
For information on the supported configurations for the AMQ Streams 2.5 LTS release, see the AMQ Streams 2.5 Release Notes.

CHAPTER 10. COMPONENT DETAILS

The following table shows the component versions for each AMQ Streams release.

AMQ Streams	Apache Kafka	Strimzi Operators	Kafka Bridge	Oauth	Cruise Control
2.6.0	3.6.0	0.38.0	0.27	0.14.0	2.5.128
2.5.1	3.5.0	0.36.0	0.26	0.13.0	2.5.123
2.5.0	3.5.0	0.36.0	0.26	0.13.0	2.5.123
2.4.0	3.4.0	0.34.0	0.25.0	0.12.0	2.5.112
2.3.0	3.3.1	0.32.0	0.22.3	0.11.0	2.5.103
2.2.2	3.2.3	0.29.0	0.21.5	0.10.0	2.5.103
2.2.1	3.2.3	0.29.0	0.21.5	0.10.0	2.5.103
2.2.0	3.2.3	0.29.0	0.21.5	0.10.0	2.5.89
2.1.0	3.1.0	0.28.0	0.21.4	0.10.0	2.5.82
2.0.1	3.0.0	0.26.0	0.20.3	0.9.0	2.5.73
2.0.0	3.0.0	0.26.0	0.20.3	0.9.0	2.5.73
1.8.4	2.8.0	0.24.0	0.20.1	0.8.1	2.5.59
1.8.0	2.8.0	0.24.0	0.20.1	0.8.1	2.5.59
1.7.0	2.7.0	0.22.1	0.19.0	0.7.1	2.5.37
1.6.7	2.6.3	0.20.1	0.19.0	0.6.1	2.5.11
1.6.6	2.6.3	0.20.1	0.19.0	0.6.1	2.5.11
1.6.5	2.6.2	0.20.1	0.19.0	0.6.1	2.5.11
1.6.4	2.6.2	0.20.1	0.19.0	0.6.1	2.5.11
1.6.0	2.6.0	0.20.0	0.19.0	0.6.1	2.5.11
1.5.0	2.5.0	0.18.0	0.16.0	0.5.0	-

AMQ Streams	Apache Kafka	Strimzi Operators	Kafka Bridge	Oauth	Cruise Control
1.4.1	2.4.0	0.17.0	0.15.2	0.3.0	-
1.4.0	2.4.0	0.17.0	0.15.2	0.3.0	-
1.3.0	2.3.0	0.14.0	0.14.0	0.1.0	-
1.2.0	2.2.1	0.12.1	0.12.2	-	-
1.1.1	2.1.1	0.11.4	-	-	-
1.1.0	2.1.1	0.11.1	-	-	-
1.0	2.0.0	0.8.1	-	-	-



NOTE

Strimzi 0.26.0 contains a Log4j vulnerability. The version included in the product has been updated to depend on versions that do not contain the vulnerability.

CHAPTER 11. SUPPORTED INTEGRATION WITH RED HAT PRODUCTS

AMQ Streams 2.6 supports integration with the following Red Hat products:

Red Hat Single Sign-On

Provides OAuth 2.0 authentication and OAuth 2.0 authorization.

Red Hat 3scale API Management

Secures the Kafka Bridge and provides additional API management features.

Red Hat build of Debezium

Monitors databases and creates event streams.

Red Hat build of Apicurio Registry

Provides a centralized store of service schemas for data streaming.

Red Hat build of Apache Camel K

Provides a lightweight integration framework.

For information on the functionality these products can introduce to your AMQ Streams deployment, refer to the product documentation.

11.1. RED HAT SINGLE SIGN-ON

AMQ Streams supports the use of OAuth 2.0 token-based authorization through Red Hat Single Sign-On Authorization Services, which allows you to manage security policies and permissions centrally.

11.2. RED HAT 3SCALE API MANAGEMENT

If you deployed the Kafka Bridge on OpenShift Container Platform, you can use it with 3scale. 3scale API Management can secure the Kafka Bridge with TLS, and provide authentication and authorization. Integration with 3scale also means that additional features like metrics, rate limiting and billing are available.

For information on deploying 3scale, see Using 3scale API Management with the AMQ Streams Kafka Bridge.

11.3. RED HAT BUILD OF DEBEZIUM FOR CHANGE DATA CAPTURE

The Red Hat build of Debezium is a distributed change data capture platform. It captures row-level changes in databases, creates change event records, and streams the records to Kafka topics. Debezium is built on Apache Kafka. You can deploy and integrate the Red Hat build of Debezium with AMQ Streams. Following a deployment of AMQ Streams, you deploy Debezium as a connector configuration through Kafka Connect. Debezium passes change event records to AMQ Streams on OpenShift. Applications can read these *change event streams* and access the change events in the order in which they occurred.

Debezium has multiple uses, including:

- Data replication
- Updating caches and search indexes
- Simplifying monolithic applications

- Data integration
- Enabling streaming queries

Debezium provides connectors (based on Kafka Connect) for the following common databases:

- Db2
- MongoDB
- MySQL
- PostgreSQL
- SQL Server

For more information on deploying Debezium with AMQ Streams, refer to the product documentation for the Red Hat build of Debezium .

11.4. RED HAT BUILD OF APICURIO REGISTRY FOR SCHEMA VALIDATION

You can use the Red Hat build of Apicurio Registry as a centralized store of service schemas for data streaming. For Kafka, you can use the Red Hat build of Apicurio Registry to store *Apache Avro* or JSON schema.

Apicurio Registry provides a REST API and a Java REST client to register and query the schemas from client applications through server-side endpoints.

Using Apicurio Registry decouples the process of managing schemas from the configuration of client applications. You enable an application to use a schema from the registry by specifying its URL in the client code.

For example, the schemas to serialize and deserialize messages can be stored in the registry, which are then referenced from the applications that use them to ensure that the messages that they send and receive are compatible with those schemas.

Kafka client applications can push or pull their schemas from Apicurio Registry at runtime.

For more information on using the Red Hat build of Apicurio Registry with AMQ Streams, refer to the product documentation for the Red Hat build of Apicurio Registry .

11.5. RED HAT BUILD OF APACHE CAMEL K

The Red Hat build of Apache Camel K is a lightweight integration framework built from Apache Camel K that runs natively in the cloud on OpenShift. Camel K supports serverless integration, which allows for development and deployment of integration tasks without the need to manage the underlying infrastructure. You can use Camel K to build and integrate event-driven applications with your AMQ Streams environment. For scenarios requiring real-time data synchronization between different systems or databases, Camel K can be used to capture and transform change in events and send them to AMQ Streams for distribution to other systems.

For more information on using the Camel K with AMQ Streams, refer to the product documentation for the Red Hat build of Apache Camel K .

Additional resources

- Red Hat Single Sign-On Supported Configurations
- Red Hat 3scale API Management Supported Configurations
- Red Hat build of Debezium Supported Configurations
- Red Hat build of Apicurio Registry Supported Configurations
- Red Hat build of Apache Camel K Supported Configurations

Revised on 2023-12-19 17:46:51 UTC