Red Hat AMQ 2021.q2

Release Notes for AMQ Streams 1.7 on OpenShift

For use with AMQ Streams on OpenShift Container Platform

Last Updated: 2021-04-23

Red Hat AMQ 2021.q2 Release Notes for AMQ Streams 1.7 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 1.7 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 1.7 is based on Strimzi 0.22.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. OPENSHIFT CONTAINER PLATFORM SUPPORT

AMQ Streams 1.7 is supported on OpenShift Container Platform 4.6 and 4.7.

For more information about the supported platform versions, see the Red Hat Knowledgebase article Red Hat AMQ 7 Supported Configurations .

1.2. KAFKA 2.7.0 SUPPORT

AMQ Streams now supports Apache Kafka version 2.7.0.

AMQ Streams uses Kafka 2.7.0. Only Kafka distributions built by Red Hat are supported.

You must upgrade the Cluster Operator to AMQ Streams version 1.7 before you can upgrade brokers and client applications to Kafka 2.7.0. For upgrade instructions, see Upgrading AMQ Streams .

Refer to the Kafka 2.6.0 and Kafka 2.7.0 Release Notes for additional information.

NOTE

Kafka 2.6.x is supported only for the purpose of upgrading to AMQ Streams 1.7.

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

Kafka 2.7.0 requires the same ZooKeeper version as Kafka 2.6.x (ZooKeeper version 3.5.8). Therefore, the Cluster Operator does *not* perform a ZooKeeper upgrade when you upgrade from AMQ Streams 1.6 to AMQ Streams 1.7.

1.3. INTRODUCING THE V1BETA2 API VERSION

AMQ Streams 1.7 introduces the **v1beta2** API version, which updates the schemas of the AMQ Streams custom resources. Older API versions are now deprecated.

After you have upgraded to AMQ Streams 1.7, you **must** upgrade your custom resources to use API version **v1beta2**. You can do this any time after upgrading, but the upgrades **must be completed before the next AMQ Streams minor version update (AMQ Streams 1.8)**.

The **Red Hat AMQ Streams 1.7.0 API Conversion Tool** is provided to support the upgrade of custom resources. You can download the API conversion tool from the AMQ Streams download site . Instructions for using the tool are included in the documentation and the provided readme.

Upgrading custom resources to **v1beta2** prepares AMQ Streams for Kubernetes CRD **v1**, which will be required for Kubernetes 1.22.

1.3.1. Upgrading custom resources to v1beta2

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 AMQ Streams custom resource upgrades

1.3.2. kubectl apply commands with v1beta2

After upgrading custom resources to **v1beta2**, the **kubectl apply** command no longer works when performing some tasks. You need to use alternative commands in order to accommodate the larger file sizes of **v1beta2** custom resources.

Use **kubectl create -f** instead of **kubectl apply -f** when deploying the AMQ Streams Operators.

Use **kubectl replace -f** instead of **kubectl apply -f** when:

- Upgrading the Cluster Operator
- Downgrading the Cluster Operator to a previous version

If the custom resource you are creating already exists (for example, if it is already installed through a different namespace) use **kubectl replace**. If the custom resource does not exist, use **kubectl create**.

See Deploying and upgrading AMQ Streams

1.4. MULTI-VERSION PRODUCT UPGRADES

You can now upgrade from a previous AMQ Streams version directly to the latest AMQ Streams version within a single upgrade. For example, upgrading from AMQ Streams 1.5 directly to AMQ Streams 1.7, skipping intermediate versions.

See Upgrading AMQ Streams

1.5. KAFKA CONNECT BUILD CONFIGURATION

You can now use **build** configuration so that AMQ Streams automatically builds a container image with the connector plugins you require for your data connections.

For AMQ Streams to create the new image automatically, the **build** configuration requires **output** properties to reference a container registry that stores the container image, and **plugins** properties to list the connector plugins and their artifacts to add to the image.

A **build** configuration can also reference an imagestream. Imagestreams reference a container image stored in OpenShift Container Platform's integrated registry.

The **output** properties describe the type and name of the image, and optionally the name of the Secret containing the credentials needed to access the container registry. The **plugins** properties describe the type of artifact and the URL from which the artifact is downloaded. Additionally, you can specify a SHA-512 checksum to verify the artifact before unpacking it.

Example Kafka Connect configuration to create a new image automatically

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  # ...
  build:
    output:
      type: docker
      image: my-registry.io/my-org/my-connect-cluster:latest
      pushSecret: my-registry-credentials
    plugins:
      - name: debezium-postgres-connector
        artifacts:
          - type: tgz
            url: https://ARTIFACT-ADDRESS.tgz
            sha512sum: HASH-NUMBER-TO-VERIFY-ARTIFACT
      # ...
  #...
```

NOTE

Support for Kafka Connect with Source-to-Image (S2I) is deprecated, as described in Chapter 4, *Deprecated features*.

See:

- Creating a new container image automatically using AMQ Streams
- Build schema reference

1.6. METRICS CONFIGURATION

Metrics are now configured using a ConfigMap that is automatically created when a custom resource is deployed.

Use the **metricsConfig** property to enable and configure Prometheus metrics for Kafka components. The **metricsConfig** property contains a reference to a ConfigMap containing additional configuration for the Prometheus JMX exporter.

Example metrics configuration for Kafka

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    metricsConfig:
      type: jmxPrometheusExporter
      valueFrom:
        configMapKeyRef:
          name: my-config-map
          key: my-key
    # ...
  zookeeper:
    # ...
```

The ConfigMap stores the YAML configuration for the JMX Prometheus exporter under a key.

Example ConfigMap with metrics configuration for Kafka

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: my-configmap
data:
  my-key: |
    lowercaseOutputName: true
    rules:
    # Special cases and very specific rules
    - pattern: kafka.server<type=(.+), name=(.+), clientId=(.+), topic=(.+), partition=(.*)><>Value
      name: kafka_server_$1_$2
      type: GAUGE
      labels:
       clientId: "$3"
       topic: "$4"
       partition: "$5"
    # further configuration
```

To enable Prometheus metrics export without further configuration, you can reference a ConfigMap containing an empty file under **metricsConfig.valueFrom.configMapKeyRef.key**. When referencing an empty file, all metrics are exposed as long as they have not been renamed.

The **spec.metrics** property is deprecated, as described in Chapter 4, *Deprecated features*.

See Common configuration properties

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 product documentation.

CHAPTER 2. ENHANCEMENTS

The enhancements added in this release are outlined below.

2.1. KAFKA 2.7.0 ENHANCEMENTS

For an overview of the enhancements introduced with Kafka 2.7.0, refer to the Kafka 2.7.0 Release Notes.

2.2. CONFIGURING THE DEPLOYMENT STRATEGY

You can now configure the Deployment strategy for Kafka Connect, MirrorMaker, and the Kafka Bridge.

The **RollingUpdate** strategy is used by default for all resources. During a rolling update of a Kafka cluster, the old and new pods in the **Deployment** are run in parallel. This is the optimal strategy for most use cases.

To reduce resource consumption, you can choose the **Recreate** strategy. With this strategy, during a rolling update, the old pods in the **Deployment** are terminated before any new pods are created.

You set the Deployment strategy in **spec.template.deployment** in the **KafkaConnect**, **KafkaMirrorMaker**, **KafkaMirrorMaker2**, and **KafkaBridge** resources.

Example of Recreate Deployment strategy for Kafka Connect

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  template:
    deployment:
      deploymentStrategy: Recreate
  #...
```

If **spec.template.deployment** is not configured, the **RollingUpdate** strategy is used.

See **DeploymentTemplate** schema reference

2.3. DISABLING OWNER REFERENCE IN CA SECRETS

Cluster and Client CA Secrets are created with an **ownerReference** field, which is set to the **Kafka** custom resource.

Now, you can disable the CA Secrets **ownerReference** by adding the **generateSecretOwnerReference: false** property to your Kafka cluster configuration. If the **ownerReference** for a CA Secret is disabled, the Secret is not deleted by OpenShift when the corresponding **Kafka** custom resource is deleted. The CA Secret is then available for reuse with a new Kafka cluster.

Example configuration to disable ownerReference in Cluster and Client CA Secrets

```
apiVersion: kafka.strimzi.io/v1beta1
```

```
kind: Kafka
# ...
spec:
# ...
  clusterCa:
    generateCertificateAuthority: true
    generateSecretOwnerReference: false
  clientsCa:
    generateCertificateAuthority: true
    generateSecretOwnerReference: false
# ...
```

See Disabling **ownerReference** in the CA Secrets

2.4. PREFIX FOR KAFKA USER SECRET NAME

You can now use the **secretPrefix** property to configure the User Operator, which adds a prefix to all secret names created for a **KafkaUser** resource.

For example, this configuration:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    userOperator:
      secretPrefix: kafka-
    # ...
```

Creates a secret named **kafka-my-user** for a user named **my-user**.

See **EntityUserOperatorSpec** schema reference

2.5. ROLLING INDIVIDUAL KAFKA AND ZOOKEEPER PODS THROUGH THE CLUSTER OPERATOR

Using an annotation, you can manually trigger a rolling update of an existing pod that is part of the Kafka cluster or ZooKeeper cluster StatefulSets. When multiple pods from the same StatefulSet are annotated at the same time, consecutive rolling updates are performed within the same reconciliation run.

See Performing a rolling update using a Pod annotation

2.6. TOPIC OPERATOR TOPIC STORE

AMQ Streams no longer uses ZooKeeper to store topic metadata. Topic metadata is now brought into the Kafka cluster, and under the control of the Topic Operator.

This change is required to prepare AMQ Streams for the future removal of ZooKeeper as a Kafka dependency.

The Topic Operator now uses persistent storage to store topic metadata describing topic configuration as key-value pairs. Topic metadata is accessed locally in-memory. Updates from operations applied to the local in-memory topic store are persisted to a backup topic store on disk. The topic store is continually synchronized with updates from Kafka topics.

When upgrading to AMQ Streams 1.7, the transition to Topic Operator control of the topic store is seamless. Metadata is found and migrated from ZooKeeper, and the old store is cleansed.

New internal topics

To support the handling of topic metadata in the topic store, two new internal topics are created in your Kafka cluster when you upgrade to AMQ Streams 1.7:

Internal topic name	Description
__strimzi_store_topic	Input topic for storing the topic metadata.
__strimzi-topic-operator-kstreams-topic-store-changelog	Retains a log of compacted topic store values.

WARNING

Do not delete these topics, as they are essential to the running of the Topic Operator.

See Topic Operator topic store

2.7. JAAS CONFIGURATION

The JAAS configuration string in the **sasl.jaas.config** property has been added to the generated secrets for a **KafkaUser** with SCRAM-SHA-512 authentication.

See SCRAM-SHA-512 Authentication

2.8. CLUSTER IDENTIFICATION FOR KAFKA STATUS

The **KafkaStatus** schema is updated to include the **clusterId** to identify a Kafka cluster. The **status** property of the Kafka resource provides status information on a Kafka cluster.

Kafka status property

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
  # ...
status:
```

```
  conditions:
    lastTransitionTime: "YEAR-MONTH-20T11:37:00.706Z"
    status: "True"
    type: Ready
  observedGeneration: 1
  clusterId: CLUSTER-ID
  # ...
```

When you retrieve the status of a Kafka resource, the id of the Kafka cluster is also returned:

```
oc get kafka MY-KAFKA-CLUSTER -o jsonpath='{.status}'
```

You can also retrieve only the cluster id for the Kafka resource:

```
oc get kafka MY-KAFKA-CLUSTER -o jsonpath='{.status.clusterId}'
```

See **KafkaStatus** schema reference and Finding the status of a custom resource

2.9. KAFKA CONNECT STATUS

When you retrieve the status of a KafkaConnector resource, the list of topics used by the connector is now returned in the **topics** property.

See **KafkaConnectorStatus** schema reference and Finding the status of a custom resource

2.10. RUNNING AMQ STREAMS WITH READ-ONLY ROOT FILE SYSTEM

You can now run AMQ Streams with a read-only root file system. Additional volume has been added so that temporary files are written to a mounted /**tmp** file. Previously, the /**tmp** directory was used directly from the container.

In this way, the container file system does not need to be modified, and AMQ Streams can run unimpeded from a read-only root file system.

2.11. EXAMPLE YAML FILES SPECIFY INTER-BROKER PROTOCOL VERSION

The example Kafka configuration files provided with AMQ Streams now specify the **inter.broker.protocol.version**. The **inter.broker.protocol.version** and **log.message.format.version** properties for the Kafka **config** are the versions supported by the specified Kafka version (**spec.kafka.version**). The properties represent the log format version appended to messages and the version of protocol used in a Kafka cluster. Updates to these properties are required when upgrading your Kafka version.

Specified Kafka versions

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    version: 2.7.0
```

```
  #...
  config:
    #...
    log.message.format.version: 2.7
    inter.broker.protocol.version: 2.7
```

See Upgrading Kafka

2.12. RESTRICTING CLUSTER OPERATOR ACCESS WITH NETWORK POLICY

The Cluster Operator can run in the same namespace as the resources it manages, or in a separate namespace. Two new environment variables now control which namespaces can access the Cluster Operator.

By default, the **STRIMZI_OPERATOR_NAMESPACE** environment variable is configured to use the Kubernetes Downward API to find which namespace the Cluster Operator is running in. If the Cluster Operator is running in the same namespace as the resources, only local access is required, and allowed by Strimzi.

If the Cluster Operator is running in a separate namespace to the resources it manages, any namespace in the Kubernetes cluster is allowed access to the Cluster Operator unless network policy is configured. Use the optional **STRIMZI_OPERATOR_NAMESPACE_LABELS** environment variable to establish network policy for the Cluster Operator using namespace labels. By adding namespace labels, access to the Cluster Operator is restricted to the namespaces specified.

Network policy configured for the Cluster Operator deployment

```
#...
env:
  - name: STRIMZI_OPERATOR_NAMESPACE_LABELS
    value: label1=value1,label2=value2
  #...
```

See Cluster Operator configuration

2.13. ADDING LABELS AND ANNOTATIONS TO SECRETS

By configuring the **clusterCaCert** template property in the **Kafka** custom resource, you can add custom labels and annotations to the Cluster CA Secrets created by the Cluster Operator. Labels and annotations are useful for identifying objects and adding contextual information. You configure template properties in Strimzi custom resources.

Example template customization to add labels and annotations to Secrets

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    template:
      clusterCaCert:
```

```
    metadata:
      labels:
        label1: value1
        label2: value2
      annotations:
        annotation1: value1
        annotation2: value2
  # ...
```

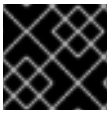
See Customizing OpenShift resources

2.14. PAUSING RECONCILIATION OF CUSTOM RESOURCES

You can pause the reconciliation of a custom resource by setting the **strimzi.io/pause-reconciliation** annotation to **true** in its configuration. For example, you can apply the annotation to the **KafkaConnect** resource so that reconciliation by the Cluster Operator is paused.

Example custom resource with a paused reconciliation condition type

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  annotations:
    strimzi.io/pause-reconciliation: "true"
    strimzi.io/use-connector-resources: "true"
  creationTimestamp: 2021-03-12T10:47:11Z
  #...
spec:
  # ...
status:
  conditions:
  - lastTransitionTime: 2021-03-12T10:47:41.689249Z
    status: "True"
    type: ReconciliationPaused
```



IMPORTANT

It is not currently possible to pause reconciliation of **KafkaTopic** resources.

See Pausing reconciliation of custom resources

2.15. RESTARTING CONNECTORS AND TASKS

Connector instances and their tasks can now be restarted by using Kubernetes annotations on the relevant custom resources.

You can restart connectors for both Kafka Connect and MirrorMaker 2.0, which uses the Kafka Connect framework to replicate data between the source and target Kafka clusters.

- To restart a Kafka Connect connector, you annotate the corresponding **KafkaConnector** custom resource.
- To restart a MirrorMaker 2.0 connector, you annotate the corresponding **KafkaMirrorMaker2** custom resource.

The annotations can also be used to restart a specified task for a connector.

For Kafka Connect, see Performing a restart of a Kafka connector and Performing a restart of a Kafka connector task.

For MirrorMaker 2.0, see Performing a restart of a Kafka MirrorMaker 2.0 connector and Performing a restart of a Kafka MirrorMaker 2.0 connector task.

2.16. OAUTH 2.0 AUTHENTICATION AND AUTHORIZATION

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

Checks on JWT access tokens

You can now configure two additional checks on JWT access tokens. Both of these checks are configured in the OAuth 2.0 configuration for Kafka broker listeners.

Custom claim checks

Custom claim checks impose custom rules on the validation of JWT access tokens by Kafka brokers. They are defined using JsonPath filter queries.

If an access token does not contain the necessary data, it is rejected. When using *introspection endpoint* token validation, the custom check is applied to the introspection endpoint response JSON.

To configure custom claim checks, add the **customClaimCheck** option and define a JsonPath filter query. Custom claim checks are disabled by default.

See Configuring OAuth 2.0 support for Kafka brokers

Audience checks

Your authorization server might provide **aud** (audience) claims in JWT access tokens.

When audience checks are enabled, the Kafka broker rejects tokens that do not contain the broker's **clientId** in their **aud** claims.

To enable audience checks, set the **checkAudience** option to **true**. Audience checks are disabled by default.

See Configuring OAuth 2.0 support for Kafka brokers

Support for OAuth 2.0 over SASL PLAIN authentication

You can now configure the PLAIN mechanism for OAuth 2.0 authentication between Kafka clients and Kafka brokers. Previously, the only supported authentication mechanism was OAUTHBEARER.

PLAIN is a simple authentication mechanism used by all Kafka client tools (including developer tools such as kafkacat). AMQ Streams includes server-side callbacks that enable PLAIN to be used with OAuth 2.0 authentication. These capabilities are referred to as *OAuth 2.0 over PLAIN*.

NOTE

Red Hat recommends using OAUTHBEARER authentication for clients whenever possible. OAUTHBEARER provides a higher level of security than PLAIN because client credentials are *never* shared with Kafka brokers. Consider using PLAIN only with Kafka clients that do not support OAUTHBEARER.

When used with the provided *OAuth 2.0 over PLAIN* callbacks, Kafka clients can authenticate with Kafka brokers using either of the following methods:

- Client ID and secret (by using the OAuth 2.0 client credentials mechanism)
- A long-lived access token, obtained manually at configuration time

To use PLAIN, you must enable it in the **oauth** listener configuration for the Kafka broker. Three new configuration options are now supported:

- **enableOauthBearer**
- **enablePlain**
- **tokenEndpointUri**

Example oauth listener configuration

```
# ...
name: external
port: 9094
type: loadbalancer
tls: true
authentication:
  type: oauth
  # ...
  checkIssuer: false
  fallbackUserNameClaim: client_id
  fallbackUserNamePrefix: client-account-
  validTokenType: bearer
  userInfoEndpointUri: https://OAUTH-SERVER-ADDRESS/auth/realms/external/protocol/openid-connect/userinfo
  enableOauthBearer: false (1)
  enablePlain: true (2)
  tokenEndpointUri: https://OAUTH-SERVER-ADDRESS/auth/realms/external/protocol/openid-connect/token (3)
  #...
```

1. Disables OAUTHBEARER authentication on the listener. If **true** or the option is not specified, OAUTHBEARER authentication is enabled.
2. Enables PLAIN authentication on the listener. Default is **false**.
3. The OAuth 2.0 token endpoint URL to your authorization server. Must be set if **enablePlain** is **true**, and the client ID and secret are used for authentication.

See OAuth 2.0 authentication mechanisms and Configuring OAuth 2.0 support for Kafka brokers

2.17. KAFKA CONNECT RACK PROPERTY

A new **rack** property is now available for Kafka Connect. Rack awareness is configured to spread replicas across different racks. By configuring a **rack** for a Kafka Connect cluster, consumers are allowed to fetch data from the closest replica. This is useful when a Kafka cluster spans multiple datacenters.

A topology key must match the label of a cluster node.

Example rack configuration

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

See **KafkaConnectSpec** schema reference and **KafkaConnectS2ISpec** schema reference

2.18. POD TOPOLOGY SPREAD CONSTRAINTS

Pod topology spread constraints are now supported for the following AMQ Streams custom resources:

- **Kafka**, including:
  - ZooKeeper
  - Entity Operator
- **KafkaConnect**
- **KafkaConnectS2I**
- **KafkaBridge**
- **KafkaMirrorMaker2** and **KafkaMirrorMaker**

Pod topology spread constraints allow you to distribute Kafka related pods across nodes, zones, regions, or other user-defined domains. You can use them together with the existing **affinity** and **tolerations** properties for pod scheduling.

Constraints are specified in the **template.pod.topologySpreadConstraints** property in the relevant custom resource.

Example pod topology spread constraint for Kafka Connect

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
#...
spec:
  # ...
  template:
    pod:
      topologySpreadConstraints:
      - maxSkew: "1"
        whenUnsatisfiable: DoNotSchedule
        labelSelector:
          matchLabels:
            label1: value1
#...
```

See:

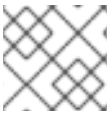
- Specifying affinity, tolerations, and topology spread constraints in the *Using AMQ Streams* guide.
- Controlling pod placement by using pod topology spread constraints in the OpenShift Container Platform documentation.

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. 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 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 Technology Preview of Cruise Control for cluster rebalancing.

New goal: Minimum topic leaders per broker

You can use a new goal named **MinTopicLeadersPerBrokerGoal**.

For each topic in a defined group of topics, the goal ensures that each active broker has at least a certain number of leader replicas.

MinTopicLeadersPerBrokerGoal is a default goal and is preset as a hard goal.

See Optimization goals overview

Logging enhancements: Dynamic logging configuration and Log4j 2

Cruise Control now supports dynamic logging configuration. This means that changing the logging level for Cruise Control no longer triggers a rolling update to the Kafka cluster or the Cruise Control pod.

Log4j 2 is now used for Cruise Control logging.

You **must** update existing configurations for Cruise Control logging from **Log4j** to **Log4j 2** compatible syntax. Logging is configured in the **Kafka** custom resource.

- For *inline logging*, replace the **cruisecontrol.root.logger** property with the **rootLogger.level** property.
- For *external logging*, replace the existing configuration with a new configuration file named **log4j2.properties**. The configuration **must** use **Log4j 2** compatible syntax.

See External logging and Cruise Control 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. KAFKA CONNECT WITH SOURCE-TO-IMAGE (S2I)

AMQ Streams 1.7 introduces **build** configuration to the **KafkaConnect** resource, as described in Chapter 1, *Features*. 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) is deprecated.

To prepare for this change, you can migrate Kafka Connect S2I instances to Kafka Connect instances.

See Migrating from Kafka Connect with S2I to Kafka Connect

4.2. METRICS CONFIGURATION

Metrics configuration is now specified as a ConfigMap for Kafka components. Previously, the **spec.metrics** property was used.

To update the configuration, and enable Prometheus metrics export, a new ConfigMap must be created that matches the configuration for the **.spec.metrics** property. The **.spec.metricsConfig** property is used to specify the ConfigMap, as described in Chapter 1, *Features*.

See Upgrading AMQ Streams

4.3. API VERSIONS

The introduction of **v1beta2** updates the schemas of the custom resources. Older API versions are deprecated.

The **v1alpha1** API version is deprecated for the following AMQ Streams custom resources:

- **Kafka**
- **KafkaConnect**
- **KafkaConnectS2I**
- **KafkaConnector**
- **KafkaMirrorMaker**
- **KafkaMirrorMaker2**
- **KafkaTopic**
- **KafkaUser**
- **KafkaBridge**
- **KafkaRebalance**

The **v1beta1** API version is deprecated for the following AMQ Streams custom resources:

- **Kafka**
- **KafkaConnect**
- **KafkaConnectS2I**
- **KafkaMirrorMaker**
- **KafkaTopic**
- **KafkaUser**

IMPORTANT

The **v1alpha1** and **v1beta1** versions will be removed in the next minor release.

See AMQ Streams custom resource upgrades.

4.4. ANNOTATIONS

The following annotations are deprecated, and will be removed in AMQ Streams 1.8.0:

Table 4.1. Deprecated annotations and their replacements

Deprecated annotation	Replacement annotation
cluster.operator.strimzi.io/delete-claim	**strimzi.io/delete-claim** (Internal)
operator.strimzi.io/generation	**strimzi.io/generation** (Internal)
operator.strimzi.io/delete-pod-and-pvc	**strimzi.io/delete-pod-and-pvc**
operator.strimzi.io/manual-rolling-update	**strimzi.io/manual-rolling-update**

CHAPTER 5. FIXED ISSUES

The issues fixed in AMQ Streams 1.7 are shown in the following table. For details of the issues fixed in Kafka 2.7.0, refer to the Kafka 2.7.0 Release Notes .

Issue Number	Description
ENTMQST-1561	OpenSSL tasks should be executed on separate worker executor and not on the main thread
ENTMQST-1607	Check the **log.message.format.version** and **inter.broker.protocol.version** from the brokers during upgrade
ENTMQST-1631	Topic Operator sometimes renames KafkaTopics
ENTMQST-1676	Simplify Kafka upgrades and downgrades
ENTMQST-1914	Move to Java 11 language level
ENTMQST-2030	If the **bin/kafka-acls.sh** utility is used to add or remove an ACL, the operation is successful but a warning is generated
ENTMQST-2085	All KafkaTopic custom resources are deleted and recreated when restarting kafka and zookeeper at the same time
ENTMQST-2184	Metrics reporter not able to produce metrics when Kafka **min.insync.replicas** is greater than 1
ENTMQST-2188	MirrorMaker: Enable synchronization of offsets to the consumer group on the target cluster
ENTMQST-2269	kafka-configs.sh is deprecating **--zookeeper** option but does not provide alternative functionality to list configs for users
ENTMQST-2295	Watching the wrong reconciliations for metrics in the Topic Operator
ENTMQST-2311	Improve configuration of network policies
ENTMQST-2335	Run the ConnectS2I deployment with **tini** init
ENTMQST-2386	Adding or removing JBOD volumes not working as expected

Issue Number	Description
ENTMQST-2472	Replacing namespace in Prometheus operator bundle file produces invalid YAML
ENTMQST-2480	Inconsistent use of CPU metric in Grafana dashboards
ENTMQST-2483	KafkaConnect Build: Declarative management of connector plugins in Kafka Connect custom resource
ENTMQST-2525	Add annotations to perform connector/task restart operations
ENTMQST-2529	Kafka Exporter dashboard does not auto-select namespace and cluster name
ENTMQST-2547	Network policies not properly configured when metrics ConfigMap is used
ENTMQST-2548	User-certificate renewal never triggered when it expires independently on the CA
ENTMQST-2550	Cruise Control Grafana dashboard presentation is affected when Cruise Control pod is rolled or deleted
ENTMQST-2595	Topic Operator fails to create a topic when replica or partition count is decreased
ENTMQST-2625	JMX configuration issues cause *port already in use* error
ENTMQST-2636	OAuth NullPointerException when using 'keycloak' authorization with 'resource' permissions
ENTMQST-2643	Use the ISO-8601 timestamp standard for Kafka resource status

Table 5.1. Fixed common vulnerabilities and exposures (CVEs)

Issue Number	Title	Description

Issue Number	Title	Description
ENTMQST-2334	CVE-2020-25649 jackson-databind: FasterXML DOMDeserializer insecure entity expansion is vulnerable to XML external entity (XXE) [amq-st-1]	A flaw was found in FasterXML Jackson Databind, where it did not have entity expansion secured properly. This flaw allows vulnerability to XML external entity (XXE) attacks. The highest threat from this vulnerability is data integrity.

CHAPTER 6. KNOWN ISSUES

This section lists the known issues for AMQ Streams 1.7.

6.1. ISSUE WITH DEPLOYING THE CLUSTER OPERATOR TO AN IPV6 CLUSTER

Description and workaround

ENTMQST-2754

The AMQ Streams Cluster Operator does not start on Internet Protocol version 6 (IPv6) clusters.

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-1.7.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-1.7.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
```

6.2. ISSUE WITH 3SCALE DISCOVERY OF THE KAFKA BRIDGE SERVICE

Description and workaround

ENTMQST-2777

Red Hat 3scale cannot discover the Kafka Bridge service as documented in Deploying 3scale for the Kafka Bridge.

Workaround

Perform the following steps in your AMQ Streams cluster to enable service discovery:

1. Edit the **spec** properties for the **KafkaBridge** custom resource.
   Add the **discovery.3scale.net: true** template property to your existing configuration:

   Example template configuration for Kafka Bridge

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: KafkaBridge
   metadata:
     name: KAFKA-BRIDGE-NAME
   spec:
     replicas: 1
     bootstrapServers: my-cluster-kafka-bootstrap:9092
     http:
       port: 8080
     template:
       apiService:
         metadata:
           labels:
             discovery.3scale.net: true (1)
   # ...
   ```

   (1) Enables 3scale to discover the Kafka Bridge service.

2. Create or update the custom resource:

   ```
   kubectl apply -f KAFKA-BRIDGE-CONFIG-FILE
   ```

3. Continue with 3scale service discovery as described in step six of Deploying 3scale for the Kafka Bridge.

CHAPTER 7. SUPPORTED INTEGRATION PRODUCTS

AMQ Streams 1.7 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 1.4 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 1.7 documentation.

CHAPTER 8. IMPORTANT LINKS

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

Revised on 2021-04-23 11:07:11 UTC