Red Hat AMQ 2020.Q4

Release Notes for AMQ Streams 1.6 on OpenShift

For use with AMQ Streams on OpenShift Container Platform

Last Updated: 2022-02-02

Red Hat AMQ 2020.Q4 Release Notes for AMQ Streams 1.6 on OpenShift

For use with AMQ Streams on OpenShift Container Platform

Legal Notice

Abstract

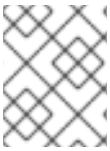
These release notes contain the latest information about new features, enhancements, fixes, and issues contained in the AMQ Streams 1.6 release.

Table of Contents

CHAPTER 1. FEATURES

AMQ Streams version 1.6 is based on Strimzi 0.20.x.

The features added in this release, and that were not in previous releases of AMQ Streams, are outlined below.



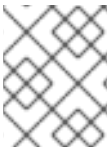
NOTE

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

1.1. KAFKA SUPPORT IN AMQ STREAMS 1.6.X (LONG TERM SUPPORT ON OCP 3.11)

This section describes the versions of Kafka and ZooKeeper that are supported in AMQ Streams 1.6 and the subsequent patch releases.

AMQ Streams 1.6.x is the Long Term Support release for use with OCP 3.11, and is supported only for as long as OpenShift Container Platform 3.11 is supported.



NOTE

AMQ Streams 1.6.4 and later patch releases are supported on OCP 3.11 only. If you are using OCP 4.x you are required to upgrade to AMQ Streams 1.7.x or later.

For information on support dates for AMQ LTS versions, see the Red Hat Knowledgebase solution How long are AMQ LTS releases supported?.

Only Kafka distributions built by Red Hat are supported. Previous versions of Kafka are supported in AMQ Streams 1.6.x only for upgrade purposes.

For more information on supported Kafka versions, see the Red Hat AMQ 7 Component Details Page on the Customer Portal.

1.1.1. Kafka support in AMQ Streams 1.6.6 and 1.6.7

The AMQ Streams 1.6.6 and 1.6.7 releases support Apache Kafka version 2.6.3.

You must upgrade the Cluster Operator before you can upgrade brokers and client applications to Kafka 2.6.3. For upgrade instructions, see AMQ Streams and Kafka upgrades .

Kafka 2.6.3 requires ZooKeeper version 3.5.9. Therefore, the Cluster Operator does *not* perform a ZooKeeper upgrade when upgrading from AMQ Streams 1.6.4 / 1.6.5.

Refer to the Kafka 2.6.3 Release Notes for additional information.

1.1.2. Kafka support in AMQ Streams 1.6.4 and 1.6.5

The AMQ Streams 1.6.4 and 1.6.5 releases support Apache Kafka version 2.6.2 and ZooKeeper version 3.5.9.

You must upgrade the Cluster Operator before you can upgrade brokers and client applications to Kafka 2.6.2. For upgrade instructions, see AMQ Streams and Kafka upgrades .

Kafka 2.6.2 requires ZooKeeper version 3.5.9. Therefore, the Cluster Operator will perform a ZooKeeper upgrade when upgrading from AMQ Streams 1.6.2.

Refer to the Kafka 2.6.2 Release Notes for additional information.

1.1.3. Kafka support in AMQ Streams 1.6.0 and 1.6.2

AMQ Streams 1.6.0 and 1.6.2 support Apache Kafka version 2.6.0.

You must upgrade the Cluster Operator before you can upgrade brokers and client applications to Kafka 2.6.0. For upgrade instructions, see AMQ Streams and Kafka upgrades .

Refer to the Kafka 2.5.0 and Kafka 2.6.0 Release Notes for additional information.

Kafka 2.6.0 requires the same ZooKeeper version as Kafka 2.5.x (ZooKeeper version 3.5.7 / 3.5.8). Therefore, the Cluster Operator does *not* perform a ZooKeeper upgrade when upgrading from AMQ Streams 1.5.

1.2. CONTAINER IMAGES MOVE TO JAVA 11

AMQ Streams container images move to Java 11 as the Java runtime environment (JRE). The JRE version in the images changes from OpenJDK 8 to OpenJDK 11.

1.3. CLUSTER OPERATOR LOGGING

Cluster Operator logging is now configured using a ConfigMap that is automatically created when the Cluster Operator is deployed. The ConfigMap is described in the following new YAML file:

```
install/cluster-operator/050-ConfigMap-strimzi-cluster-operator.yaml
```

To configure Cluster Operator logging:

1. In the **050-ConfigMap-strimzi-cluster-operator.yaml** file, edit the **data.log4j2.properties** field:

   Example Cluster Operator logging configuration

   ```
   kind: ConfigMap
   apiVersion: v1
   metadata:
     name: strimzi-cluster-operator
     labels:
       app: strimzi
   data:
     log4j2.properties: |
       name = COConfig
       monitorInterval = 30
       appender.console.type = Console
       appender.console.name = STDOUT
       # ...
   ```

2. Update the custom resource:

   ```
   oc apply -f install/cluster-operator/050-ConfigMap-strimzi-cluster-operator.yaml
   ```

To change the frequency that logs are reloaded, set a time in seconds in the **monitorInterval** field (the default reload time is 30 seconds).

NOTE

As a result of this change, the **STRIMZI_LOG_LEVEL** environment variable has been removed from the **060-Deployment-strimzi-cluster-operator.yaml** file. Set the log level in the ConfigMap instead.

See Cluster Operator configuration.

1.4. OAUTH 2.0 AUTHORIZATION

Support for OAuth 2.0 authorization moves out of Technology Preview to a generally available component of AMQ Streams.

If you are using OAuth 2.0 for token-based authentication, you can now also use OAuth 2.0 authorization rules to constrain client access to Kafka brokers.

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.

Security policies and permissions defined in Red Hat Single Sign-On are used to grant access to resources on Kafka brokers. Users and clients are matched against policies that permit access to perform specific actions on Kafka brokers.

See Using OAuth 2.0 token-based authorization .

1.5. OPEN POLICY AGENT (OPA) INTEGRATION

Open Policy Agent (OPA) is an open-source policy engine. You can integrate OPA with AMQ Streams to act as a policy-based authorization mechanism for permitting client operations on Kafka brokers.

When a request is made from a client, OPA will evaluate the request against policies defined for Kafka access, then allow or deny the request.

You can define access control for Kafka clusters, consumer groups and topics. For instance, you can define an authorization policy that allows write access from a producer client to a specific broker topic.

See **KafkaAuthorizationOpa** schema reference

NOTE

Red Hat does not support the OPA server.

1.6. DEBEZIUM FOR CHANGE DATA CAPTURE INTEGRATION

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

Debezium has multiple uses, including:

- Data replication
- Updating caches and search indexes
- Simplifying monolithic applications
- Data integration
- Enabling streaming queries

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

- MySQL
- PostgreSQL
- SQL Server
- MongoDB

For more information on deploying Debezium with AMQ Streams, refer to the product documentation.

1.7. SERVICE REGISTRY

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

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

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

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

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

See Managing schemas with Service Registry.

CHAPTER 2. ENHANCEMENTS

The enhancements added in this release are outlined below.

2.1. KAFKA ENHANCEMENTS

For an overview of the enhancements introduced with:

- Kafka 2.6.2, refer to the Kafka 2.6.2 Release Notes (applies only to AMQ Streams 1.6.4)
- Kafka 2.6.1, refer to the Kafka 2.6.1 Release Notes (applies only to AMQ Streams 1.6.4)
- Kafka 2.6.0, refer to the Kafka 2.6.0 Release Notes

2.2. KAFKA BRIDGE ENHANCEMENTS

This release includes the following enhancements to the Kafka Bridge component of AMQ Streams.

Retrieve partitions and metadata

The Kafka Bridge now supports the following operations:

- Retrieve a list of partitions for a given topic:

```
GET /topics/{topicname}/partitions
```

- Retrieve metadata for a given partition, such as the partition ID, the leader broker, and the number of replicas:

```
GET /topics/{topicname}/partitions/{partitionid}
```

See the Kafka Bridge API reference.

Support for Kafka message headers

Messages sent using the Kafka Bridge can now include Kafka message headers.

In a POST request to the **/topics** endpoint, you can optionally specify headers in the message payload, which is contained in the request body. Message header values must be in binary format and encoded as Base64.

Example request with Kafka message header

```
curl -X POST \
  http://localhost:8080/topics/my-topic \
  -H 'content-type: application/vnd.kafka.json.v2+json' \
  -d '{
    "records": [
        {
            "key": "my-key",
            "value": "sales-lead-0001"
            "partition": 2
            "headers": [
              {
                "key": "key1",
```

```
            "value": "QXBhY2hlIEthZmthIGlzIHRoZSBib21iIQ=="
          }
        ]
      },
  ]
}'
```

See Requests to the Kafka Bridge

2.3. OAUTH 2.0 AUTHENTICATION AND AUTHORIZATION

This release includes the following enhancements to OAuth 2.0 token-based authentication and authorization.

Session re-authentication

OAuth 2.0 authentication in AMQ Streams now supports *session re-authentication* for Kafka brokers. This defines the maximum duration of an authenticated OAuth 2.0 session between a Kafka client and a Kafka broker. Session re-authentication is supported for both types of token validation: fast local JWT and introspection endpoint.

To configure session re-authentication, use the new **maxSecondsWithoutReauthentication** option in the OAuth 2.0 configuration for Kafka brokers.

For a specific listener, **maxSecondsWithoutReauthentication** allows you to:

- Enable session re-authentication
- Set the maximum duration, in seconds, of an authenticated session between a Kafka client and a Kafka broker

Example configuration for session re-authentication after 1 hour

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
spec:
  kafka:
    listeners:
      #...
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: oauth
          maxSecondsWithoutReauthentication: 3600
          # ...
```

An authenticated session is closed if it exceeds the configured **maxSecondsWithoutReauthentication**, or if the access token expiry time is reached. Then, the client must log in to the authorization server again, obtain a new access token, and then re-authenticate to the Kafka broker. This will establish a new authenticated session over the existing connection.

When re-authentication is next required, any operation that is attempted by the client (apart from re-authentication) will cause the broker to terminate the connection.

See: Session re-authentication for Kafka brokers and Configuring OAuth 2.0 support for Kafka brokers.

JWKS keys refresh interval

When configuring Kafka brokers to use fast local JWT token validation, you can now set the **jwksMinRefreshPauseSeconds** option in the external listener configuration. This defines the minimum interval between attempts by the broker to refresh JSON Web Key Set (JWKS) public keys issued by the authorization server.

With this release, the Kafka broker will attempt to refresh JWKS keys immediately, without waiting for the regular refresh schedule, if it detects an unknown signing key.

Example configuration for a 2-minute pause between attempts to refresh JWKS keys

```
listeners:
  #...
  - name: external2
    port: 9095
    type: loadbalancer
    tls: false
    authentication:
      type: oauth
      validIssuerUri: <https://<auth-server-address>/auth/realms/external>
      jwksEndpointUri: <https://<auth-server-address>/auth/realms/external/protocol/openid-connect/certs>
      userNameClaim: preferred_username
      tlsTrustedCertificates:
      - secretName: oauth-server-cert
        certificate: ca.crt
      disableTlsHostnameVerification: true
      jwksExpirySeconds: 360
      jwksRefreshSeconds: 300
      jwksMinRefreshPauseSeconds: 120
      enableECDSA: "true"
```

The refresh schedule for JWKS keys is set in the **jwksRefreshSeconds** option. When an unknown signing key is encountered, a JWKS keys refresh is scheduled outside of the refresh schedule. The refresh will not start until the time since the last refresh reaches the interval specified in **jwksMinRefreshPauseSeconds**.

jwksMinRefreshPauseSeconds has a default value of **1**.

See Configuring OAuth 2.0 support for Kafka brokers.

Refreshing grants from Red Hat Single Sign-On

New configuration options have been added for OAuth 2.0 token-based authorization through Red Hat Single Sign-On. When configuring Kafka brokers, you can now define the following options related to refreshing grants from Red Hat SSO Authorization Services:

- **grantsRefreshPeriodSeconds**: The time between two consecutive grants refresh runs. The default value is **60**. If set to **0** or less, refreshing of grants is disabled.
- **grantsRefreshPoolSize**: The number of threads that can fetch grants for the active session in parallel. The default value is **5**.

See Using OAuth 2.0 token-based authorization and Configuring OAuth 2.0 authorization support.

Detection of permission changes in Red Hat Single Sign-On

With this release, the **keycloak** (Red Hat SSO) authorization regularly checks for changes in permissions for the active sessions. User changes and permissions management changes are now detected in real time.

2.4. METRICS FOR KAFKA BRIDGE AND CRUISE CONTROL

You can now add metrics configuration to Kafka Bridge and Cruise Control.

Example metrics files for Kafka Bridge and Cruise Control are provided with AMQ Streams, including:

- Custom resource YAML files with metrics configuration
- Grafana dashboard JSON files

With the metrics configuration deployed, and Prometheus and Grafana set up, you can use the example Grafana dashboards for monitoring.

Example metrics files provided with AMQ Streams

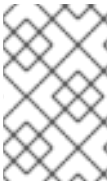
```
metrics
├── grafana-dashboards
│   ├── strimzi-cruise-control.json
│   ├── strimzi-kafka-bridge.json
│   ├── strimzi-kafka-connect.json
│   ├── strimzi-kafka-exporter.json
│   ├── strimzi-kafka-mirror-maker-2.json
│   ├── strimzi-kafka.json
│   ├── strimzi-operators.json
│   └── strimzi-zookeeper.json
├── grafana-install
│   └── grafana.yaml
├── prometheus-additional-properties
│   └── prometheus-additional.yaml
├── prometheus-alertmanager-config
│   └── alert-manager-config.yaml
├── prometheus-install
│   ├── alert-manager.yaml
│   ├── prometheus-rules.yaml
│   ├── prometheus.yaml
│   ├── strimzi-pod-monitor.yaml
├── kafka-bridge-metrics.yaml
├── kafka-connect-metrics.yaml
├── kafka-cruise-control-metrics.yaml
├── kafka-metrics.yaml
└── kafka-mirror-maker-2-metrics.yaml
```

Table 2.1. Example custom resources with metrics configuration

Component	Custom resource	Example YAML file
Kafka and ZooKeeper	**Kafka**	**kafka-metrics.yaml**

Component	Custom resource	Example YAML file
Kafka Connect	**KafkaConnect** and **KafkaConnectS2I**	**kafka-connect-metrics.yaml**
Kafka MirrorMaker 2.0	**KafkaMirrorMaker2**	**kafka-mirror-maker-2-metrics.yaml**
Kafka Bridge	**KafkaBridge**	**kafka-bridge-metrics.yaml**
Cruise Control	**Kafka**	**kafka-cruise-control-metrics.yaml**

See Introducing Metrics to Kafka .



NOTE

The Prometheus server is not supported as part of the AMQ Streams distribution. However, the Prometheus endpoint and JMX exporter used to expose the metrics are supported.

2.5. DYNAMIC UPDATES FOR LOGGING CHANGES

With this release, changing the logging levels, both inline and external, of most custom resources no longer triggers rolling updates to the Kafka cluster. Logging changes are now applied dynamically (without a restart).

This enhancement applies to the following resources:

- Kafka clusters
- Kafka Connect and Kafka Connect S2I
- Kafka Mirror Maker 2.0
- Kafka Bridge

It does not apply to Mirror Maker or Cruise Control.

If you use external logging via a ConfigMap, a rolling update is still triggered when you change a logging appender. For example:

```
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
```

See External logging and the Deployment configuration chapter of the *Using AMQ Streams on OpenShift* guide.

2.6. PODMONITORS USED FOR METRICS SCRAPING

The way that Prometheus metrics are scraped from pods (for Kafka, ZooKeeper, Kafka Connect, and others) has changed in this release.

Metrics are now scraped from pods by **PodMonitors** only, defined in **strimzi-pod-monitor.yaml**. Previously, this was performed by **ServiceMonitors** and **PodMonitors**. **ServiceMonitors** have been **removed** from AMQ Streams in this release.

You need to upgrade your monitoring stack to use **PodMonitors** as described in Upgrading your monitoring stack to use PodMonitors, below.

As a result of this change, the following elements have been **removed** from services related to Kafka and ZooKeeper:

- The **tcp-prometheus** monitoring port (port 9404)
- Prometheus annotations

This change applies to the following services:

- **cluster-name-zookeeper-client**
- **cluster-name-kafka-brokers**

To add a Prometheus annotation, you should now use the **template** property in the relevant AMQ Streams custom resource, as described in Customizing OpenShift resources.

Upgrading your monitoring stack to use PodMonitors

To avoid an interruption to the monitoring of your Kafka cluster, perform the following steps **before** upgrading to AMQ Streams 1.6.

1. Using the new **AMQ Streams 1.6** installation artifacts, apply the **strimzi-pod-monitor.yaml** file to your AMQ Streams 1.5 cluster:

```
oc apply -f examples/metrics/prometheus-install/strimzi-pod-monitor.yaml
```

2. Delete the existing **ServiceMonitor** resources from your AMQ Streams 1.5 cluster.
3. Delete the **Secret** named **additional-scrape-configs**.
4. Create a new **Secret**, also named **additional-scrape-configs**, from the **prometheus-additional.yaml** file provided in the **AMQ Streams 1.6** installation artifacts.
5. Check that the Prometheus targets for the Prometheus user interface are up and running again.
6. Proceed with the upgrade to AMQ Streams 1.6, starting with Upgrading the Cluster Operator.

After completing the upgrade to AMQ Streams 1.6, you can load the example Grafana dashboards for AMQ Streams 1.6.

See Introducing Metrics to Kafka.

2.7. GENERIC LISTENER CONFIGURATION

A **GenericKafkaListener** schema is introduced in this release.

The schema is for the configuration of Kafka listeners in a **Kafka** resource, and replaces the **KafkaListeners** schema, which is deprecated.

With the **GenericKafkaListener** schema, you can configure as many listeners as required, as long as their names and ports are unique. The **listeners** configuration is defined as an array, but the deprecated format is also supported.

See **GenericKafkaListener** schema reference

Updating listeners to the new configuration

The **KafkaListeners** schema uses sub-properties for **plain**, **tls** and **external** listeners, with fixed ports for each. After a Kafka upgrade, you can convert listeners configured using the **KafkaListeners** schema into the format of the **GenericKafkaListener** schema.

For example, if you are currently using the following configuration in your **Kafka** configuration:

Old listener configuration

```
listeners:
  plain:
    # ...
  tls:
    # ...
  external:
    type: loadbalancer
    # ...
```

Convert the listeners into the new format using:

New listener configuration

```
listeners:
  #...
  - name: plain
    port: 9092
    type: internal
    tls: false (1)
  - name: tls
    port: 9093
    type: internal
    tls: true
  - name: external
    port: 9094
    type: EXTERNAL-LISTENER-TYPE (2)
    tls: true
```

1. The TLS property is now required for all listeners.
2. Options: **ingress**, **loadbalancer**, **nodeport**, **route**.

Make sure to use the the **exact** names and port numbers shown.

For any additional **configuration** or **overrides** properties used with the old format, you need to update them to the new format.

Changes introduced to the listener **configuration**:

- **overrides** is merged with the **configuration** section

- **dnsAnnotations** has been renamed **annotations**
- **preferredAddressType** has been renamed **preferredNodePortAddressType**
- **address** has been renamed **alternativenames**
- **loadBalancerSourceRanges** and **externalTrafficPolicy** move to the listener configuration from the now deprecated **template**

All listeners now support configuring the advertised hostname and port.

For example, this configuration:

Old additional listener configuration

```
listeners:
  external:
    type: loadbalancer
    authentication:
      type: tls
    overrides:
      bootstrap:
        dnsAnnotations:
          #...
```

Changes to:

New additional listener configuration

```
listeners:
    #...
  - name: external
    port: 9094
    type:loadbalancer
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        annotations:
          #...
```

IMPORTANT

The name and port numbers shown in the new listener configuration **must** be used for backwards compatibility. Using any other values will cause renaming of the Kafka listeners and Kubernetes services.

2.8. MIRRORMAKER 2.0 TOPIC RENAMING UPDATE

The MirrorMaker 2.0 architecture supports bidirectional replication by automatically renaming remote topics to represent the source cluster. The name of the originating cluster is prepended to the name of the topic.

Optionally, you can now override automatic renaming by adding **IdentityReplicationPolicy** to the source connector configuration. With this configuration applied, topics retain their original names.

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  #...
  mirrors:
  - sourceCluster: "my-cluster-source"
    targetCluster: "my-cluster-target"
    sourceConnector:
      config:
        replication.factor: 1
        offset-syncs.topic.replication.factor: 1
        sync.topic.acls.enabled: "false"
        replication.policy.separator: ""
        replication.policy.class: "io.strimzi.kafka.connect.mirror.IdentityReplicationPolicy" (1)
  #...
```

(1) Adds a policy that overrides the automatic renaming of remote topics. Instead of prepending the name with the name of the source cluster, the topic retains its original name.

The override is useful, for example, in an *active/passive* cluster configuration where you want to make backups or migrate data to another cluster. In either situation, you might not want automatic renaming of remote topics.

See Kafka MirrorMaker 2.0 configuration

2.9. SUPPORT FOR HOSTALIASES

It is now possible to configure **hostAliases** when customizing a deployment of Kubernetes templates and pods.

Example hostAliases configuration

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
#...
spec:
  # ...
  template:
    pod:
      hostAliases:
      - ip: "192.168.1.86"
        hostnames:
        - "my-host-1"
        - "my-host-2"
      #...
```

If a list of hosts and IPs is specified, they are injected into the **/etc/hosts** file of the pod. This is especially useful for Kafka Connect or MirrorMaker when a connection outside of the cluster is also requested by users.

See **PodTemplate** schema reference

2.10. RECONCILED RESOURCE METRIC

A new operator metric provides information about the status of a specified resource, that is, whether or not it was reconciled successfully.

Reconciled resource metric definition

```
strimzi_resource_state{kind="Kafka", name="my-cluster", resource-namespace="my-kafka-namespace"}
```

2.11. SECRET METADATA FOR KAFKAUSER

You can now use template properties for the **Secret** created by the User Operator. Using **KafkaUserTemplate**, you can use labels and annotations to configure metadata that defines how the **Secret** is generated for the **KafkaUser** resource.

An example showing the KafkaUserTemplate

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: tls
  template:
    secret:
      metadata:
        labels:
          label1: value1
        annotations:
          anno1: value1
  # ...
```

See **KafkaUserTemplate** schema reference

2.12. ADDITIONAL TOOLS IN CONTAINER IMAGES

The following tools have been added to the AMQ Streams container images:

- **jstack**
- **jcmd**
- **jmap**
- **netstat** (**net-tools**)
- **lsof**

2.13. REMOVAL OF KAFKA EXPORTER SERVICE

The Kafka Exporter service has been **removed** from AMQ Streams. This service is no longer required because Prometheus now scrapes the Kafka Exporter metrics directly from the Kafka Exporter pods through the **PodMonitor** declaration.

See Introducing Metrics to Kafka .

2.14. DEPRECATION OF ZOOKEEPER OPTION IN KAFKA ADMINISTRATIVE TOOLS

The **--zookeeper** option was deprecated in the following Kafka administrative tools:

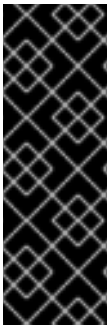
- **bin/kafka-configs.sh**
- **bin/kafka-leader-election.sh**
- **bin/kafka-topics.sh**

When using these tools, you should now use the **--bootstrap-server** option to specify the Kafka broker to connect to. For example:

```
kubectl exec BROKER-POD -c kafka -it -- \
  /bin/kafka-topics.sh --bootstrap-server localhost:9092 --list
```

Although the **--zookeeper** option still works, it will be removed from all the administrative tools in a future Kafka release. This is part of ongoing work in the Apache Kafka project to remove Kafka's dependency on ZooKeeper.

CHAPTER 3. TECHNOLOGY PREVIEWS



IMPORTANT

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

3.1. CLUSTER REBALANCING WITH CRUISE CONTROL

Cruise Control stays in Technology Preview in this release, with some new enhancements.

You can now deploy Cruise Control to your AMQ Streams cluster and use it to rebalance the Kafka cluster using *optimization goals* - predefined constraints on CPU, disk, and network load. In a balanced Kafka cluster, the workload is more evenly distributed across the broker pods.

Cruise Control is configured and deployed as part of a **Kafka** resource. Example YAML configuration files for Cruise Control are provided in **examples/cruise-control/**.

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

- Generate optimization proposals from multiple optimization goals
- Rebalance a Kafka cluster based on an optimization proposal

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

See Cruise Control for cluster rebalancing.

3.1.1. Enhancements to the Technology Preview

The following enhancements have been added to the initial Technology Preview of cluster rebalancing with Cruise Control.

Rebalance performance tuning

Five new *performance tuning options* allow you to control how cluster rebalances are executed and reduce their performance impact.

For each batch of *partition reassignment commands* that comprise a cluster rebalance, you can configure the following:

- Maximum concurrent partition movements per broker (the default is 5)
- Maximum concurrent intra-broker partition movements (the default is 2)
- Maximum concurrent leader movements (the default is 1000)
- Bandwidth in bytes-per-second to assign to partition reassignment (the default is no limit)
- Replica movement strategy (the default is **BaseReplicaMovementStrategy**)

Previously, AMQ Streams inherited these options from Cruise Control, so their default values could not be adjusted.

You can set performance tuning options for the Cruise Control server, individual rebalances, or both.

- For the Cruise Control server, set options in the **Kafka** custom resource, under **spec.cruiseControl.config**.
- For a cluster rebalance, set options in the **spec** property of the **KafkaRebalance** custom resource.

See Rebalance performance tuning overview.

Exclude topics from optimization proposals

You can now exclude one or more topics from an optimization proposal. Those topics are not included in the calculation of partition replica and partition leadership movements for the cluster rebalance.

To exclude topics, specify a regular expression matching the topic names in the **KafkaRebalance** custom resource, in the **spec.excludedTopicsRegex** property.

When viewing a generated optimization proposal, the **excludedTopics** property shows you the topics that were excluded.

See Rebalance performance tuning overview.

CPU capacity goal support

Rebalancing a Kafka cluster based on CPU capacity is now supported through the following configurations:

- The **CpuCapacityGoal** optimization goal
- The **cpuUtilization** capacity limit

The CPU capacity goal prevents the CPU utilization of each broker from exceeding a maximum percentage threshold. The default threshold is set as 100% of CPU capacity per broker.

To reduce the percentage threshold, configure the **cpuUtilization** capacity limit in the **Kafka** custom resource. Capacity limits apply to all brokers.

CPU capacity is preset as a hard goal in Cruise Control. Therefore, it is inherited from Cruise Control as a hard goal, unless you override the preset hard goals in the **hard.goals** property in **Kafka.spec.cruiseControl.config**.

Example configuration for CPU capacity goal in a KafkaRebalance custom resource

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaRebalance
metadata:
  name: my-rebalance
  labels:
    strimzi.io/cluster: amq-streams-cluster
spec:
  goals:
```

```
    - CpuCapacityGoal
    - DiskCapacityGoal
  #...
```

Example configuration for percentage CPU utilization capacity limit

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: amq-streams-cluster
spec:
  # ...
  cruiseControl:
    # ...
    brokerCapacity:
      cpuUtilization: 85
      disk: 100Gi
    # ...
```

See Optimization goals overview and Capacity configuration.

CHAPTER 4. DEPRECATED FEATURES

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

4.1. KAFKALISTENERS SCHEMA

The **KafkaListeners** schema in the **Kafka.KafkaSpec.KafkaClusterSpec** resource is deprecated.

For more information, see Generic listener configuration in the Enhancements section.

CHAPTER 5. FIXED ISSUES

The following sections list the issues fixed in AMQ Streams 1.6.x. Red Hat recommends that you upgrade to the latest patch release if you are using AMQ Streams 1.6.x with OpenShift Container Platform 3.11

For details of the issues fixed in:

- Kafka 2.6.3, refer to the Kafka 2.6.3 Release Notes
- Kafka 2.6.2, refer to the Kafka 2.6.2 Release Notes
- Kafka 2.6.1, refer to the Kafka 2.6.1 Release Notes
- Kafka 2.6.0, refer to the Kafka 2.6.0 Release Notes

5.1. FIXED ISSUES FOR AMQ STREAMS 1.6.7

The AMQ Streams 1.6.7 patch release (Long Term Support) is now available.

AMQ Streams 1.6.7 is the latest Long Term Support release for use with OpenShift Container Platform 3.11 only, and is supported only for as long as OpenShift Container Platform 3.11 is supported.

Note that AMQ Streams 1.6.7 is supported on OCP 3.11 only.

The AMQ Streams product images have been upgraded to version 1.6.7.

For additional details about the issues resolved in AMQ Streams 1.6.7, see AMQ Streams 1.6.x Resolved Issues.

Log4j vulnerabilities

AMQ Streams includes log4j 1.2.17. The release fixes a number of log4j vulnerabilities.

For more information on the vulnerabilities addressed in this release, see the following CVE articles:

- CVE-2022-23307
- CVE-2022-23305
- CVE-2022-23302
- CVE-2021-4104
- CVE-2020-9488
- CVE-2019-17571
- CVE-2017-5645

5.2. FIXED ISSUES FOR AMQ STREAMS 1.6.6

For additional details about the issues resolved in AMQ Streams 1.6.6, see AMQ Streams 1.6.x Resolved Issues.

Log4j2 vulnerabilities

AMQ Streams includes log4j2 2.17.1. The release fixes a number of log4j2 vulnerabilities.

For more information on the vulnerabilities addressed in this release, see the following CVE articles:

- CVE-2021-45046
- CVE-2021-45105
- CVE-2021-44832
- CVE-2021-44228

5.3. FIXED ISSUES FOR AMQ STREAMS 1.6.5

For additional details about the issues resolved in AMQ Streams 1.6.5, see AMQ Streams 1.6.x Resolved Issues.

Log4j2 vulnerability

The 1.6.5 release fixes a remote code execution vulnerability for AMQ Streams components that use log4j2. The vulnerability could allow a remote code execution on the server if the system logs a string value from an unauthorized source. This affects log4j versions between 2.0 and 2.14.1.

For more information, see CVE-2021-44228.

5.4. FIXED ISSUES FOR AMQ STREAMS 1.6.4

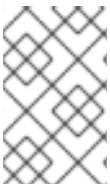
For additional details about the issues resolved in AMQ Streams 1.6.4, see AMQ Streams 1.6.x Resolved Issues.

5.5. FIXED ISSUES FOR AMQ STREAMS 1.6.2

The AMQ Streams 1.6.2 patch release is now available. The release includes a number fixes related to Kafka Connect.

The AMQ Streams product images have not changed and remain at version 1.6.

For additional details about the issues resolved in AMQ Streams 1.6.2, see AMQ Streams 1.6.2 Resolved Issues.



NOTE

Following a CVE update, the version of AMQ Streams managed by the Operator Lifecycle Manager (OLM) was changed to 1.6.1. To avoid confusion, the patch release for AMQ Streams 1.6 was given a version number of 1.6.2.

5.6. FIXED ISSUES FOR AMQ STREAMS 1.6.0

Issue Number	Description
ENTMQST-2049	Kafka Bridge: Kafka consumer should be tracked with group-consumerid key

Issue Number	Description
ENTMQST-2289	Allow downgrade with message version older than the downgrade version
ENTMQST-2292	Diff PodDisruptionBudgets before patching them to not recreate them on every reconciliation
ENTMQST-2146	MirrorMaker 2 on OCP doesn't properly mirror messages with headers
ENTMQST-2147	MirrorMaker 2 doesn't properly configure Jaeger tracing in the connectors
ENTMQST-2099	When set to blank value for toleration Kafka cluster keeps rolling updates repeatedly
ENTMQST-2084	Zookeeper version on the docs doesn't match with the version in AMQ Streams 1.5
ENTMQST-2340	Connection Leak in Operator when Using KafkaConnect API
ENTMQST-2338	Fix Secrets or ConfigMaps with dots mounted into Connect
ENTMQST-2294	OLM install - yaml contains typo for 'authentication'

Table 5.1. Fixed common vulnerabilities and exposures (CVEs)

Issue Number	Description
ENTMQST-2332	CVE-2020-13956 httpclient: apache-httpclient: incorrect handling of malformed authority component in request URIs [amq-st-1]

CHAPTER 6. KNOWN ISSUES

This section lists the known issues for AMQ Streams.

Issue Number

ENTMQST-2386 - Adding or removing JBOD volumes does not work

Description and workaround

AMQ Streams is not able to add or remove JBOD volumes for newer Kubernetes versions. Improved validation in the **StatefulSet Controller** means pods can only be deleted and recreated in sequence (pod-0, then pod-1, and so on). Currently, the AMQ Streams rolling update procedure does not trigger pod deletions in sequence.

The current workaround is to manually delete all pods in sequence. After the pods are recreated, the **StatefulSet Controller** will not fail and operates as expected.

Issue

S2I cannot download source image in environment specified by SHA digest

Description and workaround

Kafka Connect S2I fails to build new connector plugins on disconnected clusters due to an Openshift imagestreams limitation. If the path to the image registry is changed by specifying an **ImageContentSourcePolicy**, it is ignored. Instead, imagestreams will try to download from the source repository.

The current workaround is to deploy Kafka Connect manually.

CHAPTER 7. SUPPORTED INTEGRATION PRODUCTS

AMQ Streams 1.6 supports integration with the following Red Hat products.

- **Red Hat Single Sign-On 7.4 and later** for OAuth 2.0 authentication and OAuth 2.0 authorization
- **Red Hat 3scale API Management 2.6 and later** to secure the Kafka Bridge and provide additional API management features
- **Red Hat Debezium 1.0 and later** for monitoring databases and creating event streams
- **Service Registry 2020-Q4 and later** as 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 AMQ Streams 1.6 documentation.

CHAPTER 8. IMPORTANT LINKS

- Red Hat AMQ 7 Supported Configurations
- Red Hat AMQ 7 Component Details

Revised on 2022-02-02 16:19:20 UTC