Red Hat AMQ Streams 2.0

Release Notes for AMQ Streams 2.0 on OpenShift

Highlights of what is new and what has changed with this AMQ Streams on OpenShift Container Platform release

Last Updated: 2022-06-07

Red Hat AMQ Streams 2.0 Release Notes for AMQ Streams 2.0 on OpenShift

Highlights of what is new and what has changed with this AMQ Streams on OpenShift Container Platform release

Abstract

The release notes summarize the new features, enhancements, and fixes introduced in the AMQ Streams 2.0 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 version 2.0 is based on Strimzi 0.26.x.

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

The AMQ Streams 2.0.1 patch release is now available.

The AMQ Streams product images have been upgraded to version 2.0.1.

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 2.0 is supported on OpenShift Container Platform 4.6 to 4.9.

For more information about the supported platform versions, see the Red Hat Knowledgebase article Red Hat Streams for Apache Kafka Supported Configurations .

1.2. KAFKA 3.0.0 SUPPORT

AMQ Streams now supports Apache Kafka version 3.0.0.

AMQ Streams uses Kafka 3.0.0. Only Kafka distributions built by Red Hat are supported.

You must upgrade the Cluster Operator to AMQ Streams version 2.0 before you can upgrade brokers and client applications to Kafka 3.0.0. For upgrade instructions, see Upgrading AMQ Streams .

Refer to the Kafka 2.8.0 and Kafka 3.0.0 Release Notes for additional information.

NOTE

Kafka 2.8.x is supported only for the purpose of upgrading to AMQ Streams 2.0.

For more information on supported versions, see the Red Hat Streams for Apache Kafka Component Details.

Kafka 3.0.0 requires ZooKeeper version 3.6.3. Therefore, the Cluster Operator will perform a ZooKeeper upgrade when you upgrade from AMQ Streams 1.8 to AMQ Streams 2.0.

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 2.0.

See Deploying and upgrading AMQ Streams .

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 download site .

You perform the custom resources upgrades in two steps.

Step one: Convert the format of custom resources

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

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

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

Step two: Upgrade CRDs to v1beta2

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

For full instructions, see Upgrading AMQ Streams .

1.4. SCALA UPGRADE

AMQ Streams now uses Scala 2.13, which replaces Scala 2.12.

If you are using the Kafka Streams dependency in your Maven build, you must update it to specify Scala 2.13 and Kafka version 3.0.0. If the dependency is not updated, it will not build.

Kafka Streams dependency

```
<dependency>
   <groupId>org.apache.kafka</groupId>
   <artifactId>kafka-streams-scala_2.13</artifactId>
   <version>3.0.0</version>
</dependency>
```

See Kafka Streams Scala dependency

1.5. SUPPORT FOR POWERPC ARCHITECTURE

AMQ Streams 2.0 is enabled to run on PowerPC *ppc64le* architecture.

Support for IBM Power systems applies to AMQ Streams running with Kafka 3.0.0 on OpenShift Container Platform 4.9. The Kafka versions shipped with AMQ Streams 1.8 and earlier versions **do not contain** the ppc64 binaries.

1.5.1. Requirements for IBM Power systems

- OpenShift Container Platform 4.9
- Kafka 3.0.0

1.5.2. Unsupported on IBM Power systems

- Kafka 2.8.0 or older
- AMQ Streams upgrades and downgrades since this is the first release on ppc64le
- AMQ Streams on disconnected OpenShift Container Platform environments
- AMQ Streams OPA integration

1.6. AMQ STREAMS DRAIN CLEANER FOR POD EVICTIONS

AMQ Streams 2.0 introduces the AMQ Streams Drain Cleaner. Use the AMQ Streams Drain Cleaner in combination with the Cluster Operator to move Kafka broker or ZooKeeper pods from nodes that are being drained.

You deploy the AMQ Streams Drain Cleaner to the OpenShift cluster where the Cluster Operator and Kafka cluster are running. By deploying the AMQ Streams Drain Cleaner, you can use the Cluster Operator to move Kafka pods instead of OpenShift.

When you run the AMQ Streams Drain Cleaner, it annotates pods with a rolling update pod annotation. The Cluster Operator performs rolling updates based on the annotation. The Cluster Operator ensures that topics are never under-replicated.

See Evicting pods with AMQ Streams Drain Cleaner

1.7. DEBEZIUM FOR CHANGE DATA CAPTURE INTEGRATION

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

Debezium has multiple uses, including:

- Data replication
- Updating caches and search indexes

- Simplifying monolithic applications
- Data integration
- Enabling streaming queries

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

- Db2
- MongoDB
- MySQL
- PostgreSQL
- SQL Server

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

1.8. SERVICE REGISTRY

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

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

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

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

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

For more information on using Service Registry with AMQ Streams, refer to the Service Registry documentation.

CHAPTER 2. ENHANCEMENTS

The enhancements added in this release are outlined below.

2.1. KAFKA 3.0.0 ENHANCEMENTS

For an overview of the enhancements introduced with Kafka 3.0.0, refer to the Kafka 3.0.0 Release Notes.

2.2. DISABLE AUTOMATIC NETWORK POLICIES FOR LISTENERS

You can now disable the automatic creation of **NetworkPolicy** resources for listeners and use your own custom network policies instead.

By default, AMQ Streams automatically creates a **NetworkPolicy** resource for every listener that is enabled on a Kafka broker. For a particular listener, the network policy allow applications across all namespaces to connect. This behavior can be restricted using **networkPolicyPeers**.

To disable the auto-created **NetworkPolicies**, set the **STRIMZI_NETWORK_POLICY_GENERATION** environment variable to **false** in the Cluster Operator configuration.

See Cluster Operator configuration and Network policies.

2.3. SCRAM USERS NOW MANAGED USING KAFKA ADMIN API

The User Operator now uses the Kafka Admin API instead of ZooKeeper to manage the credentials of SCRAM-SHA-512 users. The Operator connects directly to Kafka for SCRAM-SHA-512 credentials management.

This change is part of ongoing work in the Apache Kafka project to remove Kafka's dependency on ZooKeeper.

Some ZooKeeper related configuration has been deprecated and removed.

See Deprecated features and Using the User Operator.

2.4. USE MAVEN COORDINATES FOR CONNECTOR PLUGINS

You can deploy Kafka Connect with **build** configuration that automatically builds a container image with the connector plugins you require for your data connections. You can now specify the connector plugin artifacts as Maven coordinates.

AMQ Streams supports the following types of artifacts:

- JAR files, which are downloaded and used directly
- TGZ archives, which are downloaded and unpacked
- Maven artifacts, which uses Maven coordinates
- Other artifacts, which are downloaded and used directly

The Maven coordinates identify plugin artifacts and dependencies so that they can be located and fetched from a Maven repository.

Example Maven artifact

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins:
      - name: my-plugin
        artifacts:
          - type: maven (1)
          - repository: https://mvnrepository.com (2)
          - group: org.apache.camel.kafkaconnector (3)
          - artifact: camel-kafka-connector (4)
          - version: 0.11.0 (5)
  #...
```

1. (Required) Type of artifact.
2. (Optional) Maven repository to download the artifacts from. If you do not specify a repository, Maven Central repository is used by default.
3. (Required) Maven group ID.
4. (Required) Maven artifact type.
5. (Required) Maven version number.

See **Build** schema reference

2.5. ENVIRONMENT VARIABLES CONFIGURATION PROVIDER FOR EXTERNAL CONFIGURATION DATA

Use the Environment Variables Configuration Provider plugin to load configuration data from environment variables.

You can use the provider to load configuration data for all Kafka components, including producers and consumers. Use the provider, for example, to provide the credentials for Kafka Connect connector configuration.

The values for the environment variables can be mapped from secrets or config maps. You can use the Environment Variables Configuration Provider, for example, to load certificates or JAAS configuration from environment variables mapped from OpenShift secrets.

See Loading configuration values from external sources

2.6. ALLOW CONNECTOR PLUGIN ARTIFACTS TO BE DOWNLOADED FROM INSECURE SERVERS

It is now possible to download connector plugin artifacts from insecure servers. This applies to JAR, TGZ, and other files that are downloaded and added to a container image. You can set this using the **insecure** property in the connector **build** configuration. When you specify the plugin artifact for download, you set the property to **true**. This disables all TLS verification. If the server is insecure, the artifact still downloads.

Example TGZ plugin artifact allowing insecure download

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins:
      - name: my-plugin
        artifacts:
          - type: tgz
            url: https://my-domain.tld/my-connector-archive.tgz
            sha512sum: 158...jg10
            insecure: true
  #...
```

See **Build** schema reference

2.7. SPECIFY THE SECRET FOR PULLING KAFKA CONNECT BASE IMAGES

You can now specify the container registry secret used to pull base images for Kafka Connect builds on OpenShift. The secret is configured as a **template** customization. You can use the **buildConfig** template in the Kafka Connect **spec** to specify the secret using the **pullSecret** property. The secret contains the credentials for pulling the base image.

Example template customization for a pull secret

```
# ...
template:
  buildConfig:
    metadata:
      labels:
        label1: value1
        label2: value2
      annotations:
        annotation1: value1
        annotation2: value2
    pullSecret: "<secret_credentials>"
# ...
```

2.8. SPECIFY THE VOLUME OF THE /TMP DIRECTORY

You can now configure the total local storage size for /**tmp**, the temporary **emptyDir** volume. The specification currently works for single containers running in a pod. The default storage size is **1Mi**. The size is configured as a **template** customization. You can use the **pod** template in the **spec** of the resource to specify the volume size using the **tmpDirSizeLimit** property.

Example template customization to specify local storage size for /**tmp**

```
# ...
template:
  pod:
    tmpDirSizeLimit: "2Mi"
# ...
```

2.9. METERING

As a cluster administrator, you can use the Metering tool that is available on OpenShift to analyze what is happening in your cluster. You can now use the tool to generate reports and analyze your installed AMQ Streams components. Using Prometheus as a default data source, you can generate reports on pods, namespaces, and most other Kubernetes resources. You can also use the OpenShift Metering operator to analyze your installed AMQ Streams components to determine whether you are in compliance with your Red Hat subscription.

To use metering with AMQ Streams, you must first install and configure the Metering operator on OpenShift Container Platform.

See Using Metering on AMQ Streams

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. 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

3.2. CRUISE CONTROL FOR CLUSTER REBALANCING

NOTE

Cruise Control remains in Technology Preview, with some new enhancements.

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

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

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

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

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

See Cruise Control for cluster rebalancing.

3.2.1. Enhancements to the Technology Preview

Cruise Control REST API security

The Cruise Control REST API is secured with HTTP Basic authentication and SSL to protect the cluster against potentially destructive Cruise Control operations, such as decommissioning Kafka brokers.

The following settings are enabled by default:

- **webserver.security.enable: true**
- **webserver.ssl.enable: true**

You can disable the built-in HTTP Basic authentication or SSL settings, though this is **not recommended**. Configuration options to disable security have been added to **spec.cruiseControl.config** in the **Kafka** resource.

NOTE

When Cruise Control is deployed to an AMQ Streams cluster, you perform operations using custom resources, not the REST API.

See Cruise Control configuration

Anomaly detection for Cruise Control

Cruise Control’s anomaly detection is now enabled for use with AMQ Streams. You can use anomaly detection to monitor Cruise Control metrics. Cruise Control’s anomaly notifier provides alerts on broker failures and other conditions that block the generation of optimization goals, as well as any actions taken.

See Cruise Control for cluster rebalancing

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. JAVA 8

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

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

4.2. ZOOKEEPER CONFIGURATION FOR USER OPERATOR

Two environment variables were **removed** from the standalone User Operator configuration:

- **STRIMZI_ZOOKEEPER_CONNECT**
- **STRIMZI_ZOOKEEPER_SESSION_TIMEOUT_MS**

Previously, these environment variables were set in **install/user-operator/05-Deployment-strimzi-user-operator.yaml**, which is used for deploying the standalone User Operator only.

The User Operator **zookeeperSessionTimeoutSeconds** property is no longer required and is now **deprecated**. This property was set in the **userOperator** property in the **Kafka** resource.

4.3. KAFKA MIRRORMAKER 1

Kafka MirrorMaker replicates data between two or more active Kafka clusters, within or across data centers. Kafka MirrorMaker 1 is deprecated for Kafka 3.0.0 and will be removed in Kafka 4.0.0. MirrorMaker 2.0 will be the only version available. MirrorMaker 2.0 is based on the Kafka Connect framework, connectors managing the transfer of data between clusters.

As a consequence, the AMQ Streams **KafkaMirrorMaker** custom resource which is used to deploy Kafka MirrorMaker 1 has been deprecated. The **KafkaMirrorMaker** resource will be removed from AMQ Streams when Kafka 4.0.0 is adopted.

If you are using MirrorMaker 1 (referred to as just *MirrorMaker* in the AMQ Streams documentation), use the **KafkaMirrorMaker2** custom resource with the **IdentityReplicationPolicy**. MirrorMaker 2.0 renames topics replicated to a target cluster. **IdentityReplicationPolicy** configuration overrides the automatic renaming. Use it to produce the same active/passive unidirectional replication as MirrorMaker 1.

See Kafka MirrorMaker 2.0 cluster configuration

4.4. OPENSHIFT TEMPLATES REMOVED FROM EXAMPLES

The OpenShift templates provided with the AMQ Streams example files have been removed and are no longer supported. The template configuration files were provided to deploy Kafka components from the OpenShift console. For example, templates were included for deploying Kafka with ephemeral or persistent storage. The templates offered some benefit when using OpenShift 3.11. Since OpenShift 4.x, it is easier to use the Cluster Operator to install and manage Kafka components in the OpenShift console.

4.5. SUPPORT FOR KAFKA CONNECT WITH SOURCE-TO-IMAGE (S2I) REMOVED

With the introduction of **build** configuration to the KafkaConnect resource, AMQ Streams can now automatically build a container image with the connector plugins you require for your data connections.

As a result, support for Kafka Connect with Source-to-Image (S2I) was removed.

You can migrate Kafka Connect S2I instances to Kafka Connect instances.

See Migrating from Kafka Connect with S2I to Kafka Connect

CHAPTER 5. FIXED ISSUES

The following sections list the issues fixed in AMQ Streams 2.0.x. Red Hat recommends that you upgrade to the latest patch release.

For details of the issues fixed in Kafka 3.0.0, refer to the Kafka 3.0.0 Release Notes .

5.1. FIXED ISSUES FOR AMQ STREAMS 2.0.1

The AMQ Streams 2.0.1 patch release is now available.

The AMQ Streams product images have been upgraded to version 2.0.1.

For additional details about the issues resolved in AMQ Streams 2.0.1, see AMQ Streams 2.0.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 2.0.0

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

Table 5.1. Fixed issues

Issue Number	Description
ENTMQST-3022	Metrics not working properly for explicit specified watching namespaces
ENTMQST-3053	**strimzi_resource_state** not updated to 0 when operator doesn't see a Kafka resource anymore due to changed selector label
ENTMQST-3207	Add advertised hostnames to certificate SANs even for internal listeners
ENTMQST-3250	Changing log level does not seem to work in Kafka Exporter
ENTMQST-3274	Fix regex validation in CRDs
ENTMQST-3296	Load all certificates in Kafka Exporter
ENTMQST-3297	Use all public keys from Cluster CA in **ZookeeperScaler** and in **DefaultAdminClientProvider**
ENTMQST-3318	Fix reconciliation of TLS users with quotas
ENTMQST-3601	Certificates of internal components are not renewed after executing force-renew

Table 5.2. Fixed common vulnerabilities and exposures (CVEs)

Issue Number	Description
ENTMQST-3146	*CVE-2021-34429* jetty-server: jetty: crafted URIs allow bypassing security constraints
ENTMQST-3307	*CVE-2021-38153* Kafka: Timing attack vulnerability for Apache Kafka Connect and Clients
ENTMQST-3308	*CVE-2021-38153* kafka-clients: Kafka: Timing attack vulnerability for Apache Kafka Connect and Clients
ENTMQST-3316	*CVE-2021-37136* netty-codec: Bzip2Decoder doesn't allow setting size restrictions for decompressed data
ENTMQST-3317	*CVE-2021-37137* netty-codec: SnappyFrameDecoder doesn't restrict chunk length and may buffer skippable chunks in an unnecessary way
ENTMQST-3428	*CVE-2021-37136* netty-codec: Bzip2Decoder doesn't allow setting size restrictions for decompressed data - Drain Cleaner
ENTMQST-3532	*CVE-2021-44228* log4j-core: Remote code execution in Log4j 2.x when logs contain an attacker-controlled string valuer

Issue Number	Description
ENTMQST-3555	*CVE-2021-45046* log4j-core: DoS in log4j2.x with thread context message pattern and context lookup pattern
ENTMQST-3587	*CVE-2021-45105* log4j-core: DoS in log4j 2.x with Thread Context Map (MDC) input data contains a recursive lookup and context lookup pattern
ENTMQST-3602	*CVE-2021-44832* log4j-core: remote code execution through JDBC Appender

CHAPTER 6. KNOWN ISSUES

This section lists the known issues for AMQ Streams 2.0.

6.1. SMTP APPENDER FOR LOG4J

AMQ Streams ships with a potentially vulnerable version of log4j (**log4j-1.2.17.redhat-3**). The vulnerability lies with the SMTP appender functionality, which is not used by AMQ Streams in its default configuration.

Table 6.1. CVE issue

Issue Number	Description
ENTMQST-1934	CVE-2020-9488 log4j: improper validation of certificate with host mismatch in SMTP appender.

Workaround

If you are using the SMTP appender, ensure that **mail.smtp.ssl.checkserveridentity** is set to **true**.

6.2. 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>
```

6.3. 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 the number of logical processors of a node is not equal to the CPU limit of a Kafka broker pod on that node. The issue can prevent cluster rebalances when the pod is under heavy load.

Workaround

You can work around the issue by excluding 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 **OptimizationFailureException** due to insufficient CPU capacity

CHAPTER 7. SUPPORTED INTEGRATION PRODUCTS

AMQ Streams 2.0 supports integration with the following Red Hat products.

Red Hat Single Sign-On 7.4 and later

Provides OAuth 2.0 authentication and OAuth 2.0 authorization.

Red Hat 3scale API Management 2.6 and later

Secures the Kafka Bridge and provides additional API management features.

Red Hat Debezium 1.5

Monitors databases and creates event streams.

Red Hat Service Registry 2.0

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 AMQ Streams 2.0 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 8. IMPORTANT LINKS

- Red Hat Streams for Apache Kafka Supported Configurations
- Red Hat Streams for Apache Kafka Component Details

Revised on 2022-06-07 08:53:58 UTC