Red Hat AMQ Streams 2.3

Release Notes for AMQ Streams 2.3 on OpenShift

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

Last Updated: 2023-03-08

Red Hat AMQ Streams 2.3 Release Notes for AMQ Streams 2.3 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 2.3 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 2.3 introduces the features described in this section.

AMQ Streams 2.3 on OpenShift is based on Kafka 3.3.1 and Strimzi 0.32.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 2.3 is supported on OpenShift Container Platform 4.8 to 4.12.

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

1.2. KAFKA 3.3.1 SUPPORT

AMQ Streams now supports Apache Kafka version 3.3.1.

AMQ Streams uses Kafka 3.3.1. Only Kafka distributions built by Red Hat are supported.

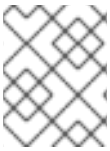
You must upgrade the Cluster Operator to AMQ Streams version 2.3 before you can upgrade brokers and client applications to Kafka 3.3.1. For upgrade instructions, see Upgrading AMQ Streams.

Refer to the Kafka 3.3.0 and Kafka 3.3.1 Release Notes for additional information.

NOTE

Kafka 3.2.x is supported only for the purpose of upgrading to AMQ Streams 2.3.

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



NOTE

Kafka 3.3.1 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

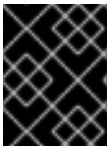
The **v1beta2** API version for all custom resources was introduced with AMQ Streams 1.7. For AMQ Streams 1.8, **v1alpha1** and **v1beta1** API versions were removed from all AMQ Streams custom resources apart from **KafkaTopic** and **KafkaUser**.

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 2.3.

1.3.1. Upgrading custom resources to v1beta2

To support the upgrade of custom resources to **v1beta2**, AMQ Streams provided the **Red Hat AMQ Streams API Conversion Tool** with AMQ Streams 1.8. Download the tool from the AMQ Streams 1.8 software downloads page.

You perform the custom resources upgrades in two steps.

Step one: Convert the format of custom resources

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

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

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

Step two: Upgrade CRDs to v1beta2

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

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

1.4. AUTOMATIC APPROVAL OF CRUISE CONTROL OPTIMIZATION PROPOSALS

When using AMQ Streams with Cruise Control, you can now automate the process of approving the optimization proposals generated. You generate an optimization proposal using the **KafkaRebalance** custom resource. To enable auto-approval, you add **strimzi.io/rebalance-auto-approval: "true"** as an annotation to the **KafkaRebalance** custom resource before you generate the proposal.

With manual approval, you make another request when a generated proposal has a **ProposalReady** status. You approve the proposal by adding the **strimzi.io/rebalance: approve** annotation to the **KafkaRebalance** resource in the new request.

With automatic approval, the proposal is generated and approved to complete the rebalance in a single request.

See Generating optimization proposals.

1.5. SUPPORT FOR MULTIPLE OPERATIONS IN ACL RULE CONFIGURATION

The **KafkaUser** custom resource has been updated to make configuration of ACL lists easier to manage.

Previously, you configured operations for ACL rules separately for each resource using the **operation** property.

Old format for configuring ACL rules

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

A new **operations** property allows you to list multiple ACL operations as a single rule for the same resource.

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.

See **ACLRule** schema reference.

1.6. NEW CLUSTER-IP INTERNAL LISTENER TYPE

Listeners are used for client connection to Kafka brokers. They are configured in the **Kafka** resource using **.spec.kafka.listeners** properties.

A new **cluster-ip** type of internal listener exposes a Kafka cluster based on per-broker **ClusterIP** services.

Example cluster-ip listener configuration

```
#...
spec:
  kafka:
    #...
    listeners:
      - name: external-cluster-ip
        type: cluster-ip
        tls: false
        port: 9096
    #...
```

This is a useful option when you can't route through the headless service or you wish to incorporate a custom access mechanism. For example, you might use this listener when building your own type of external listener for a specific Ingress controller or the Kubernetes Gateway API.

See **GenericKafkaListener** schema reference.

1.7. CLUSTER OPERATOR LEADER ELECTION TO RUN MULTIPLE REPLICAS

Use leader election to run multiple parallel replicas of the Cluster Operator. One replica is elected as the active leader and operates the deployed resources. The other replicas run in standby mode. Replicas are useful for high availability. Additional replicas safeguard against disruption caused by major failure. This is especially important since the introduction of StrimziPodSets, whereby AMQ Streams handles the creation and management of pods for Kafka clusters. The Cluster Operator is responsible for restarting the pods.

To enable leader election, the **STRIMZI_LEADER_ELECTION_ENABLED** environment variable for the Cluster Operator must be set to true (default). The environment variable is set, along with related environment variables, in the **Deployment** custom resource that is used to deploy the Cluster Operator. By default, AMQ Streams runs with a single Cluster Operator replica that is always the leader replica. To add more replicas, you update the **spec.replicas** value in the **Deployment** custom resource.

Deployment configuration for Cluster Operator replicas and leader election

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-cluster-operator
  labels:
    app: strimzi
spec:
  replicas: 1
  # ...
    spec:
      # ...
      containers:
        - name: strimzi-cluster-operator
          image: registry.redhat.io/amq7/amq-streams-rhel8-operator:2.3.0
```

```
        # ...
        env:
          # ...
          - name: STRIMZI_LEADER_ELECTION_ENABLED
            value: "true"
          - name: STRIMZI_LEADER_ELECTION_LEASE_NAME
            value: "strimzi-cluster-operator"
          - name: STRIMZI_LEADER_ELECTION_LEASE_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          - name: STRIMZI_LEADER_ELECTION_IDENTITY
            valueFrom:
              fieldRef:
# ...
```

See Running multiple Cluster Operator replicas with leader election and Leader election environment variables.

1.8. SUPPORT FOR IBM Z AND LINUXONE ARCHITECTURE

AMQ Streams 2.3 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.

1.8.1. Requirements for IBM Z and LinuxONE

- OpenShift Container Platform 4.10 and later

1.8.2. Unsupported on IBM Z and LinuxONE

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

1.9. SUPPORT FOR IBM POWER ARCHITECTURE

AMQ Streams 2.3 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.

1.9.1. Requirements for IBM Power

- OpenShift Container Platform 4.9 and later

1.9.2. Unsupported on IBM Power

- AMQ Streams on disconnected OpenShift Container Platform environments

1.10. 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.

1.11. 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 2.3 adds a number of enhancements.

2.1. KAFKA 3.3.1 ENHANCEMENTS

For an overview of the enhancements introduced with Kafka 3.3.0 and 3.3.1, refer to the Kafka 3.3.0 and Kafka 3.3.1 Release Notes.

2.2. KAFKA CONNECTOR STATUS

The status of the **KafkaConnector** custom resource now shows **NotReady** if the connector or any associated tasks are reporting as **FAILED**. Previously, the custom resource would show **READY** even when the connector or a task had failed.

2.3. CONTROLPLANELISTENER FEATURE GATE MOVES TO GA

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

With **ControlPlaneListener** enabled, the connections between the Kafka controller and brokers use an internal control plane listener on port 9090.

See **ControlPlaneListener** feature gate.

IMPORTANT

With the **ControlPlaneListener** feature gate permanently enabled, it is no longer possible to upgrade or downgrade directly between AMQ Streams 1.7 and earlier and AMQ Streams 2.3 and newer. You have to upgrade or downgrade through one of the AMQ Streams versions in between. For more information, see Upgrading from an AMQ Streams version earlier than 1.7.

2.4. SERVICEACCOUNTPATCHING FEATURE GATE MOVES TO GA

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

With **ServiceAccountPatching** enabled, the Cluster Operator always reconciles service accounts and updates them when needed. For example, when you change service account labels or annotations using the template property of a custom resource, the operator automatically updates them on the existing service account resources.

See **ServiceAccountPatching** feature gate.

2.5. USESTRIMZIPODSETS FEATURE GATE MOVES TO BETA

The **UseStrimziPodSets** feature gate moves to a beta level of maturity, and is now enabled by default. This means StrimziPodSets are used by default instead of StatefulSets.

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

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

2.6. RACK AWARENESS CONFIGURATION FOR THE KAFKA BRIDGE

Rack awareness for Kafka Bridge pods is now supported. Use the **KafkaBridge** custom resource to configure rack awareness. You can configure a Kafka Bridge pod to be aware of the rack in which it runs. A rack can represent an availability zone, data center, or an actual rack in your data center.

Example rack configuration for Kafka Bridge

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
# ...
spec:
  # ...
  rack:
    topologyKey: topology.kubernetes.io/zone
  # ...
```

See **Rack** schema reference.

2.7. PLUGGABLE POD SECURITY PROFILES

Security context defines constraints on pods and containers. OpenShift uses built-in security context constraints (SCCs) to control permissions. SCCs are the settings and strategies that control the security features a pod has access to. You can also create and manage your own SCCs.

The optional **STRIMZI_POD_SECURITY_PROVIDER_CLASS** environment variable for the Cluster Operator provides security context configuration for pods and containers.

See Applying security context to AMQ Streams pods and containers .

2.8. KAFKA BROKER RESTART EVENTS

After the Cluster Operator restarts a Kafka pod in an OpenShift cluster, it emits an OpenShift event into the pod's namespace explaining why the pod restarted. For help in understanding cluster behavior, you can check the reason for a restart event from the command line. When checking restart **events** from the command line, you can also specify a **reason** or other **field-selector** options to filter the events returned.

The following example returns restart events that were triggered due to an error.

Returning restart events that were triggered for a specified reason

```
oc -n kafka get events --field-selector reportingController=strimzi.io/cluster-
operator,reason=PodForceRestartOnError
```

See Finding information on Kafka restarts.

2.9. CONFIGURABLE KAFKA ADMIN CLIENT

A new User Operator environment variable called **STRIMZI_KAFKA_ADMIN_CLIENT_CONFIGURATION** can pass additional configuration to the Kafka Admin client. The Kafka Admin client helps manage brokers and topics. You can now tune the Kafka Admin client without rebuilding the User Operator. For example, you can use it to pass SASL configuration or tune timeouts.

Kafka Admin client timeout configuration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-user-operator
  labels:
    app: strimzi
spec:
  # ...
  template:
    # ...
    spec:
      # ...
      containers:
        - name: strimzi-user-operator
          # ...
          env:
            - name: STRIMZI_KAFKA_ADMIN_CLIENT_CONFIGURATION
              value: |
                default.api.timeout.ms=120000
                request.timeout.ms=60000
```

NOTE

This is an advanced configuration option, provided without validation.

See Deploying the standalone User Operator.

2.10. CRUISE CONTROL CAPACITY OVERRIDES

New Cruise Control configuration options allow you to specify overrides that set the network capacity and CPU limits for each Kafka broker. You can use these options when brokers are running on nodes with heterogeneous network or CPU resources.

Override capacity limits can be set for the following broker resources:

- **cpu** - CPU resource in millicores or CPU cores (Default: 1)
- **inboundNetwork** - Inbound network throughput in byte units per second (Default: 10000KiB/s)
- **outboundNetwork** - Outbound network throughput in byte units per second (Default: 10000KiB/s)

An example of Cruise Control capacity overrides configuration using bibyte units

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    # ...
    brokerCapacity:
      cpu: "1"
      inboundNetwork: 10000KiB/s
      outboundNetwork: 10000KiB/s
      overrides:
      - brokers: [0]
        cpu: "2.755"
        inboundNetwork: 20000KiB/s
        outboundNetwork: 20000KiB/s
      - brokers: [1, 2]
        cpu: 3000m
        inboundNetwork: 30000KiB/s
        outboundNetwork: 30000KiB/s
```

See Capacity overrides.

2.11. OAUTH 2.0 PASSWORD GRANTS FOR KAFKA CLIENTS

You can now configure Kafka clients to use the OAuth password grants mechanism for interaction with Kafka brokers.

Password grants mechanism properties

```
security.protocol=SASL_SSL
sasl.mechanism=OAUTHBEARER
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required \
  oauth.token.endpoint.uri="<token_endpoint_url>" \
  oauth.client.id="<client_id>" \ 1
  oauth.client.secret="<client_secret>" \ 2
  oauth.password.grant.username="<username>" \ 3
  oauth.password.grant.password="<password>" \ 4
  oauth.scope="<scope>" \
  oauth.audience="<audience>" ;
  # ...
```

1. Client ID, which is the name used when creating the *client* in the authorization server.
2. (Optional) Client secret created when creating the *client* in the authorization server.
3. Username for password grant authentication. OAuth password grant configuration (username and password) uses the OAuth 2.0 password grant method. To use password grants, create a user account for a client on your authorization server with limited permissions. The account should act like a service account. Use in environments where user accounts are required for authentication, but consider using a refresh token first.
4. Password for password grant authentication.

See Configuring Kafka Java clients to use OAuth 2.0 .

NOTE

At the time of release, a minor issue was discovered that currently prevents password grants from working with the Kafka Bridge for AMQ Streams on OpenShift. For more information, see the known issue for OAuth password grants configuration .

2.12. AUTHENTICATION AND AUTHORIZATION METRICS

You can now collect metrics specific to **oauth** authentication and **opa** or **keycloak** authorization. You do this by setting the **enableMetrics** property to true in the listener configuration of the Kafka resource. For example, set **enableMetrics** to **true** in **spec.kafka.listeners.authentication** and **spec.kafka.authorization**. Similarly, you can enable metrics for oauth authentication in the **KafkaBridge**, **KafkaConnect**, **KafkaMirrorMaker**, and **KafkaMirrorMaker2** custom resources.

See Introducing metrics.

CHAPTER 3. TECHNOLOGY PREVIEWS

Technology Preview features included with AMQ Streams 2.3.

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

This release introduces OpenTelemetry for distributed tracing 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.

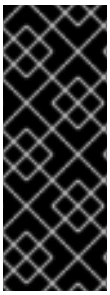
Example Kafka Static Quota plugin configuration

```
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 2.3.



IMPORTANT

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

4.1. USEKRAFT FEATURE GATE

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, 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** 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 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.
- 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.

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

CHAPTER 6. DEPRECATED FEATURES

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

6.1. 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 will be 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 .

6.2. 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.

6.3. 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. For more information, see Section 1.5, “Support for multiple operations in ACL rule configuration” .

6.4. 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 .

6.5. 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.

6.6. 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 .

6.7. 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.

6.8. 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 2.3 on OpenShift.

For details of the issues fixed in Kafka 3.3.0 and 3.3.1, refer to the Kafka 3.3.0 and Kafka 3.3.1 Release Notes.

Table 7.1. Fixed issues

Issue Number	Description
ENTMQST-4273	Avoid unnecessary rolling updates during upgrades when only inter.broker.protocol.version is set
ENTMQST-4095	Badly formatted response from Connect can cause the operator to get stack on a single reconciliation
ENTMQST-1366	Remove unused **@DefaultValue** annotations
ENTMQST-4065	Fix NPE when user specifies incorrect TLS secret or key inside that secret
ENTMQST-4178	Fix various issues in the standalone UO and TO installation files
ENTMQST-4177	Ignore errors when trying to clean the /tmp directory
ENTMQST-4483	KafkaTopic **config: {}** is automatically added or removed
ENTMQST-4016	Incorrect **KafkaRebalance** related annotation value logged when it's not valid/unknown
ENTMQST-3840	ZooKeeper rolling update handling of unready pods
ENTMQST-4093	[KAFKA] **log.cleaner.io.max.bytes.per.second** cannot be changed

Table 7.2. Fixed common vulnerabilities and exposures (CVEs)

Issue Number	Description
ENTMQST-4312	CVE-2022-42004 jackson-databind: use of deeply nested arrays
ENTMQST-4311	CVE-2022-42003 jackson-databind: deep wrapper array nesting wrt UNWRAP_SINGLE_VALUE_ARRAYS
ENTMQST-4302	CVE-2022-38752 snakeyaml: Uncaught exception in java.base/java.util.ArrayList.hashCode
ENTMQST-4188	CVE-2022-2047 jetty-http: improver hostname input handling

CHAPTER 8. KNOWN ISSUES

This section lists the known issues for AMQ Streams 2.3 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.

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.

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. OAUTH PASSWORD GRANTS CONFIGURATION

OAuth password grants are currently not being handled correctly by the Kafka Bridge. The OAuth authentication is not being configured properly.

This will be fixed for the next release.

Issue Number	Description
ENTMQST-4479	Newly added OAuth Password Grant feature not working in Kafka Bridge

8.6. OPENTELEMETRY: RUNNING JAEGER WITH TLS ENABLED

Support for tracing using OpenTelemetry is built in to the following Kafka components:

- Kafka Connect
- MirrorMaker

- MirrorMaker 2
- AMQ Streams Kafka Bridge

When using the Jaeger exporter, trace data is retrieved through the Jaeger gPRC endpoint. By default, this endpoint does not have TLS enabled. However, it can still be configured to use TLS when deploying the Jaeger instance using the Jaeger operator. For example, when running the Red Hat OpenShift distributed tracing operator on OpenShift, which is a Jaeger operator, the operator automatically enables TLS. Jaeger instances with TLS on the gRPC endpoint are not supported on AMQ Streams.

There are two workarounds for this issue.

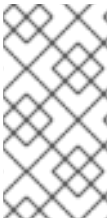
Workaround one: Disable TLS on the gRPC endpoint

Create a Jaeger custom resource and disable TLS on the gRPC port by specifying the following properties.

- **collector.grpc.tls.enabled: false**
- **reporter.grpc.tls.enabled: false**

Example Jaeger custom resource to disable TLS

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: my-jaeger
spec:
  allInOne:
    options:
      agent.grpc.tls.enabled: false
      collector.grpc.tls.enabled: false
```



NOTE

This configuration uses the **allInOne** strategy, which deploys all Jaeger components in a single pod. Other deployment strategies, such as the **production** strategy for production environments, separate the Jaeger components into separate pods for increased scalability and reliability.

Workaround two: Export traces through an OpenTelemetry collector

Use a collector to receive, process, and export the OpenTelemetry trace data. To resolve the issue by exporting trace data through an OpenTelemetry collector, you can follow these steps:

1. Deploy the Red Hat OpenShift distributed tracing collection operator.
2. Configure an **OpenTelemetryCollector** custom resource to deploy the collector to receive trace data through a non-TLS-enabled endpoint and pass it to a TLS-enabled endpoint.
3. In the custom resource, specify the **receivers** properties to create a non-TLS-enabled Jaeger gRPC endpoint on port 14250. You can also create other endpoints, such as an OTLP endpoint, if you are using other tracing systems.
4. Specify the **exporters** properties to point to the TLS-enabled Jaeger gRPC endpoint.

5. Declare the pipeline configuration in the **pipelines** properties of the custom resource.

In this example, the pipeline is from Jaeger and OTLP receivers to a Jaeger gRPC endpoint.

Example OpenTelemetry collector configuration

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: cluster-collector
  namespace: <namespace>
spec:
  mode: deployment
  config: |
   receivers:
     otlp:
       protocols:
         grpc:
         http:
     jaeger:
       protocols:
         grpc:
   exporters:
     jaeger:
     endpoint: jaeger-all-in-one-inmemory-collector-headless.openshift-distributed-
tracing.svc.cluster.local:14250
       tls:
         ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
   service:
     pipelines:
       traces:
         receivers: [otlp,jaeger]
         exporters: [jaeger]
```

To use the collector, you then need to specify the collector endpoint as the exporter endpoint in the tracing configuration.

Example tracing configuration for Kafka Connect using OpenTelemetry

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  template:
    connectContainer:
      env:
        - name: OTEL_SERVICE_NAME
          value: my-otel-service
        - name: OTEL_EXPORTER_JAEGER_ENDPOINT
          value: "http:// jaeger-all-in-one-inmemory-collector-headless.openshift-distributed-
tracing.svc.cluster.local:14250"
```

```
tracing:
  type: opentelemetry
#...
```

CHAPTER 9. SUPPORTED INTEGRATION WITH RED HAT PRODUCTS

AMQ Streams 2.3 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 2023-03-08 11:58:48 UTC