Red Hat AMQ Streams 2.5

Release Notes for AMQ Streams 2.5 on OpenShift

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

Last Updated: 2024-02-22

Red Hat AMQ Streams 2.5 Release Notes for AMQ Streams 2.5 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 2.5 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. AMQ STREAMS 2.5 LONG TERM SUPPORT

AMQ Streams 2.5 is a Long Term Support (LTS) offering for AMQ Streams.

For information on the LTS terms and dates, see the AMQ Streams LTS Support Policy .

CHAPTER 2. FEATURES

AMQ Streams 2.5 introduces the features described in this section.

AMQ Streams 2.5 on OpenShift is based on Apache Kafka 3.5.0 and Strimzi 0.36.x.

NOTE

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

2.1. AMQ STREAMS 2.5.X (LONG TERM SUPPORT)

AMQ Streams 2.5.x is the Long Term Support (LTS) offering for AMQ Streams.

The latest patch release is AMQ Streams 2.5.1. The AMQ Streams product images have changed to version 2.5.1. The supported Kafka version remains at 3.5.0.

For information on the LTS terms and dates, see the AMQ Streams LTS Support Policy .

2.2. OPENSHIFT CONTAINER PLATFORM SUPPORT

AMQ Streams 2.5 is supported on OpenShift Container Platform 4.10 to 4.14.

For more information, see Chapter 10, *Supported Configurations*.

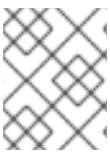
2.3. KAFKA 3.5.0 SUPPORT

AMQ Streams now supports and uses Apache Kafka version 3.5.0. Only Kafka distributions built by Red Hat are supported.

You must upgrade the Cluster Operator to AMQ Streams version 2.5 before you can upgrade brokers and client applications to Kafka 3.5.0. For upgrade instructions, see Upgrading AMQ Streams .

Refer to the Kafka 3.5.0 Release Notes for additional information.

Kafka 3.4.x is supported only for the purpose of upgrading to AMQ Streams 2.5.



NOTE

Kafka 3.5.0 provides access to KRaft mode, where Kafka runs without ZooKeeper by utilizing the Raft protocol. KRaft mode is available as a Developer Preview.

2.4. 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 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 2.5.

2.4.1. 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 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 .

2.5. (PREVIEW) NODE POOLS FOR MANAGING NODES IN A KAFKA CLUSTER

This release introduces the **KafkaNodePools** feature gate and a new **KafkaNodePool** custom resource that 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 developer 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 preview.

See Configuring node pools.

2.6. (PREVIEW) UNIDIRECTIONAL TOPIC MANAGEMENT USING THE TOPIC OPERATOR

This release also incorporates the **UnidirectionalTopicOperator** feature gate, introducing 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 developer 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.

2.7. REPORTING TOOL FOR RETRIEVING DIAGNOSTIC AND TROUBLESHOOTING DATA

The **report.sh** diagnostics tool is a script provided by Red Hat to gather essential data for troubleshooting AMQ Streams deployments on OpenShift. It collects relevant logs, configuration files, and other diagnostic data to assist in identifying and resolving issues. When you run the script, you can use additional parameters to retrieve specific data.

The tool requires the OpenShift **oc** command-line tool to establish a connection to the running cluster. After which you can open a terminal and run the tool to retrieve data on components.

From the following request, data is collected on a Kafka cluster, a Kafka Bridge cluster, and on secret keys and data values:

Example request with data collection options

```
./report.sh --namespace=my-amq-streams-namespace --cluster=my-kafka-cluster --bridge=my-bridge-component --secrets=all --out-dir=~/reports
```

The data is output to a specified directory.

See Retrieving diagnostic and troubleshooting data.

2.8. OPENTELEMETRY FOR DISTRIBUTED TRACING

OpenTelemetry for distributed tracing has moved to GA. You can use OpenTelemetry with a specified tracing system. OpenTelemetry has replaced OpenTracing for distributed tracing. Support for OpenTracing is deprecated.

By Default, OpenTelemetry uses the OTLP (OpenTelemetry Protocol) exporter for tracing. AMQ Streams with OpenTelemetry is distributed for use with the Jaeger exporter, but you can specify other tracing systems supported by OpenTelemetry. AMQ Streams plans to migrate to using OpenTelemetry with the OTLP exporter by default, and is phasing out support for the Jaeger exporter.

See Introducing distributed tracing.

CHAPTER 3. ENHANCEMENTS

AMQ Streams 2.5 adds a number of enhancements.

3.1. KAFKA 3.5.0 ENHANCEMENTS

For an overview of the enhancements introduced with Kafka 3.5.0, refer to the Kafka 3.5.0 Release Notes.

3.2. USESTRIMZIPODSETS FEATURE GATE MOVES TO GA

The **UseStrimziPodSets** feature gate has moved to GA, which means it is now permanently enabled and cannot be disabled.

StrimziPodSet resources are now used to manage pods instead of **StatefulSet** resources. This means that AMQ Streams handles the creation and management of pods instead of OpenShift, providing more control over the functionality.

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

3.3. KRAFT REQUIRES NODE POOL CONFIGURATION

To deploy a Kafka cluster in KRaft mode, you must now enable the **UseStrimziPodSets** and **KafkaNodePools** feature gates. KRaft mode is supported only by using **KafkaNodePool** resources to manage the configuration of Kafka nodes.

For more information, see the following:

- Section 2.5, "(Preview) Node pools for managing nodes in a Kafka cluster"
- Section 5.4, "**UseKRaft** feature gate"

3.4. OAUTH 2.0 SUPPORT FOR KRAFT MODE

KeycloakRBACAuthorizer, the Red Hat Single Sign-On authorizer provided with AMQ Streams, has been replaced with the **KeycloakAuthorizer**. The new authorizer is compatible with using AMQ Streams with ZooKeeper cluster management or in KRaft mode. As with the previous authorizer, to be able to use the Red Hat Single Sign-On REST endpoints for Authorization Services provided by Red Hat Single Sign-On, you configure **KeycloakAuthorizer** on the Kafka broker. **KeycloakRBACAuthorizer** can still be used when using AMQ Streams with ZooKeeper cluster management, but you should migrate to the new authorizer.

3.5. OAUTH 2.0 CONFIGURATION PROPERTIES FOR GRANT MANAGEMENT

You can now use additional configuration to manage OAuth 2.0 grants from the authorization server.

If you are using Red Hat Single Sign-On for OAuth 2.0 authorization, you can add the following properties to the authorization configuration of your Kafka brokers:

- **grantsMaxIdleTimeSeconds** specifies the time in seconds after which an idle grant in the cache can be evicted. The default value is 300.

- **grantsGcPeriodSeconds** specifies the time, in seconds, between consecutive runs of a job that cleans stale grants from the cache. The default value is 300.
- **grantsAlwaysLatest** controls whether the latest grants are fetched for a new session. When enabled, grants are retrieved from Red Hat Single Sign-On and cached for the user. The default value is **false**.

Kafka configuration to use OAuth 2.0 authorization

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    authorization:
      type: keycloak
      tokenEndpointUri: <https://<auth_server_-_address>/auth/realms/external/protocol/openid-
connect/token>
      clientId: kafka
      # ...
      grantsMaxIdleSeconds: 300
      grantsGcPeriodSeconds: 300
      grantsAlwaysLatest: false
    #...
```

See Configuring OAuth 2.0 authorization support.

3.6. OAUTH 2.0 SUPPORT FOR JSONPATH QUERIES WHEN EXTRACTING USERNAMES

To use OAuth 2.0 authentication in a Kafka cluster, you specify listener configuration in the **Kafka** custom resource with the authentication method **oauth**. When configuring the listener properties, it is now possible to use a JsonPath query to extract a username from the authorization server being used. You can use a JsonPath query to specify username extraction options in your listener for the **userNameClaim** and **fallbackUserNameClaim** properties. This allows you to extract a username from a token by accessing a specific value within a nested data structure. For example, you might have a username that is contained within a *user info* data structure within a JSON token data structure.

The following example shows how JsonPath queries are used with the properties when configuring token validation using an introspection endpoint.

Configuring token validation using an introspection endpoint

```
- name: external
  port: 9094
  type: loadbalancer
  tls: true
  authentication:
    type: oauth
    validIssuerUri: <https://<auth-server-address>/auth/realms/external>
    introspectionEndpointUri: <https://<auth-server-address>/auth/realms/external/protocol/openid-
connect/token/introspect>
    clientId: kafka-broker
```

```
    clientSecret:
      secretName: my-cluster-oauth
      key: clientSecret
    userNameClaim: "['user.info'].['user.id']" 1
    maxSecondsWithoutReauthentication: 3600
    fallbackUserNameClaim: "['client.info'].['client.id']" 2
    fallbackUserNamePrefix: client-account-
    # ...
```

1 The token claim (or key) that contains the actual user name in the token. The user name is the *principal* used to identify the user. The **userNameClaim** value depends on the authorization server used.

2 An authorization server may not provide a single attribute to identify both regular users and clients. When a client authenticates in its own name, the server might provide a *client ID*. When a user authenticates using a username and password, to obtain a refresh token or an access token, the server might provide a *username* attribute in addition to a client ID. Use this fallback option to specify the username claim (attribute) to use if a primary user ID attribute is not available. If required, you can use JsonPath query to target nested attributes.

See Configuring OAuth 2.0 support for Kafka brokers .

3.7. ADDED KAFKA EXPORTER SUPPORT TO EXCLUDE TOPICS AND CONSUMER GROUPS

Support for Kafka Exporter deployment configuration introduces new properties to exclude specified topics and consumer groups from the metrics extracted from Kafka brokers.

You can use the following properties in the Kafka Exporter specification:

- **groupExcludeRegex** to exclude specific consumer groups
- **topicExcludeRegex** to exclude specific topics

In the following example configuration, the two properties exclude topics and consumer groups that start with the prefix **excluded-**.

Example configuration for deploying Kafka Exporter

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  kafkaExporter:
    image: my-registry.io/my-org/my-exporter-cluster:latest
    groupRegex: ".*"
    topicRegex: ".*"
    groupExcludeRegex: "^excluded-.*"
    topicExcludeRegex: "^excluded-.*"
# ...
```

See **KafkaExporterSpec** schema reference.

3.8. KAFKA BRIDGE ENHANCEMENTS FOR METRICS AND OPENAPI

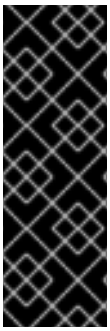
The latest release of the Kafka Bridge introduces the following changes:

- Removes the **remote** and **local** labels from HTTP server-related metrics to prevent time series sample growth.
- Eliminates accounting HTTP server metrics for requests on the /**metrics** endpoint.
- Exposes the /**metrics** endpoint through the OpenAPI specification, providing a standardized interface for metrics access and management.
- Fixes the **OffsetRecordSentList** component schema to return record offsets or errors.
- Fixes the **ConsumerRecord** component schema to return key and value as objects, not just (JSON) strings.
- Corrects the HTTP status codes returned by the /**ready** and /**healthy** endpoints:
  - Changes the successful response code from **200** to **204**, indicating no content in the response for success.
  - Adds the **500** status code to the specification for the failure case, indicating no content in the response for errors.

See Using the AMQ Streams Kafka Bridge .

CHAPTER 4. TECHNOLOGY PREVIEWS

Technology Preview features included with AMQ Streams 2.5.



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.

4.1. 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 5. DEVELOPER PREVIEWS

Developer preview features included with AMQ Streams 2.5.

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.

5.1. KAFKANODEPOOLS FEATURE GATE

To use **KafkaNodePool** resources to manage the configuration of pools of Kafka nodes, try the **KafkaNodePools** feature gate.

For more information, see Section 2.5, "(Preview) Node pools for managing nodes in a Kafka cluster" .

5.2. UNIDIRECTIONALTOPICOPERATOR FEATURE GATE

To set up the Topic Operator so that it only manages Kafka topics associated with **KafkaTopic** resources, try the **UnidirectionalTopicOperator** feature gate.

For more information, see Section 2.6, "(Preview) Unidirectional topic management using the Topic Operator".

5.3. STABLECONNECTIDENTITIES FEATURE GATE

To use **StrimziPodSet** resources to manage Kafka Connect and Kafka MirrorMaker 2 pods, try the **StableConnectIdentities** feature gate.

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

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

Enabling the StableConnectIdentities feature gate

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

See **StableConnectIdentities** feature gate.

5.4. USEKRAFT FEATURE GATE

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.

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 the following:

- **UseKRaft** feature gate
- Feature gate releases

CHAPTER 6. KAFKA BREAKING CHANGES

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

6.1. 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 the following:

- Deploying example KafkaConnector resources
- Extending Kafka Connect with connector plugins

CHAPTER 7. DEPRECATED FEATURES

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

7.1. STATEFULSET SUPPORT REMOVED

In this release, the **UseStrimziPodSets** feature gate moved to GA, which means it 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.

7.2. 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 .

7.3. OPENTRACING

Support for **type: jaeger** tracing 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.

7.4. 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.

7.5. 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.

7.6. 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.

7.7. 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.

7.8. 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.

7.9. 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 Configuring and deploying Cruise Control with Kafka.

CHAPTER 8. FIXED ISSUES

The following sections list the issues fixed in AMQ Streams 2.5.x. Red Hat recommends that you upgrade to the latest patch release.

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

8.1. FIXED ISSUES FOR AMQ STREAMS 2.5.1

The AMQ Streams 2.5.1 patch release (Long Term Support) is now available.

KAFKA-15353

The 2.5.1 patch release includes a fix for KAFKA-15353, an issue that was included in the Kafka 3.5.2 release. Note that the patch release introduced a fix for this specific issue, not all issues fixed for Kafka 3.5.2.

For more information on the issue, see the Kafka 3.5.2 Release Notes.

HTTP/2 DoS vulnerability (CVE-2023-44487)

The release addresses CVE-2023-44487, a critical Denial of Service (DoS) vulnerability in the HTTP/2 protocol. The vulnerability stems from mishandling multiplexed streams, allowing a malicious client to repeatedly request new streams and promptly cancel them using an **RST_STREAM** frame. By doing so, the attacker forces the server to expend resources setting up and tearing down streams without reaching the server-side limit for active streams per connection. For more information on this vulnerability, see the CVE-2023-44487 page for a description.

For additional details about the issues resolved in AMQ Streams 2.5.1, see AMQ Streams 2.5.x Resolved Issues.

8.2. FIXED ISSUES FOR AMQ STREAMS 2.5.0

Table 8.1. Fixed issues

Issue Number	Description
ENTMQST-3757	[KAFKA] Mirror Maker 2 negative lag
ENTMQST-3954	Topic is not successfully created without "spec:" in KafkaTopic
ENTMQST-4430	All Zookeeper pods are deleted when are rolled with invalid configuration
ENTMQST-4496	[BRIDGE] Logged HTTP response status code could be different from the actual one returned to the client
ENTMQST-4555	When KafkaRebalance resource is Ready, it should not transition due to Kafka Cluster failure
ENTMQST-4707	Make connector task backoff configurable in Kafka Connect

Issue Number	Description
ENTMQST-4723	The AMQ Streams Operator doesn't create the require Network Policy once Kafka Exporter is enabled
ENTMQST-4735	Startup failure for Cruise Control when OAuth 2.0 metrics are enabled
ENTMQST-4772	Connect/Coonector operator stuck when REST API query fails
ENTMQST-4774	Add **insecure=true** parameter to be applicable to maven type in the build of **KafkaConnect**
ENTMQST-4822	Certificate key replacement fails when Cluster Operator crashes before the trust is established
ENTMQST-4850	Provide proper error message when Cruise Control fails to generate **KafkaRebalance** proposal
ENTMQST-4909	Improve usability of resizing persistent volumes
ENTMQST-5050	Cruise Control and **KafkaNodePool** resources - operator doesn't reflect number of replicas inside **KafkaNodePool**
ENTMQST-5051	Fix various validations based on number of replicas to work with node pools

Table 8.2. Fixed common vulnerabilities and exposures (CVEs)

Issue Number	Description
ENTMQST-4484	snakeyaml: Constructor Deserialization Remote Code Execution
ENTMQST-4995	TRIAGE-CVE-2023-34454 snappy-java-repolib: snappy-java: Integer overflow in compress leads to DoS
ENTMQST-4996	TRIAGE-CVE-2023-34454 snappy-java-debuginfo: snappy-java: Integer overflow in compress leads to DoS
ENTMQST-4997	TRIAGE-CVE-2023-34454 snappy-java: Integer overflow in compress leads to DoS
ENTMQST-4998	TRIAGE-CVE-2023-34455 snappy-java: Unchecked chunk length leads to DoS
ENTMQST-5120	CVE-2023-34462 Flaw in Netty's SniHandler while navigating TLS handshake; DoS
ENTMQST-5121	CVE-2023-0482 RESTEasy: creation of insecure temp files
ENTMQST-5122	CVE-2022-24823 netty: world readable temporary file containing sensitive data

Issue Number	Description
ENTMQST-5123	CVE-2021-37137 netty-codec: SnappyFrameDecoder doesn't restrict chunk length and may buffer skippable chunks in an unnecessary way
ENTMQST-5124	CVE-2021-37136 netty-codec: Bzip2Decoder doesn't allow setting size restrictions for decompressed data
ENTMQST-5125	CVE-2023-3635 DoS of the Okio client when handling a crafted GZIP archive
ENTMQST-5126	CVE-2023-26048 Jetty servlets with multipart support may cause OOM error with client requests
ENTMQST-5127	CVE-2023-26049 Non-standard cookie parsing in Jetty may allow an attacker to smuggle cookies within other cookies
ENTMQST-5128	CVE-2022-36944 scala: deserialization gadget chain
ENTMQST-5134	TRIAGE-CVE-2023-3635 okio: GzipSource class improper exception handling
ENTMQST-5178	CVE-2023-26048 jetty-server: OutOfMemoryError for large multipart without filename read via request.getParameter()
ENTMQST-5179	CVE-2023-26049 jetty-server: Cookie parsing of quoted values can exfiltrate values from other cookies

CHAPTER 9. KNOWN ISSUES

This section lists the known issues for AMQ Streams 2.5 on OpenShift.

9.1. 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.

9.2. 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>
```

9.3. 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.

9.4. 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 10. SUPPORTED CONFIGURATIONS

Supported configurations for the AMQ Streams 2.5 release.

10.1. SUPPORTED PLATFORMS

The following platforms are tested for AMQ Streams 2.5 running with Kafka on the version of OpenShift stated.

Platform	Version	Architecture
OpenShift Container Platform	4.10 to 4.14	x86_64, ppc64le (IBM Power), s390x (IBM Z and IBM® LinuxONE), aarch64 (64-bit ARM)
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

NOTE

Support for aarch64 (64-bit ARM) applies to AMQ Streams 2.5 when running Kafka 3.5.0 only.

Unsupported features

- Red Hat MicroShift does not support Kafka Connect's build configuration for building container images with connectors.
- AMQ Streams running on IBM Power ppc64le, IBM Z s390x, or IBM® LinuxONE s390x architecture is unsupported on *disconnected* OpenShift Container Platform environments. Additionally, the IBM Z and IBM® LinuxONE s390x architecture does not support AMQ Streams OPA integration.

10.2. 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.

10.3. ADDITIONAL SUPPORTED FEATURES

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

See also, Chapter 12, *Supported integration with Red Hat products* .

10.4. 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 2.2 LTS release, see the AMQ Streams Supported Configurations article on the customer portal.

CHAPTER 11. 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
2.5.1	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	-
1.4.1	2.4.0	0.17.0	0.15.2	0.3.0	-

AMQ Streams	Apache Kafka	Strimzi Operators	Kafka Bridge	Oauth	Cruise Control
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 12. SUPPORTED INTEGRATION WITH RED HAT PRODUCTS

AMQ Streams 2.5 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 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.

12.1. 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.

12.2. 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.

12.3. 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 .

12.4. 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 .

12.5. 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 2024-02-22 14:12:19 UTC