Red Hat AMQ 7.7

Release Notes for AMQ Streams 1.5 on RHEL

For use with AMQ Streams on Red Hat Enterprise Linux

Last Updated: 2020-07-02

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

For use with AMQ Streams on Red Hat Enterprise Linux

Legal Notice

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

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.

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

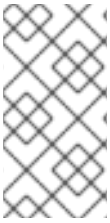
Kafka 2.4.x is supported in AMQ Streams 1.5 only for upgrade purposes.

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 2.5.0 requires ZooKeeper version 3.5.8. Therefore, the first step involved in upgrading to AMQ Streams 1.5 is to upgrade ZooKeeper to version 3.5.8, as described in AMQ Streams and Kafka upgrades.

Refer to the ZooKeeper 3.5.8 Release Notes for additional information. Note the changes to the configuration format introduced in ZooKeeper 3.5.7.



NOTE

If you are upgrading from a ZooKeeper version earlier than version 3.5.7, because of configuration changes introduced in that release, you will need to perform some additional upgrade steps. For more information, refer to the Release Notes for AMQ Streams 1.4 on RHEL and the steps for upgrading AMQ Streams 1.4 on RHEL .

1.3. 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 .

1.4. 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 Red Hat Enterprise Linux. 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.

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. 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

```
listener.name.client.oauthbearer.sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required \
  # ...
  oauth.check.issuer=false \ (1)
  oauth.fallback.username.claim="CLIENT-ID" \ (2)
  oauth.fallback.username.prefix="CLIENT-ACCOUNT" \ (3)
  oauth.valid.token.type="bearer" \ (4)
  oauth.userinfo.endpoint.uri="https://AUTH-SERVER-ADDRESS/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 **oauth.check.issuer** to **false** and do not specify a **oauth.valid.issuer.uri**. 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 **oauth.fallback.username.claim** 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 **oauth.fallback.username.claim**,

oauth.fallback.username.claim, and **oauth.fallback.username.prefix** settings are applied to the response of the **userinfo** endpoint.

Additional configuration options for Kafka components

```
# ...
System.setProperty(ClientConfig.OAUTH_SCOPE, "SCOPE-VALUE") 1
```

1 (Optional) 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 Kafka Java clients to use OAuth 2.0.

2.3. 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

```
http.enabled=true
http.host=0.0.0.0
http.port=8080
http.cors.enabled=true 1
http.cors.allowedOrigins=https://strimzi.io 2
http.cors.allowedMethods=GET,POST,PUT,DELETE,OPTIONS,PATCH 3
```

1 Set to **true** to enable CORS.

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

3 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 install Cruise Control and use it to rebalance a Kafka cluster. Cruise Control helps to reduce the time and effort involved in running an efficient and balanced Kafka cluster.

A zipped distribution of Cruise Control is available for download from the Customer Portal. To install Cruise Control, you configure each Kafka broker to use the provided Metrics Reporter. Then, you set Cruise Control properties, including optimization goals, and start Cruise Control using the provided script.

The Cruise Control server is hosted on a single machine for the whole Kafka cluster.

When Cruise Control is running, you can use the REST API to:

- Generate dry run optimization proposals from multiple optimization goals
- Initiate an optimization proposal to rebalance the Kafka cluster

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.

3.2. 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

There are no deprecated features for AMQ Streams 1.5.

CHAPTER 5. FIXED ISSUES

There are no fixed issues for AMQ Streams 1.5.

CHAPTER 6. KNOWN ISSUES

This section lists the known issues for AMQ Streams 1.5.

Issue Number

ENTMQST-2030 - kafka-ack reports **javax.management.InstanceAlreadyExistsException: kafka.admin.client:type=app-info,id=<client_id>with client.id set**

Description

If the **bin/kafka-acls.sh** utility is used in combination with the **--bootstrap-server** parameter to add or remove an ACL, the operation is successful but a warning is generated. The reason for the warning is that a second **AdminClient** instance is created. This will be fixed in a future release of Kafka.

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 (Technology Preview)
- **Red Hat Debezium 1.0 and later** for monitoring databases and creating event streams

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 2020-07-02 13:15:51 UTC