Red Hat AMQ Streams 2.4

Release Notes for AMQ Streams 2.4 on OpenShift

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

Last Updated: 2023-08-22

Red Hat AMQ Streams 2.4 Release Notes for AMQ Streams 2.4 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 2.4 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 2.4 introduces the features described in this section.

AMQ Streams 2.4 on OpenShift is based on Apache Kafka 3.4.0 and Strimzi 0.34.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 2.4 is supported on OpenShift Container Platform 4.10 to 4.13.

For more information about the supported platform versions, see the AMQ Streams Supported Configurations.

1.2. KAFKA 3.4.0 SUPPORT

AMQ Streams now supports Apache Kafka version 3.4.0.

AMQ Streams uses Kafka 3.4.0. Only Kafka distributions built by Red Hat are supported.

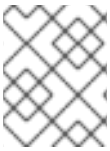
You must upgrade the Cluster Operator to AMQ Streams version 2.4 before you can upgrade brokers and client applications to Kafka 3.4.0. For upgrade instructions, see Upgrading AMQ Streams.

Refer to the Kafka 3.4.0 Release Notes for additional information.

NOTE

Kafka 3.3.x is supported only for the purpose of upgrading to AMQ Streams 2.4.

For more information on supported versions, see the AMQ Streams Component Details.



NOTE

Kafka 3.4.0 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

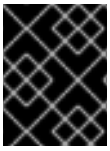
The **v1beta2** API version for all custom resources was introduced with AMQ Streams 1.7. For AMQ Streams 1.8, **v1alpha1** and **v1beta1** API versions were removed from all AMQ Streams custom resources apart from **KafkaTopic** and **KafkaUser**.

Upgrade of the custom resources to **v1beta2** prepares AMQ Streams for a move to Kubernetes CRD **v1**, which is required for Kubernetes 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 2.4.

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 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. NEW STABLECONNECTIDENTITIES FEATURE GATE TO MANAGE PODS

This release introduces the **StableConnectIdentities** feature gate, which is disabled by default. This feature gate is at an alpha level of maturity, and should be treated as a developer preview.

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 UseKRaft feature gate

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

See **StableConnectIdentities** feature gate.

1.5. IMPROVED FIPS SUPPORT

Federal Information Processing Standards (FIPS) are standards for computer security and interoperability. From this release, AMQ Streams can run on FIPS-enabled OpenShift clusters without special configuration and automatically switches to FIPS mode.

Prior to this release, running AMQ Streams on FIPS-enabled OpenShift clusters was possible only by disabling FIPS mode using the **FIPS_MODE** environment variable in the deployment configuration for the Cluster Operator. If you are currently running AMQ Streams on a FIPS-enabled OpenShift cluster with the **FIPS_MODE** set to disabled, to be FIPs-compliant you can enable it when upgrading to AMQ Streams 2.4.

A couple of things to note when running AMQ Streams in FIPS mode:

- SCRAM-SHA-512 passwords need to be at least 32 characters long. If you have a Kafka cluster with custom configuration that uses a password length that is less than 32 characters, you need to update your configuration.
- If you are using FIPS-enabled OpenShift clusters, you may experience higher memory consumption compared to regular OpenShift clusters. To avoid any issues, we suggest increasing the memory request to at least 512Mi.

See the following:

- FIPS support
- Switching to FIPS mode when upgrading AMQ Streams

1.6. AUTOMATIC RESTARTS FOR CONNECTORS

A new configuration property enables automatic restarts of failed connectors and tasks for Kafka Connect and Kafka MirrorMaker 2. If the **autoRestart** property is set to **true**, up to seven restart attempts are made, after which restarts must be made manually.

For Kafka Connect connectors, you configure the **autoRestart** property in the **KafkaConnector** custom resource.

Example Kafka Connect configuration with automatic restarts enabled

```
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
  config:
    file: "/opt/kafka/LICENSE"
    topic: my-topic
    # ...
```

For MirrorMaker 2 connectors, you configure the **autoRestart** property in the **KafkaMirrorMaker2** custom resource. You can enable automatic restarts for each of the internal connectors used by MirrorMaker 2: **sourceConnector**, **heartbeatConnector**, and **checkpointConnector**.

Example MirrorMaker 2 configuration with automatic restarts enabled

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mm2-cluster
spec:
  mirrors:
  - sourceConnector:
      autoRestart:
        enabled: true
      # ...
    heartbeatConnector:
      autoRestart:
        enabled: true
      # ...
    checkpointConnector:
      autoRestart:
        enabled: true
      # ...
```

See **AutoRestart** schema properties.

1.7. SUPPORT FOR IBM Z AND LINUXONE ARCHITECTURE

AMQ Streams 2.4 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.

1.7.1. Requirements for IBM Z and LinuxONE

- OpenShift Container Platform 4.10 and later

1.7.2. Unsupported on IBM Z and LinuxONE

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

1.8. SUPPORT FOR IBM POWER ARCHITECTURE

AMQ Streams 2.4 is enabled to run on IBM Power ppc64le architecture.

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

1.8.1. Requirements for IBM Power

- OpenShift Container Platform 4.10 and later

1.8.2. Unsupported on IBM Power

- AMQ Streams on disconnected OpenShift Container Platform environments

1.9. 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 Red Hat build of Debezium documentation.

1.10. 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 Red Hat build of Apicurio Registry documentation .

CHAPTER 2. ENHANCEMENTS

AMQ Streams 2.4 adds a number of enhancements.

2.1. KAFKA 3.4.0 ENHANCEMENTS

For an overview of the enhancements introduced with Kafka 3.4.0, refer to the Kafka 3.4.0 Release Notes.

2.2. OAUTH 2.0 CONFIGURATION FOR HTTP REQUESTS

You can now use configuration to control HTTP requests to an OAuth 2.0 authorization server.

If you are creating a listener for your Kafka brokers that uses OAuth 2.0 authentication or authorization, you can add the following properties to the listener configuration:

- **httpRetries** to control the maximum number of times to retry a failed HTTP request to the authorization server.
- **httpRetryPauseMs** to control the time to wait in milliseconds before attempting another retry of a failed HTTP request to the authorization server.

You can also use the properties when configuring OAuth 2.0 authentication for Kafka components.

Example listener configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3
    version: 3.4.0
    # ...
    listeners:
      - name: external
        port: 9094
        type: loadbalancer
        tls: true
        authentication:
          type: oauth
          # ...
          httpRetries: 2
          httpRetryPauseMs: 300
    # ...
```

See the following:

- Configuring OAuth 2.0 support for Kafka brokers
- Configuring OAuth 2.0 for Kafka components
- Configuring OAuth 2.0 authorization support

2.3. SUPPORT FOR ENCRYPTED CONNECTION TO OPEN POLICY AGENT (OPA) SERVER

If you are using OPA for authorized access to your Kafka brokers, you can now configure an encrypted HTTPS connection to access the OPA server. Add trusted certificates to your OPA configuration using the **tlsTrustedCertificates** property.

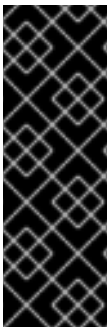
Example Open Policy Agent configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    authorization:
      type: opa
      # ...
      tlsTrustedCertificates:
        - secretName: opa-server-ca
          certificate: tls.crt
    # ...
```

See **KafkaAuthorizationOpa** schema reference.

CHAPTER 3. TECHNOLOGY PREVIEWS

Technology Preview features included with AMQ Streams 2.4.



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. OPENTELEMETRY FOR DISTRIBUTED TRACING

OpenTelemetry for distributed tracing is available as a technology preview. You can use OpenTelemetry with a specified tracing system. OpenTelemetry is replacing 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.

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. DEVELOPER PREVIEWS

Developer preview features included with AMQ Streams 2.4.

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. STABLECONNECTIDENTITIES FEATURE GATE

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

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

For more information, see Section 1.4, "New **StableConnectIdentities** feature gate to manage pods".

4.2. 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.

This feature gate is at an alpha level of maturity, and should be treated as a developer preview.

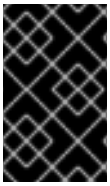
CAUTION

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** as a value for the **STRIMZI_FEATURE_GATES** environment variable in the Cluster Operator configuration.

Enabling the UseKRaft feature gate

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



IMPORTANT

The **UseKRaft** feature gate depends on the **UseStrimziPodSets** feature gate. When enabling the **UseKRaft** feature gate, make sure that the **UseStrimziPodSets** feature gate is enabled as well.

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 Topic Operator is not supported. The **spec.entityOperator.topicOperator** property **must be removed** from the **Kafka** custom resource.
- 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.
- All Kafka nodes have both the **controller** and **broker** KRaft roles. Kafka clusters with separate **controller** and **broker** nodes are not supported.

See the following:

- **UseKRaft** feature gate
- 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.

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 6. DEPRECATED FEATURES

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

6.1. STATEFULSET SUPPORT REMOVED IN AMQ STREAMS 2.5.0

In the AMQ Streams 2.3 release, the **UseStrimziPodSets** feature gate moved to a beta level of maturity and is enabled by default. For AMQ Streams 2.5, the **UseStrimziPodSets** feature gate moves to GA, which means it will be permanently enabled and cannot be disabled. For this reason, support for **StatefulSet** resources to manage pods is deprecated and will be removed in AMQ Streams 2.5.

6.2. JAVA 8 SUPPORT REMOVED IN AMQ STREAMS 2.4.0

Support for Java 8 was deprecated in AMQ Streams 2.0.0. Support for Java 8 is removed in AMQ Streams 2.4.0. This applies to all AMQ Streams components, including clients.

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.

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 .

6.3. 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.

6.4. 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.

6.5. 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 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**. 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 Kafka MirrorMaker 2 cluster configuration.

6.6. KAFKA MIRRORMAKER 2 IDENTITY REPLICATION POLICY

Identity replication policy is used with MirrorMaker 2 to override the automatic renaming of remote topics. You 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.

To implement an identity replication policy, you specify a replication policy class (**replication.policy.class**) in the MirrorMaker 2 configuration. The AMQ Streams **mirror-maker-2-extensions** component, which includes the **io.strimzi.kafka.connect.mirror.IdentityReplicationPolicy** class, 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**).

6.7. 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.

6.8. 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.

6.9. 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 7. FIXED ISSUES

The issues fixed in AMQ Streams 2.4 on OpenShift.

For details of the issues fixed in Kafka 3.4.0, refer to the Kafka 3.4.0 Release Notes.

Table 7.1. Fixed issues

Issue Number	Description
ENTMQST-2033	Rolling update after cluster cert deletion is stuck when operationTimeout is on 30s
ENTMQST-3756	User Operator does not scale
ENTMQST-3799	Bridge raising access denied exception when missing content-type in the request
ENTMQST-4107	[KAFKA] MM2 connector task stopped and didn't result in failed state
ENTMQST-4109	[KAFKA] Confusing error in MM2 when offsets for a group cannot be synced
ENTMQST-4115	KafkaRoller: NPEs when the Pod does not exist
ENTMQST-4346	Delete the StrimziPodSet or StatefulSet first when migrating between them
ENTMQST-4349	KafkaConnect build does not use custom repository for parent maven dependency resolution
ENTMQST-4405	Enabling metrics fails bridge startup with **NoClassDefFoundError** exception
ENTMQST-4427	Fix validation of the **useServiceDnsDomain** for cluster-ip type listeners
ENTMQST-4428	HTTP client not get assigned partitions via /assignments endpoint
ENTMQST-4429	Resources should validate correctness of new configuration
ENTMQST-4479	Newly added OAuth Password Grant feature not working in Kafka Bridge
ENTMQST-4489	Sending messages with CORS enabled raises a 400 Bad request with Null body error
ENTMQST-4493	[Kafka Bridge] Producing async=true drives to OpenTelemetry spans not linked together
ENTMQST-4494	Add support to cgroups v2 in Kafka Bridge
ENTMQST-4513	Confusing Cruise Control logs when finished **KafkaRebalance** resource is not deleted
ENTMQST-4521	Connector auto-restart counter does not reset back to 0

Issue Number	Description
ENTMQST-4721	Allow Kafka exporter to change the timezone
ENTMQST-4779	Kafka Exporter dashboard does not work in newer Grafana versions
ENTMQST-4821	Certificate key replacement fails when Cluster Operator crashes after the trust is established

Table 7.2. Fixed common vulnerabilities and exposures (CVEs)

Issue Number	Description
CVE-2022-1471	Drain Cleaner dependency: Red Hat build of Quarkus 2.13.7
CVE-2022-4147	Drain Cleaner dependency: Red Hat build of Quarkus 2.13.5
ENTMQST-4786	CVE-2022-42003 CVE-2022-42003 jackson-databind: deep wrapper array nesting when **UNWRAP_SINGLE_VALUE_ARRAYS** enabled
ENTMQST-4788	CVE-2022-42004 jackson-databind: use of deeply nested arrays
ENTMQST-4795	CVE-2023-25194: Apache Kafka: Possible RCE/Denial of service attack via SASL JAAS **JndiLoginModule** configuration using Kafka Connect
ENTMQST-4796	CVE-2020-36518 jackson-databind before 2.13.0 allows a Java StackOverflow exception and denial of service via a large depth of nested objects
ENTMQST-4797	CVE-2021-37137 Snappy frame decoder function doesn't restrict the chunk length which may lead to excessive memory usage
ENTMQST-4798	CVE-2021-37136 Bzip2 decompression decoder function doesn't allow setting size restrictions on the decompressed output data
ENTMQST-4799	CVE-2022-24823 Local information disclosure vulnerability in Netty
ENTMQST-4802	CVE-2022-36944 Scala 2.13.x before 2.13.9 has a Java deserialization risk via a gadget chain
ENTMQST-4803	CVE-2023-1370 JSON processor lib may cause stack exhaustion (stack overflow) due to recursive nesting of arrays/objects
ENTMQST-4804	CVE-2023-24815 Vert.x-Web apps serving files using **StaticHandler** on Windows issue

CHAPTER 8. KNOWN ISSUES

This section lists the known issues for AMQ Streams 2.4 on OpenShift.

8.1. 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.

8.2. 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>
```

8.3. 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.

8.4. 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.

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.

8.6. STARTUP FAILURE FOR CRUISE CONTROL WHEN OAUTH 2.0 METRICS ARE ENABLED

If you enable OAuth 2.0 metrics using the **enableMetrics** property, Cruise Control fails with an exception. To be able to use Cruise Control, you must not configure the **enableMetrics** property or set it to **false**. We are investigating the issue and working to resolve it in a future release.

Track this issue: ENTMQST-4735.

CHAPTER 9. SUPPORTED INTEGRATION WITH RED HAT PRODUCTS

AMQ Streams 2.4 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 Red Hat build of Apicurio 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 10. IMPORTANT LINKS

- AMQ Streams Supported Configurations
- AMQ Streams Component Details

Revised on 2023-08-22 16:11:31 UTC