Red Hat AMQ 7.7

Release Notes for AMQ Streams 1.5 on OpenShift

For use with AMQ Streams on OpenShift Container Platform

Last Updated: 2020-09-09

Red Hat AMQ 7.7 Release Notes for AMQ Streams 1.5 on OpenShift

For use with AMQ Streams on OpenShift Container Platform

Legal Notice

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

These release notes contain the latest information about new features, enhancements, fixes, and issues contained in the AMQ Streams 1.5 release.

Table of Contents

CHAPTER 1. FEATURES

AMQ Streams version 1.5 is based on Strimzi 0.18.x.

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

1.1. KAFKA 2.5.0 SUPPORT

AMQ Streams now supports Apache Kafka version 2.5.0.

AMQ Streams uses Kafka 2.5.0. Only Kafka distributions built by Red Hat are supported.

You must upgrade the Cluster Operator to AMQ Streams version 1.5 before you can upgrade brokers and client applications to Kafka 2.5.0. For upgrade instructions, see AMQ Streams and Kafka upgrades .

Refer to the Kafka 2.4.0 and Kafka 2.5.0 Release Notes for additional information.

NOTE

Kafka 2.4.x is supported in AMQ Streams 1.5 only for upgrade purposes.

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

1.2. ZOOKEEPER 3.5.8 SUPPORT

Kafka version 2.5.0 requires ZooKeeper version 3.5.8.

You do **not** need to manually upgrade to ZooKeeper 3.5.8; the Cluster Operator performs the ZooKeeper upgrade when it upgrades Kafka brokers. However, you might notice some additional rolling updates during this procedure.

1.3. OPENSHIFT 4.X DISCONNECTED INSTALLATION

Support for *disconnected installation* on OpenShift 4.x moves out of Technology Preview to a generally available component of AMQ Streams.

You can perform a disconnected installation of AMQ Streams when your OpenShift cluster is being used as a *disconnected cluster* on a restricted network.

For a disconnected installation, you obtain the required images and push them to your container registry locally. If you are using the Operator Lifecycle Manager (OLM) this means disabling the default sources used by the OperatorHub and creating local mirrors to install AMQ Streams from local sources.

See Using Operator Lifecycle Manager on restricted networks .

1.4. MIRRORMAKER 2.0

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

MirrorMaker 2.0 is based on the Kafka Connect framework, with *connectors* managing the transfer of data between clusters.

MirrorMaker 2.0 uses:

- Source cluster configuration to consume data from the source cluster
- Target cluster configuration to output data to the target cluster

MirrorMaker 2.0 introduces an entirely new way of replicating data in clusters. If you choose to use MirrorMaker 2.0, there is currently no legacy support, so any resources must be manually converted into the new format.

See Using AMQ Streams with MirrorMaker 2.0 .

1.5. 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.

1.6. 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 2.5.0 ENHANCEMENTS

For an overview of the enhancements introduced with Kafka 2.5.0, refer to the Kafka 2.5.0 Release Notes.

2.2. ZOOKEEPER ENHANCEMENTS

Moving to support for ZooKeeper 3.5.* versions means that it is no longer necessary to use TLS sidecars in ZooKeeper pods. The sidecars were removed, and native ZooKeeper support for TLS is now used.

2.3. EXPANDED OAUTH 2.0 AUTHENTICATION CONFIGURATION OPTIONS

New configuration options make it possible to integrate with a wider set of authorization servers.

Depending on how you apply OAuth 2.0 authentication, and the type of authorization server, there are additional (optional) configuration settings you can use.

Additional configuration options for Kafka brokers

```
# ...
authentication:
  type: oauth
  # ...
  checkIssuer: false (1)
  fallbackUserNameClaim: client_id (2)
  fallbackUserNamePrefix: client-account- (3)
  validTokenType: bearer (4)
  userInfoEndpointUri: https://AUTH-SERVER-ADDRESS/auth/realms/external/protocol/openid-connect/userinfo (5)
```

(1) If your authorization server does not provide an **iss** claim, it is not possible to perform an issuer check. In this situation, set **checkIssuer** to **false** and do not specify a **validIssuerUri**. Default is **true**.

(2) An authorization server may not provide a single attribute to identify both regular users and clients. A client authenticating in its own name might provide a *client ID*. But a user authenticating using a username and password, to obtain a refresh token or an access token, might provide a *username* attribute in addition to a client ID. Use this fallback option to specify the username claim (attribute) to use if a primary user ID attribute is not available.

(3) In situations where **fallbackUserNameClaim** is applicable, it may also be necessary to prevent name collisions between the values of the username claim, and those of the fallback username claim. Consider a situation where a client called **producer** exists, but also a regular user called **producer** exists. In order to differentiate between the two, you can use this property to add a prefix to the user ID of the client.

(4) (Only applicable when using an introspection endpoint URI) Depending on the authorization server you are using, the introspection endpoint may or may not return the *token type* attribute, or it may

contain different values. You can specify a valid token type value that the response from the introspection endpoint has to contain.

5 (Only applicable when using an introspection endpoint URI) The authorization server may be configured or implemented in such a way to not provide any identifiable information in an Introspection Endpoint response. In order to obtain the user ID, you can configure the URI of the **userinfo** endpoint as a fallback. The **userNameClaim**, **fallbackUserNameClaim**, and **fallbackUserNamePrefix** settings are applied to the response of **userinfo** endpoint.

Additional configuration options for Kafka components

```
# ...
spec:
  # ...
  authentication:
    # ...
    disableTlsHostnameVerification: true
    checkAccessTokenType: false
    accessTokenIsJwt: false
    scope: any 1
```

1 The **scope** for requesting the token from the token endpoint. An authorization server may require a client to specify the scope.

See Configuring OAuth 2.0 support for Kafka brokers and Configuring OAuth 2.0 for Kafka components.

2.4. METRICS FOR THE CLUSTER OPERATOR, TOPIC OPERATOR AND USER OPERATOR

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.

As well as monitoring Kafka components, you can now add metrics configuration to monitor the Cluster Operator, Topic Operator and User Operator.

An example Grafana dashboard is provided, which allows you to view the following operator metrics exposed by Prometheus:

- The number of custom resources being processed
- The number of reconciliations being processed
- JVM metrics

See Introducing Metrics to Kafka.

2.5. SSL CONFIGURATION OPTIONS

You can now run external listeners on specific ciphers for a TLS version.

For Kafka components running on AMQ Streams, you can use the three allowed **ssl** configuration options to run external listeners with a specific cipher suite for a TLS version. A cipher suite combines algorithms for secure connection and data transfer.

The example here shows the configuration for a Kafka resource:

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  config: (1)
    # ...
    ssl.cipher.suites: "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384" (2)
    ssl.enabled.protocols: "TLSv1.2" (3)
    ssl.protocol: "TLSv1.2" (4)
```

(2) The cipher suite for TLS using a combination of **ECDHE** key exchange mechanism, **RSA** authentication algorithm, **AES** bulk encyption algorithm and **SHA384** MAC algorithm.

(3) The SSl protocol **TLSv1.2** is enabled.

(4) Specifies the **TLSv1.2** protocol to generate the SSL context.

See Custom Resource API Reference.

2.6. CROSS-ORIGIN RESOURCE SHARING (CORS) FOR KAFKA BRIDGE

You can now enable and define access control for the Kafka Bridge through Cross-Origin Resource Sharing (CORS). CORS is a HTTP mechanism that allows browser access to selected resources from more than one origin. To configure CORS, you define a list of allowed resource origins and HTTP methods to access them. Additional HTTP headers in requests describe the origins that are permitted access to the Kafka cluster.

HTTP configuration for the Kafka Bridge

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  http:
    port: 8080
    cors:
      allowedOrigins: "https://strimzi.io" (1)
      allowedMethods: "GET,POST,PUT,DELETE,OPTIONS,PATCH" (2)
  # ...
```

(1) Comma-separated list of allowed CORS origins. You can use a URL or a Java regular expression.

(2) Comma-separated list of allowed HTTP methods for CORS.

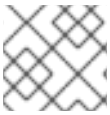
See Kafka Bridge HTTP configuration .

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



NOTE

This is a Technology Preview feature.

You can now deploy Cruise Control to your AMQ Streams cluster and use it to rebalance the Kafka cluster. Cruise Control helps to reduce the time and effort involved in running an efficient and balanced Kafka cluster.

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

By deploying an instance of Cruise Control to each Kafka cluster, you can:

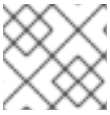
- Generate optimization proposals from multiple optimization goals
- Rebalance a Kafka cluster based on an optimization proposal

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.

There is a known issue related to setting user-provided optimization goals. See Chapter 6, *Known issues*.

3.2. OAUTH 2.0 AUTHORIZATION



NOTE

This is a Technology Preview feature.

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 .

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. API VERSIONS

In the AMQ Streams 1.2 release, the **v1alpha1** versions of the following resources were deprecated and replaced by **v1beta1**:

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

In the next release, the **v1alpha1** versions of these resources will be removed. For guidance on upgrading the resources, see AMQ Streams resource upgrades.

CHAPTER 5. FIXED ISSUES

The following sections list the issues fixed in AMQ Streams 1.5.1 and 1.5.

5.1. FIXED ISSUES FOR AMQ STREAMS 1.5.1

The AMQ Streams 1.5.1 patch release is now available. This micro release updates the Operator Metadata that is used to install AMQ Streams 1.5.0 from the OpenShift OperatorHub user interface. The AMQ Streams product images have not changed and remain at version 1.5.0.

Red Hat recommends that you upgrade to AMQ Streams 1.5.1 if you are using OpenShift Container Platform 4.5.

For additional details about the issues resolved in AMQ Streams 1.5.1, see AMQ Streams 1.5.x Resolved Issues.

5.2. FIXED ISSUES FOR AMQ STREAMS 1.5

Issue Number	Description
ENTMQST-622	Refactor **SimpleAclOperator** to make use of Admin Client API
ENTMQST-643	Fix to generate new cluster or client secret/key when CA key deleted
ENTMQST-1337	StorageDiff should handle scaling nodes up and down
ENTMQST-1556	Exponential backoff in KafkaRoller sometimes takes 20+ minutes
ENTMQST-1557	Improve the logging in KafkaRoller class
ENTMQST-1691	KafkaConnectS2I doesn't do rolling update after resources edit
ENTMQST-1740	Automate ZooKeeper scaling
ENTMQST-1835	Propagate all labels from template
ENTMQST-1839	When KafkaConnect is scaled to 0 the reconciliation always timeouts
ENTMQST-1859	Increase Cluster Operator memory limits from install YAMLs
ENTMQST-1877	The CR of a topic returns the wrong status after decreasing partitions

Issue Number	Description
ENTMQST-1888	The CR of a topic returns the wrong status after changing the value of **min.insync.replicas**
ENTMQST-1889	Kafka Entity Operator has plain-text passwords in logs
ENTMQST-1922	KafkaRoller blocks rolling update when topic with RF=1 and MIN-ISR=2 exists
ENTMQST-1927	Cluster Operator fails to start on OCP
ENTMQST-1980	Fix parsing of forbidden configuration options and their exceptions
ENTMQST-2019	Fix the namespaces in RBAC files for installing the Cluster Operator to watch all namespaces
ENTMQST-2022	Kafka cluster is ending up with duplicate kafkaTopic after upgrade from 1.3 to 1.4

CHAPTER 6. KNOWN ISSUES

This section lists the known issues for AMQ Streams 1.5.

Issue Number

ENTMQST-2060 - Cruise Control default **hard.goals** still include unsupported goals

Description and workaround

If you create a **KafkaRebalance** custom resource containing one or more *supported* optimization goals in the **goals** field, Cruise Control returns the following error:

```
Missing hard goals [NetworkInboundCapacityGoal, DiskCapacityGoal, RackAwareGoal,
NetworkOutboundCapacityGoal, CpuCapacityGoal, ReplicaCapacityGoal] in the provided goals...
```

To workaround this error, do one of the following:

- Add **skipHardGoalCheck: true** to the **KafkaRebalance** custom resource:

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaRebalance
metadata:
  name: my-rebalance
  labels:
    strimzi.io/cluster: my-cluster
spec:
  goals:
    - NetworkInboundCapacityGoal
    - DiskCapacityGoal
    - RackAwareGoal
    - NetworkOutboundCapacityGoal
    - ReplicaCapacityGoal
  skipHardGoalCheck: true
```

- Specify the following hard goals in the the **cruiseControl** property in the **Kafka** resource:

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  cruiseControl:
  config:
    hard.goals: >
      com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
      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
```

CHAPTER 7. SUPPORTED INTEGRATION PRODUCTS

AMQ Streams 1.5 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 (as a Technology Preview)
- **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 2020-04 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 1.5 documentation.

CHAPTER 8. IMPORTANT LINKS

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

Revised on 2020-09-09 13:45:47 UTC