Red Hat AMQ 7.6

Release Notes for AMQ Streams 1.4 on OpenShift

For use with AMQ Streams on OpenShift Container Platform

Last Updated: 2020-04-20

Red Hat AMQ 7.6 Release Notes for AMQ Streams 1.4 on OpenShift

For use with AMQ Streams on OpenShift Container Platform

Abstract

These release notes contain the latest information about new features, enhancements, fixes, and issues contained in the AMQ Streams 1.4 release.

Table of Contents

CHAPTER 1. FEATURES 3
1.1. KAFKA 2.4.0 SUPPORT 3
1.1.1. ZooKeeper 3.5.7 3
1.2. KAFKACONNECTOR RESOURCES 3
1.2.1. Enabling KafkaConnectors 4
1.3. KAFKA LISTENER CERTIFICATES 5
1.4. OAUTH 2.0 AUTHENTICATION 5
1.5. DEBEZIUM FOR CHANGE DATA CAPTURE INTEGRATION 6

CHAPTER 2. ENHANCEMENTS 8
2.1. KAFKA 2.4.0 ENHANCEMENTS 8
2.2. KAFKA BRIDGE NOW SUPPORTS DISTRIBUTED TRACING 8
2.3. USER QUOTAS 8
2.4. PKCS #12 STORAGE 9
2.5. DOCKERFILE USER FOR KAFKA CONNECT BASE IMAGE 9

CHAPTER 3. TECHNOLOGY PREVIEWS 10
3.1. OAUTH 2.0 AUTHORIZATION 10
3.2. SERVICE REGISTRY 10
3.3. MIRRORMAKER 2.0 11
3.4. OPENSHIFT 4.X DISCONNECTED INSTALLATION 11

CHAPTER 4. DEPRECATED FEATURES 12

CHAPTER 5. FIXED ISSUES 13

CHAPTER 6. KNOWN ISSUES 14
6.1. SCALING ZOOKEEPER 3.5.7 UP OR DOWN 14

CHAPTER 7. SUPPORTED INTEGRATION PRODUCTS 18

CHAPTER 8. IMPORTANT LINKS 19

CHAPTER 1. FEATURES

AMQ Streams version 1.4 is based on Strimzi 0.17.x.

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

1.1. KAFKA 2.4.0 SUPPORT

AMQ Streams now supports Apache Kafka version 2.4.0.

AMQ Streams uses Kafka 2.4.0. Only Kafka distributions built by Red Hat are supported.

You must upgrade the Cluster Operator to AMQ Streams version 1.4 before you can upgrade brokers and client applications to Kafka 2.4.0. For upgrade instructions, see AMQ Streams and Kafka upgrades .

Refer to the Kafka 2.3.0 and Kafka 2.4.0 Release Notes for additional information.

NOTE

Kafka 2.3.x is supported in AMQ Streams 1.4 only for upgrade purposes.

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

Changes to the partition rebalance protocol in Kafka 2.4.0

Kafka 2.4.0 adds *incremental cooperative rebalancing* for consumers and Kafka Streams applications. This is an improved rebalance protocol for implementing *partition rebalances* according to a defined *rebalance strategy*. Using the new protocol, consumers keep their assigned partitions during a rebalance and only revoke them at the end of the process if required to achieve cluster balance. This reduces the unavailability of the consumer group or Kafka Streams application during a rebalance.

To take advantage of incremental cooperative rebalancing, you must upgrade consumers and Kafka Streams applications to use the new protocol instead of the old *eager rebalance protocol*.

See Upgrading consumers and Kafka Streams applications to cooperative rebalancing and Notable changes in 2.4.0 in the Apache Kafka documentation.

1.1.1. ZooKeeper 3.5.7

Kafka version 2.4.0 requires a new version of ZooKeeper, version 3.5.7.

You do **not** need to manually upgrade to ZooKeeper 3.5.7; the Cluster Operator performs the ZooKeeper upgrade when it upgrades Kafka brokers. However, you might notice some additional rolling updates during this procedure.

There is a known issue in AMQ Streams 1.4 related to scaling ZooKeeper. For more information, see Chapter 6, *Known issues*.

1.2. KAFKACONNECTOR RESOURCES

AMQ Streams now provides Kubernetes-native management of connectors in a Kafka Connect cluster using a new custom resource, named **KafkaConnector**, and an internal operator.

A **KafkaConnector** YAML file describes the configuration of a source or sink connector that you deploy to your Kubernetes cluster to either create a new connector instance or manage a running one. Like other Kafka resources, the Cluster Operator updates running connector instances to match the configurations defined in their **KafkaConnectors**.

The Installation and Example Files now include an example **KafkaConnector** resource in **examples/connector/source-connector.yaml**. Deploy the example YAML file to create a **FileStreamSourceConnector** that sends each line of the license file to Kafka as a message in a topic named **my-topic**.

Example KafkaConnector

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: KafkaConnector
metadata:
  name: my-source-connector
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
  class: org.apache.kafka.connect.file.FileStreamSourceConnector
  tasksMax: 2
  Config:
    file: "/opt/kafka/LICENSE"
    topic: my-topic
    # ...
```

1.2.1. Enabling KafkaConnectors

To ensure compatibility with earlier versions of AMQ Streams, **KafkaConnectors** are disabled by default. They might become the default way to create and manage connectors in future AMQ Streams releases.

To enable **KafkaConnectors** for an AMQ Streams 1.4 Kafka Connect cluster, add the **strimzi.io/use-connector-resources** annotation to the **KafkaConnect** resource. For example:

Example Kafka Connect cluster with KafkaConnectors enabled

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  # ...
```

If **KafkaConnectors** are enabled, manual changes made directly using the Kafka Connect REST API are reverted by the Cluster Operator.

NOTE

The Kafka Connect REST API (on port 8083) is still required to restart failed tasks.

See Creating and managing connectors, Deploying a **KafkaConnector** resource to Kafka Connect , and Enabling **KafkaConnector** resources.

1.3. KAFKA LISTENER CERTIFICATES

You can now provide your own server certificates and private keys for the following types of listeners:

- TLS listeners
- External listeners with TLS encryption enabled

These user-provided certificates are called *Kafka listener certificates*.

You can use your organization’s private Certificate Authority (CA) or a public CA to generate and sign your own Kafka listener certificates.

Listener configuration

You configure Kafka listener certificates in the **configuration.brokerCertChainAndKey** property of the listener. For example:

```
# ...
listeners:
  plain: {}
  external:
    type: loadbalancer
    configuration:
      brokerCertChainAndKey:
        secretName: my-secret
        certificate: my-listener-certificate.crt
        key: my-listener-key.key
    tls: true
    authentication:
      type: tls
# ...
```

See Kafka listener certificates and Providing your own Kafka listener certificates .

1.4. OAUTH 2.0 AUTHENTICATION

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

AMQ Streams supports the use of OAuth 2.0 authentication using the *SASL OAUTHBEARER* mechanism. Using OAuth 2.0 token based authentication, application clients can access resources on application servers (called ‘resource servers’) without exposing account credentials. The client presents an access token as a means of authenticating, which application servers can also use to find more information about the level of access granted. The authorization server handles the granting of access and inquiries about access.

In the context of AMQ Streams:

- Kafka brokers act as resource servers
- Kafka clients act as resource clients

The brokers and clients communicate with the OAuth 2.0 authorization server, as necessary, to obtain or validate access tokens.

For a deployment of AMQ Streams, OAuth 2.0 integration provides:

- Server-side OAuth 2.0 support for Kafka brokers
- Client-side OAuth 2.0 support for Kafka MirrorMaker, Kafka Connect and the Kafka Bridge

Red Hat Single Sign-On integration

You can deploy Red Hat Single Sign-On as an authorization server and configure it for integration with AMQ Streams.

You can use Red Hat Single Sign-On to:

- Configure authentication for Kafka brokers
- Configure and authorize clients
- Configure users and roles
- Obtain access and refresh tokens

See Using OAuth 2.0 token-based authentication .

1.5. DEBEZIUM FOR CHANGE DATA CAPTURE INTEGRATION

NOTE

Debezium for Change Data Capture is only supported on OpenShift 4.x.

Debezium for Change Data Capture is a distributed platform that monitors databases and creates change event streams. Debezium is built on Apache Kafka and can be deployed and integrated with AMQ Streams. Following a deployment of AMQ Streams, you deploy Debezium as a connector configuration through Kafka Connect. Debezium captures row-level changes to a database table and passes corresponding change events 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.

CHAPTER 2. ENHANCEMENTS

The enhancements added in this release are outlined below.

2.1. KAFKA 2.4.0 ENHANCEMENTS

For an overview of the enhancements introduced with Kafka 2.4.0, refer to the Kafka 2.4.0 Release Notes.

2.2. KAFKA BRIDGE NOW SUPPORTS DISTRIBUTED TRACING

Distributed tracing using Jaeger is now supported for the Kafka Bridge component of AMQ Streams on OpenShift.

The Kafka Bridge generates traces when it sends and receives messages to and from HTTP clients, and when HTTP clients send requests to the Kafka Bridge REST API to create a consumer, retrieve messages, and so on. You can view these traces in the Jaeger user interface.

To enable tracing for the Kafka Bridge, configure the **KafkaBridge** custom resource for Jaeger tracing. For example:

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  #...
  template:
    bridgeContainer:
      env:
        - name: JAEGER_SERVICE_NAME
          value: my-jaeger-service
        - name: JAEGER_AGENT_HOST
          value: jaeger-agent-name
        - name: JAEGER_AGENT_PORT
          value: "6831"
  tracing:
    type: jaeger
#...
```

Use **kubectl apply** to update the resource in your Kafka cluster. When the resource is updated, a Jaeger tracer based on your configuration is initialized by the Kafka Bridge.

See Distributed tracing and Enabling tracing in Mirror Maker, Kafka Connect, and Kafka Bridge resources.

2.3. USER QUOTAS

User quotas prevent users from exceeding a defined level of access to Kafka brokers. You can now set two types of user quotas on the **KafkaUser** resource:

- Network usage quotas, based on a byte threshold
- CPU utilization quotas, based on a time limit of CPU utilization

To set a user quota, edit the **KafkaUser.spec.quotas** property of the **KafkaUser** resource.

See Kafka User resource, **KafkaUser** schema reference, and Quotas in the Apache Kafka documentation.

2.4. PKCS #12 STORAGE

AMQ Streams uses *Secrets* to store private keys and certificates for Kafka cluster components and clients. Secrets are used for establishing TLS encrypted connections between Kafka brokers, and between brokers and clients. They are also used for mutual TLS authentication.

PKCS #12 defines an archive file format (**.p12**) for storing cryptography objects into a single file with password protection. You can now use PKCS #12 to manage certificates and keys in one place.

See PKCS #12 storage.

2.5. DOCKERFILE USER FOR KAFKA CONNECT BASE IMAGE

The **USER** specified in the Dockerfile when creating a Docker image from the Kafka Connect base image has changed.

AMQ Streams version	Value of **USER** instruction in Dockerfile
1.3	**USER jboss:jboss**
1.4	**USER 1001**

See Creating a Docker image from the Kafka Connect base image .

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. 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 Keycloak to configure authorization rules to constrain client access to Kafka brokers.

Red Hat Single Sign-On 7.3 does not support this Technoloy Preview of OAuth 2.0 token-based authorization. If you wish to try this feature, it is tested for use in a development environment with Keycloak 8.0.2 as the authorization server.

AMQ Streams supports the use of OAuth 2.0 token-based authorization through Keycloak Authorization Services, which allows you to manage security policies and permissions centrally.

Security policies and permissions defined in Keycloak 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 .

3.2. SERVICE REGISTRY

NOTE

This is a Technology Preview feature.

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.

3.3. MIRRORMAKER 2.0

NOTE

This is a Technology Preview feature.

You can now use MirrorMaker 2.0 with AMQ Streams.

MirrorMaker 2.0 is based on the Kafka Connect framework, *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.

NOTE

For this Technology Preview, all connectors are currently restarted for every reconcile of the MirrorMaker 2.0 operator. This does not affect the functionality, but it does affect the performance.

See Using AMQ Streams with MirrorMaker 2.0 .

3.4. OPENSHIFT 4.X DISCONNECTED INSTALLATION

NOTE

This is a Technology Preview feature.

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 .

CHAPTER 4. DEPRECATED FEATURES

There are no deprecated features for AMQ Streams 1.4.

CHAPTER 5. FIXED ISSUES

The following table lists the issues fixed in AMQ Streams 1.4.

Issue Number	Description
ENTMQST-1106	Adding/changing **overrides.bootstrap.address** should trigger a rolling update
ENTMQST-1153	Scaling Kafka logs and exception and doesn't work properly
ENTMQST-1402	Incorrect URL in KafkaBridge status
ENTMQST-1411	TLS-sidecar can terminate earlier than Kafka container itself
ENTMQST-1481	Use **admin.** configuration prefix in Kafka Connect configuration
ENTMQST-1530	*[RFE]* Add functionality to convert units
ENTMQST-1531	**-Xmx** for pods would not be determined from memory limit but memory request
ENTMQST-1536	Dashboards for Zookeeper and Kafka use wrong expression for memory and JVM memory
ENTMQST-1551	CA renewal breaks the cluster when interrupted in the middle
ENTMQST-1563	Alert manager configuration file has a wrong file name
ENTMQST-1652	**COORDINATOR_NOT_AVAILABLE** with single node cluster
ENTMQST-1717	Fix build of ZSTD library

CHAPTER 6. KNOWN ISSUES

This section contains the known issues for AMQ Streams 1.4 on OpenShift.

6.1. SCALING ZOOKEEPER 3.5.7 UP OR DOWN

There is a known issue related to scaling ZooKeeper up or down. *Scaling ZooKeeper up* means adding servers to a ZooKeeper cluster. *Scaling ZooKeeper down* means removing servers from a ZooKeeper cluster.

Kafka 2.4.0 requires ZooKeeper 3.5.7.

The configuration procedure for ZooKeeper 3.5.7 servers is significantly different than for ZooKeeper 3.4.x servers. Referred to as dynamic reconfiguration, the new configuration procedure requires that servers are added or removed using the ZooKeeper CLI or Admin API. This ensures that a stable ZooKeeper cluster is maintained during the scale up or scale down process.

To scale a ZooKeeper 3.5.7 cluster up or down, you must perform the procedures described in this known issue.

NOTE

In future AMQ Streams releases, ZooKeeper scale up and scale down will be handled by the Cluster Operator.

Scaling up ZooKeeper 3.5.7 servers in an AMQ Streams 1.4 cluster

This procedure assumes that:

- AMQ Streams is running in the **namespace** namespace and the Kafka cluster is named **my-cluster**.
- A 3 node ZooKeeper cluster is running.

Perform the following steps for each ZooKeeper server, **one at a time**:

1. Edit the **spec.zookeeper.replicas** property in the **Kafka** custom resource. Set the replica count to **4** (**n=4**).

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
  # ...
  zookeeper:
    replicas: 4
    storage:
      type: persistent-claim
      size: 100Gi
      deleteClaim: false
  # ...
```

2. Allow the ZooKeeper server (**zookeeper-3**) to start up normally and establish a link to the existing quorum.
   You can check this by running:

   ```
   kubectl exec -n <namespace> -it <my-cluster>-zookeeper-3 -c zookeeper -- bash -c "echo
   'srvr' | nc 127.0.0.1 21813 | grep 'Mode:'"
   ```

   The output of this command should be similar to: **Mode: follower**.

   > **NOTE**
   >
   > The index number in the name of the new ZooKeeper node, **zookeeper-x**, matches the final number of the client port in the **nc 127.0.0.1 2181x** command.

3. Open a **zookeeper-shell** session on one of the nodes in the original cluster (nodes 0, 1, or 2):

   ```
   kubectl exec -n <namespace> -it <my-cluster>-zookeeper-0 -c zookeeper -- ./bin/zookeeper-
   shell.sh localhost:21810
   ```

4. In the shell session, enter the following line to add the new server to the quorum as a voting member:

   ```
   reconfig -add server.4=127.0.0.1:28883:38883:participant;127.0.0.1:21813
   ```

   > **NOTE**
   >
   > Within the ZooKeeper cluster, nodes are indexed from one, not zero as in the node names. So, the new **zookeeper-3** node is referred to as **server.4** within the ZooKeeper configuration.

   This outputs the new cluster configuration:

   ```
   server.1=127.0.0.1:28880:38880:participant;127.0.0.1:21810
   server.2=127.0.0.1:28881:38881:participant;127.0.0.1:21811
   server.3=127.0.0.1:28882:38882:participant;127.0.0.1:21812
   server.4=127.0.0.1:28883:38883:participant;127.0.0.1:21813
   version=100000054
   ```

   The new configuration propagates to the other servers in the ZooKeeper cluster; the new server is now a full member of the quorum.

5. In **spec.zookeeper.replicas** in the **Kafka** custom resource, increase the replica count by one (**n=5**).

6. Allow the ZooKeeper server (**zookeeper-<n-1>**) to start up normally and establish a link to the existing quorum. You can check this by running:

   ```
   kubectl exec -n <namespace> -it <my-cluster>-zookeeper-<n-1> -c zookeeper -- bash -c
   "echo 'srvr' | nc 127.0.0.1 2181<n-1> | grep 'Mode:'"
   ```

   The output of the command should be similar to: **Mode: follower**.

7. Open a **zookeeper-shell** session on one of the nodes in the original cluster (in this example, nodes 0 >= i <= n-2):

```
kubectl exec -n <namespace> -it <my-cluster>-zookeeper-<i> -c zookeeper --
./bin/zookeeper-shell.sh localhost:2181<i>
```

8. In the shell session, enter the following line to add the new ZooKeeper server to the quorum as a voting member:

```
reconfig -add server.<n>=127.0.0.1:2888<n-1>:3888<n-1>:participant;127.0.0.1:2181<n-1>
```

9. Repeat steps 5-8 for every server you want to add.

10. When you have a cluster of the desired size, you need to signal to the Cluster Operator that it is safe to roll the ZooKeeper cluster again. To do so, set the **manual-zk-scaling** annotation to **false** in the **Kafka** custom resource. The Cluster Operator automatically sets this to **true** when you change the number of ZooKeeper replicas.

```
kubectl -n <namespace> annotate statefulset <my-cluster>-zookeeper strimzi.io/manual-zk-
scaling=false --overwrite
```

Scaling down ZooKeeper 3.5.7 servers in an AMQ Streams 1.4 cluster

This procedure assumes that AMQ Streams is running in the **namespace** namespace and the Kafka cluster is named **my-cluster**.

When removing ZooKeeper nodes, the highest numbered server will be deleted first and so on in descending order. Therefore, if you have a 5 node cluster and want to scale down to 3, you would remove **zookeeper-4** and **zookeeper-3** and keep **zookeeper-0**, **zookeeper-1**, and **zookeeper-2**.

NOTE

Before proceeding, read the notes on "Removing servers" in the ZooKeeper documentation.

Perform the following steps for each ZooKeeper server, **one at a time**:

1. Log in to the **zookeeper-shell** on one of the nodes that will be **retained** after the scale down:

```
kubectl exec -n <namespace> -it <my-cluster>-zookeeper-0 -c zookeeper -- ./bin/zookeeper-
shell.sh localhost:21810
```

NOTE

The index number in the ZooKeeper node’s name, **zookeeper-x**, matches the final number of the client port in the **zookeeper-shell.sh localhost:2181x** command.

2. Output the existing cluster configuration using the **config** command:

```
config
```

Assuming you are scaling down from a cluster that had 5 ZooKeeper nodes, the output of the command should be similar to:

```
server.1=127.0.0.1:28880:38880:participant;127.0.0.1:21810
server.2=127.0.0.1:28881:38881:participant;127.0.0.1:21811
server.3=127.0.0.1:28882:38882:participant;127.0.0.1:21812
server.4=127.0.0.1:28883:38883:participant;127.0.0.1:21813
server.5=127.0.0.1:28884:38884:participant;127.0.0.1:21814
version=100000057
```

3. Next, remove the highest numbered server first, which in this case is **server.5**:

```
reconfig -remove 5
```

This outputs the new configuration that will propagate to all other members of the quorum:

```
server.1=127.0.0.1:28880:38880:participant;127.0.0.1:21810
server.2=127.0.0.1:28881:38881:participant;127.0.0.1:21811
server.3=127.0.0.1:28882:38882:participant;127.0.0.1:21812
server.4=127.0.0.1:28883:38883:participant;127.0.0.1:21813
version=200000012
```

4. When the propagation is complete, the number of replicas for the **zookeeper** section of the **Kafka** resource can be reduced by one. This will shut down **zookeeper-4** (**server.5**).

5. Repeat steps 1-4 to incrementally reduce the cluster size. **Remember to remove the servers in descending order**.

6. When you have a cluster of the desired size, you need to signal to the Cluster Operator that it is safe to roll the ZooKeeper cluster again. To do so, set the **manual-zk-scaling** annotation to **false** in the **Kafka** custom resource. The Cluster Operator automatically sets this to **true** when you change the number of ZooKeeper replicas.

```
kubectl -n <namespace> annotate statefulset <my-cluster>-zookeeper strimzi.io/manual-zk-scaling=false --overwrite
```

NOTE

It is possible to specify multiple servers to be removed at once; for example, you could enter **reconfig -remove 4,5** to remove the two highest numbered servers at once and scale down from 5 to 3 in one step. However, this can lead to instability and is **NOT** recommended.

CHAPTER 7. SUPPORTED INTEGRATION PRODUCTS

AMQ Streams 1.4 supports integration with the following Red Hat products.

- **Red Hat Single Sign-On 7.3** for OAuth 2.0 authentication (and OAuth 2.0 authorization with Keycloak as a Technology Preview)
- **Red Hat 3scale API Management 2.6** 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 2019-12 and later (Technology Preview)** 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 1.4 documentation.

CHAPTER 8. IMPORTANT LINKS

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

Revised on 2020-04-20 14:36:38 UTC