



Red Hat AMQ Streams 2.2

Release Notes for AMQ Streams 2.2 on OpenShift

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

Red Hat AMQ Streams 2.2 Release Notes for AMQ Streams 2.2 on OpenShift

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

Legal Notice

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

The release notes summarize the new features, enhancements, and fixes introduced in the AMQ Streams 2.2 release.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	4
CHAPTER 1. FEATURES	5
1.1. AMQ STREAMS 2.2.X (LONG TERM SUPPORT)	5
1.2. OPENSIFT CONTAINER PLATFORM SUPPORT	5
1.3. KAFKA 3.2.3 SUPPORT	5
1.4. SUPPORTING THE VIBETA2 API VERSION	5
1.4.1. Upgrading custom resources to v1beta2	6
1.5. SUPPORT FOR IBM Z AND LINUXONE ARCHITECTURE	6
1.5.1. Requirements for IBM Z and LinuxONE	6
1.5.2. Unsupported on IBM Z and LinuxONE	7
1.6. SUPPORT FOR IBM POWER ARCHITECTURE	7
1.6.1. Requirements for IBM Power	7
1.6.2. Unsupported on IBM Power	7
1.7. USESTRIMZIPODSETS FEATURE GATE (TECHNOLOGY PREVIEW)	7
1.8. USEKRAFT FEATURE GATE (DEVELOPMENT PREVIEW)	7
1.9. GENERAL AVAILABILITY FOR CRUISE CONTROL	8
1.10. CRUISE CONTROL SCALING AND REBALANCING MODES	9
1.11. DEBEZIUM FOR CHANGE DATA CAPTURE INTEGRATION	9
1.12. SERVICE REGISTRY	10
CHAPTER 2. ENHANCEMENTS	11
2.1. KAFKA 3.2.3 ENHANCEMENTS	11
2.2. TRACKING CRUISE CONTROL STATUS ON THE COMMAND LINE	11
2.3. RENEWING USER CERTIFICATES DURING THE MAINTENANCE TIME WINDOW	11
2.4. RENAMING CRUISE CONTROL TOPICS	11
2.5. USING CRUISE CONTROL WITHOUT ZOOKEEPER	12
2.6. CONFIGURING RACK AWARENESS FOR MIRRORMAKER 2.0	12
2.7. SETTING HEAP SIZES BASED ON THE PERCENTAGE OF AVAILABLE MEMORY	13
CHAPTER 3. TECHNOLOGY PREVIEWS	14
3.1. NEW FEATURE GATES	14
3.2. KAFKA STATIC QUOTA PLUGIN CONFIGURATION	14
CHAPTER 4. KAFKA BREAKING CHANGES	15
4.1. USING KAFKA'S EXAMPLE FILE CONNECTORS	15
CHAPTER 5. DEPRECATED FEATURES	16
5.1. OPENTRACING	16
5.2. JAVA 8	16
5.3. KAFKA MIRRORMAKER 1	16
5.4. CRUISE CONTROL TLS SIDECAR PROPERTIES	16
5.5. IDENTITY REPLICATION POLICY	16
5.6. LISTENERSTATUS TYPE PROPERTY	17
5.7. CRUISE CONTROL CAPACITY CONFIGURATION	17
CHAPTER 6. FIXED ISSUES	18
6.1. FIXED ISSUES FOR AMQ STREAMS 2.2.1	18
6.2. FIXED ISSUES FOR AMQ STREAMS 2.2.0	18
CHAPTER 7. KNOWN ISSUES	20
7.1. AMQ STREAMS CLUSTER OPERATOR ON IPV6 CLUSTERS	20
7.2. CRUISE CONTROL CPU UTILIZATION ESTIMATION	21

7.3. USER OPERATOR SCALABILITY	23
CHAPTER 8. SUPPORTED INTEGRATION WITH RED HAT PRODUCTS	24
CHAPTER 9. IMPORTANT LINKS	25

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 2.2 and subsequent patch releases introduce the features described in this section.

AMQ Streams 2.2 on OpenShift is based on Kafka 3.2.3 and Strimzi 0.29.x.



NOTE

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

1.1. AMQ STREAMS 2.2.X (LONG TERM SUPPORT)

AMQ Streams 2.2.x is the Long Term Support (LTS) offering for AMQ Streams.

The latest patch release is AMQ Streams 2.2.1. The AMQ Streams product images have changed to version 2.2.1. The supported Kafka version remains at 3.2.3.

For information on the LTS terms and dates, see the [AMQ Streams LTS Support Policy](#).

1.2. OPENSIFT CONTAINER PLATFORM SUPPORT

AMQ Streams 2.2 is supported on OpenShift Container Platform 4.8 to 4.11.

For more information about the supported platform versions, see the [AMQ Streams Supported Configurations](#).

1.3. KAFKA 3.2.3 SUPPORT

AMQ Streams now supports Apache Kafka version 3.2.3.

AMQ Streams uses Kafka 3.2.3. Only Kafka distributions built by Red Hat are supported.

You must upgrade the Cluster Operator to AMQ Streams version 2.2 before you can upgrade brokers and client applications to Kafka 3.2.3. For upgrade instructions, see [Upgrading AMQ Streams](#).

Refer to the [Kafka 3.1.0](#), [Kafka 3.2.0](#), [Kafka 3.2.1](#), and [Kafka 3.2.3](#) Release Notes for additional information.



NOTE

Kafka 3.1.x is supported only for the purpose of upgrading to AMQ Streams 2.2.

For more information on supported versions, see the [AMQ Streams Component Details](#).

Kafka 3.2.3 uses ZooKeeper version 3.6.3, which is the same version as Kafka 3.1.0.

1.4. 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 v1.22.

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 2.2.

See [Deploying and upgrading AMQ Streams](#).

1.4.1. 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 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 full instructions, see [Upgrading AMQ Streams](#).

1.5. SUPPORT FOR IBM Z AND LINUXONE ARCHITECTURE

AMQ Streams 2.2 is enabled to run on IBM Z and LinuxONE s390x architecture.

Support for IBM Z and LinuxONE applies to AMQ Streams running with Kafka on OpenShift Container Platform 4.10 and later.

1.5.1. Requirements for IBM Z and LinuxONE

- OpenShift Container Platform 4.10 and later

1.5.2. Unsupported on IBM Z and LinuxONE

- AMQ Streams on disconnected OpenShift Container Platform environments
- AMQ Streams OPA integration

1.6. SUPPORT FOR IBM POWER ARCHITECTURE

AMQ Streams 2.2 is enabled to run on IBM Power ppc64le architecture.

Support for IBM Power applies to AMQ Streams running with Kafka on OpenShift Container Platform 4.9 and later.

1.6.1. Requirements for IBM Power

- OpenShift Container Platform 4.9 and later

1.6.2. Unsupported on IBM Power

- AMQ Streams on disconnected OpenShift Container Platform environments

1.7. USESTRIMZIPODSETS FEATURE GATE (TECHNOLOGY PREVIEW)

The **UseStrimziPodSets** feature gate controls a resource for managing pods called **StrimziPodSet**. When the feature gate is enabled, this resource is used instead of the StatefulSets. AMQ Streams handles the creation and management of pods instead of OpenShift. Using StrimziPodSets instead of StatefulSets provides more control over the functionality.

The feature gate is at an alpha level of maturity, so should be treated as a technology preview.

The preview provides an opportunity to test the **StrimziPodSet** resource. The feature will be enabled by default in release 2.3.

To enable the feature gate, specify **+UseStrimziPodSets** as a value for the **STRIMZI_FEATURE_GATES** environment variable in the Cluster Operator configuration.

Enabling the UseStrimziPodSets feature gate

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

See [UseStrimziPodSets feature gate](#) and [Feature gate releases](#).

1.8. USEKRAFT FEATURE GATE (DEVELOPMENT PREVIEW)

As a Kafka cluster administrator, you can toggle a subset of features on and off using feature gates in the Cluster Operator deployment configuration.

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

This feature gate is at an alpha level of maturity, but it should be treated as a development preview.

CAUTION

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

To enable the **UseKRaft** feature gate, specify **+UseKRaft** and **+UseStrimziPodSets** as values for the **STRIMZI_FEATURE_GATES** environment variable in the Cluster Operator configuration. The **UseKRaft** feature gate depends on the **UseStrimziPodSets** feature gate.

Enabling the UseKRaft feature gate

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

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.
- Upgrades and downgrades of Apache Kafka versions or the AMQ Streams operator are not supported. Users might need to delete the cluster, upgrade the operator and deploy a new Kafka cluster.
- The Entity Operator (including the User Operator and Topic operator) is not supported. The **spec.entityOperator** property **must be removed** from the **Kafka** custom resource.
- **simple** authorization is not supported.
- SCRAM-SHA-512 authentication is not supported.
- JBOD storage is not supported. The **type: jbod** storage can be used, but the JBOD array can contain only one disk.
- Liveness and readiness probes are disabled.
- All Kafka nodes have both the **controller** and **broker** KRaft roles. Kafka clusters with separate **controller** and **broker** nodes are not supported.

See [UseKRaft feature gate](#) and [Feature gate releases](#).

1.9. GENERAL AVAILABILITY FOR CRUISE CONTROL

Cruise Control moves from Technology Preview to General Availability (GA). You can deploy [Cruise Control](#) and use it to rebalance your Kafka cluster using *optimization goals* – defined constraints on CPU, disk, network load, and more. In a balanced Kafka cluster, the workload is more evenly distributed across the broker pods.

Cruise Control is configured and deployed as part of a **Kafka** resource. You can use the default optimization goals or modify them to suit your requirements. Example YAML configuration files for Cruise Control are provided in **examples/cruise-control/**.

When Cruise Control is deployed, you can create **KafkaRebalance** custom resources to:

- Generate optimization proposals from multiple optimization goals

- Rebalance a Kafka cluster based on an optimization proposal

Other Cruise Control features are not currently supported, including anomaly detection, notifications, write-your-own goals, and changing the topic replication factor.

See [Cruise Control for cluster rebalancing](#).

1.10. CRUISE CONTROL SCALING AND REBALANCING MODES

You can now generate optimization proposals for rebalancing operations in one of the following modes:

- **full**
- **add-brokers**
- **remove-brokers**

Previously, the proposal was generated in **full** mode, where replicas might move across all brokers in a cluster. Now you use the **add-brokers** and **remove-brokers** modes to take into account scaling up and scaling down operations.

Use the **add-brokers** mode after scaling up. You specify new brokers and the rebalancing operation moves replicas from existing brokers to the newly added brokers. This is a faster option than rebalancing the whole cluster.

Use the **remove-brokers** mode before scaling down. You specify brokers you are going to remove, which means that any replicas on those brokers are moved off in the rebalance operation.

See [Rebalancing modes](#), [Generating optimization proposals](#), and [Approving optimization proposals](#).

1.11. DEBEZIUM FOR CHANGE DATA CAPTURE INTEGRATION

Red Hat 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 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](#).

1.12. SERVICE REGISTRY

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

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

Using Service 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 Service Registry at runtime.

For more information on using Service Registry with AMQ Streams, refer to the [Service Registry documentation](#).

CHAPTER 2. ENHANCEMENTS

AMQ Streams 2.2 adds a number of enhancements.

2.1. KAFKA 3.2.3 ENHANCEMENTS

For an overview of the enhancements introduced with Kafka 3.2.0, 3.2.1, and 3.2.3, refer to the [Kafka 3.2.0 Release Notes](#), [Kafka 3.2.1 Release Notes](#), and [Kafka 3.2.3 Release Notes](#).

2.2. TRACKING CRUISE CONTROL STATUS ON THE COMMAND LINE

It is now easier to check the status of optimization proposals. Instead of inspecting the resource configuration YAML, you can check the status on the command line.

When you run a proposal, run the following command and wait for the status of the optimization proposal to change to **ProposalReady**:

```
oc get kafkarebalance -o wide -w -n <namespace>
```

PendingProposal

A **PendingProposal** status means the rebalance operator is polling the Cruise Control API to check if the optimization proposal is ready.

ProposalReady

A **ProposalReady** status means the optimization proposal is ready for review and approval.

When the status changes to **ProposalReady**, the optimization proposal is ready to approve.

See [Generating optimization proposals](#) and [Approving optimization proposals](#).

2.3. RENEWING USER CERTIFICATES DURING THE MAINTENANCE TIME WINDOW

Maintenance time windows allow you to schedule certain rolling updates of your Kafka and ZooKeeper clusters to start at a convenient time. Maintenance windows are now supported in the User Operator. If you have deployed the User Operator using the Cluster Operator, and not as a standalone operator, automatic renewal of user certificates is included in the schedule.

If you are deploying the standalone User Operator, you can configure the maintenance time window during which expiring user certificates are renewed. You specify the time window as a Cron expression for the **STRIMZI_MAINTENANCE_TIME_WINDOWS** environment variable.

See [Deploying the standalone User Operator](#).

2.4. RENAMING CRUISE CONTROL TOPICS

It's now possible to rename the topics related to metrics that are created automatically by Cruise Control. You can use the following Cruise Control configuration properties to make the name change.

Example Cruise Control topic renaming

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
```

```

metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    # ...
    config: 1
    # ...
    metric.reporter.topic: cruise-control-metrics-reporter-topic-name
    partition.metric.sample.store.topic: cruise-control-partitions-metrics-name
    broker.metric.sample.store.topic: cruise-control-broker-metrics
    # ...
  # ...

```

If you change the names of the topics in an existing deployment, you need to remove the topics with the old names manually.

See [Configuring and deploying Cruise Control with Kafka](#).

2.5. USING CRUISE CONTROL WITHOUT ZOOKEEPER

Cruise Control now runs without ZooKeeper, using Kafka APIs instead. The TLS sidecar that was used for secure communication with ZooKeeper has been removed.

The TLS sidecar configuration for Cruise Control in the **Kafka** resource is no longer required. For this reason, the **.spec.cruiseControl.tlsSidecar** and **.spec.cruiseControl.template.tlsSidecar** properties are now deprecated.

See [Configuring and deploying Cruise Control with Kafka](#).

2.6. CONFIGURING RACK AWARENESS FOR MIRRORMAKER 2.0

You can now enable rack awareness in your MirrorMaker 2.0 resource configuration. This is a **specialized option** intended for a deployment within the same location, not across regions. You can use this option if you want connectors to consume from the closest replica rather than the leader replica.

A **topologyKey** in the **rack** configuration must match a node label containing the rack ID. In the following example, the standard **topology.kubernetes.io/zone** label is specified.

Rack configuration for MirrorMaker 2.0

```

apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  version: 3.2.3
  # ...
  rack:
    topologyKey: topology.kubernetes.io/zone

```

To consume from the closest replica, you must also enable the **RackAwareReplicaSelector** in the Kafka broker configuration.

Example rack configuration with enabled replica-aware selector

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    rack:
      topologyKey: topology.kubernetes.io/zone
    config:
      # ...
      replica.selector.class: org.apache.kafka.common.replica.RackAwareReplicaSelector
    # ...
```

See [Configuring Kafka MirrorMaker 2.0](#) and [Rack schema reference](#).

2.7. SETTING HEAP SIZES BASED ON THE PERCENTAGE OF AVAILABLE MEMORY

If you don't specify heap sizes in your configuration, the Cluster Operator imposes default heap sizes automatically. The Cluster Operator sets default maximum and minimum heap values based on a percentage of the memory resource configuration. The available memory allocated by default is now set at the levels shown in the following table.

Table 2.1. Default heap settings for components

Component	Percent of available memory allocated to the heap	Maximum limit
Kafka	50%	5 GB
ZooKeeper	75%	2 GB
Kafka Connect	75%	None
MirrorMaker 2.0	75%	None
MirrorMaker	75%	None
Cruise Control	75%	None
Kafka Bridge	50%	31 Gi

See [jvmOptions](#)

CHAPTER 3. TECHNOLOGY PREVIEWS



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 support scope, see [Technology Preview Features Support Scope](#).

3.1. NEW FEATURE GATES

Previews of the **UseKRaft** and **UseStrimziPodSets** feature gates are now available.

See [Chapter 1, Features](#).

3.2. KAFKA STATIC QUOTA PLUGIN CONFIGURATION

Use 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. KAFKA BREAKING CHANGES

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

4.1. 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.

Two example connector configuration files are provided:

- **examples/connect/kafka-connect-build.yaml** provides a Kafka Connect **build** configuration, which you can deploy to build a new Kafka Connect image with the file connectors.
- **examples/connect/source-connector.yaml** provides the configuration required to deploy the file connectors as **KafkaConnector** resources.

See [Deploying example KafkaConnector resources](#) and [Extending Kafka Connect with connector plugins](#).

CHAPTER 5. DEPRECATED FEATURES

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

5.1. OPENTRACING

Support for OpenTracing is deprecated.

The Jaeger clients are now retired and the OpenTracing project archived. As such, we cannot guarantee their support for future Kafka versions. We are introducing a new tracing implementation based on the OpenTelemetry project.

5.2. JAVA 8

Support for Java 8 was deprecated in Kafka 3.0.0 and AMQ Streams 2.0. Java 8 will be unsupported for all AMQ Streams components, including clients, in the future.

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

5.3. KAFKA MIRRORMAKER 1

Kafka MirrorMaker replicates data between two or more active Kafka clusters, within or across data centers. Kafka MirrorMaker 1 is deprecated for Kafka 3.0.0 and will be removed in Kafka 4.0.0. MirrorMaker 2.0 will be the only version available. MirrorMaker 2.0 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**. MirrorMaker 2.0 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 [Kafka MirrorMaker 2.0 cluster configuration](#).

5.4. CRUISE CONTROL TLS SIDECAR PROPERTIES

For this release, [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.

5.5. IDENTITY REPLICATION POLICY

Identity replication policy is used with MirrorMaker 2.0 to override the automatic renaming of remote topics. Instead of prepending the name with the name of the source cluster, the topic retains its original name. This optional setting is useful for active/passive backups and data migration.

The AMQ Streams Identity Replication Policy class (**io.strimzi.kafka.connect.mirror.IdentityReplicationPolicy**) is now deprecated and will be removed in

the future. You can update to use Kafka's own Identity Replication Policy (**class** `org.apache.kafka.connect.mirror.IdentityReplicationPolicy`).

See [Kafka MirrorMaker 2.0 cluster configuration](#).

5.6. 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](#).

5.7. CRUISE CONTROL CAPACITY CONFIGURATION

The **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 [Cruise Control configuration](#).

CHAPTER 6. FIXED ISSUES

The following sections list the issues fixed in AMQ Streams 2.2.x. Red Hat recommends that you upgrade to the latest patch release.

For details of the issues fixed in Kafka 3.2.0, 3.2.1, and 3.2.3, refer to the [Kafka 3.2.0 Release Notes](#) , [Kafka 3.2.1 Release Notes](#) , and [Kafka 3.2.3 Release Notes](#) .

6.1. FIXED ISSUES FOR AMQ STREAMS 2.2.1

The AMQ Streams 2.2.1 patch release (Long Term Support) is now available.

For additional details about the issues resolved in AMQ Streams 2.2.1, see [AMQ Streams 2.2.x Resolved Issues](#).

6.2. FIXED ISSUES FOR AMQ STREAMS 2.2.0

Table 6.1. Fixed issues

Issue Number	Description
ENTMQST-3757	[KAFKA] MirrorMaker 2.0 negative lag
ENTMQST-3762	"VertxException: Thread blocked" during Topic Operator startup
ENTMQST-3775	Bridge should not use slf4j-api and log4j-api at the same time
ENTMQST-3862	Improve logging in KafkaRoller
ENTMQST-3867	Fix non-cascading deletion of the StrimziPodSet resources
ENTMQST-3897	Reconciliation failures for KafkaConnector resources are not counted in operator metrics
ENTMQST-3918	Rolling update force-rolls pods during cluster startup
ENTMQST-3955	Add support for parsing storage in millibyte units
ENTMQST-3956	Fail reconciliation when invalid storage unit is used
ENTMQST-3958	Avoid unnecessary rolling updates of the Cruise Control deployment
ENTMQST-3972	Missing annotation ANNO_STRIMZI_IO_CLUSTER_CA_CERT_GENERATION on pods cause errors in CO log during Kafka reconciliations
ENTMQST-3997	Kafka Connect Build should fail when curl download fails
ENTMQST-4017	Errors on KafkaRebalance custom resource not logged properly

Issue Number	Description
ENTMQST-4071	Handle FIPS mode in the AMQ Streams Drain cleaner
ENTMQST-4264	[KAFKA] Unauthenticated clients may cause OutOfMemoryError on brokers

Table 6.2. Fixed common vulnerabilities and exposures (CVEs)

Issue Number	Description
ENTMQST-3917	CVE-2020-36518 jackson-databind: denial of service via a large depth of nested objects
ENTMQST-4049	CVE-2022-24823 netty: world readable temporary file containing sensitive data
ENTMQST-4050	CVE-2022-25647 com.google.code.gson-gson: Deserialization of Untrusted Data in com.google.code.gson-gson

CHAPTER 7. KNOWN ISSUES

This section lists the known issues for AMQ Streams 2.2 on OpenShift.

7.1. 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>
```

7.2. 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.

Workaround

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](#).

7.3. USER OPERATOR SCALABILITY

The User Operator can timeout when creating multiple users at the same time. Reconciliation can take too long.

Workaround

If you encounter this issue, reduce the number of users you are creating at the same time. And wait until they are ready before creating more users.

CHAPTER 8. SUPPORTED INTEGRATION WITH RED HAT PRODUCTS

AMQ Streams 2.2 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 Debezium

Monitors databases and creates event streams.

Red Hat Service Registry

Provides a centralized store of service schemas for data streaming.

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

Additional resources

- [Red Hat Single Sign-On Supported Configurations](#)
- [Red Hat 3scale API Management Supported Configurations](#)
- [Red Hat Debezium Supported Configurations](#)
- [Red Hat Service Registry Supported Configurations](#)

CHAPTER 9. IMPORTANT LINKS

- [AMQ Streams Supported Configurations](#)
- [AMQ Streams Component Details](#)

Revised on 2023-03-08 16:15:28 UTC