# Red Hat Streams for Apache Kafka 2.7

## Using the Streams for Apache Kafka Console

The Streams for Apache Kafka console supports your deployment of Streams for Apache Kafka.

Last Updated: 2024-05-30

# Red Hat Streams for Apache Kafka 2.7 Using the Streams for Apache Kafka Console

The Streams for Apache Kafka console supports your deployment of Streams for Apache Kafka.

## Abstract

Connect the console to a Kafka cluster that’s managed by Streams for Apache Kafka and use it to monitor and manage the cluster.

# Table of Contents

# PREFACE

# PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation.

To propose improvements, open a Jira issue and describe your suggested changes. Provide as much detail as possible to enable us to address your request quickly.

## Prerequisite

- You have a Red Hat Customer Portal account. This account enables you to log in to the Red Hat Jira Software instance.
  If you do not have an account, you will be prompted to create one.

## Procedure

1. Click the following: Create issue.
2. In the **Summary** text box, enter a brief description of the issue.
3. In the **Description** text box, provide the following information:
   - The URL of the page where you found the issue.
   - A detailed description of the issue.
     You can leave the information in any other fields at their default values.
4. Add a reporter name.
5. Click **Create** to submit the Jira issue to the documentation team.

Thank you for taking the time to provide feedback.

# TECHNOLOGY PREVIEW

The Streams for Apache Kafka Console is a technology preview.

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 the support scope, see Technology Preview Features Support Scope.

# CHAPTER 1. ADMINISTERING KAFKA CLUSTERS WITH STREAMS FOR APACHE KAFKA CONSOLE

The Streams for Apache Kafka Console provides a user interface to facilitate the administration of Kafka clusters, delivering real-time insights for monitoring, managing, and optimizing each cluster from its user interface.

# CHAPTER 2. CONNECTING THE STREAMS FOR APACHE KAFKA CONSOLE TO A KAFKA CLUSTER

Deploy the Streams for Apache Kafka Console to the same OpenShift cluster as the Kafka cluster managed by Streams for Apache Kafka. Use the installation files provided with the Streams for Apache Kafka Console.

For each Kafka cluster, the configuration of the **Kafka** resource used to install the cluster requires the following:

- Sufficient authorization for the console to connect to the cluster.
- A **route** listener that the console can use to connect to the cluster.
- Prometheus enabled and able to scrape metrics from the cluster.
- Metrics configuration (through a **ConfigMap**) for exporting metrics in a format suitable for Prometheus.

The Streams for Apache Kafka Console requires a Kafka user, configured as **KafkaUser** custom resource, for the console to access the cluster as an authenticated and authorized user.

When you configure the **KafkaUser** authentication and authorization mechanisms, ensure they match the equivalent **Kafka** configuration.

- **KafkaUser.spec.authentication** matches **Kafka.spec.kafka.listeners[*].authentication**
- **KafkaUser.spec.authorization** matches **Kafka.spec.kafka.authorization**

NOTE

Prometheus must be installed and configured to scrape metrics from Kubernetes and Kafka clusters and populate the metrics graphs in the console.

**Prerequisites**

- Installation requires an OpenShift user with **cluster-admin** role, such as **system:admin**.
- An OpenShift 4.12 to 4.15 cluster.
- A Kafka cluster managed by Streams for Apache Kafka running on the OpenShift cluster.
- The Prometheus Operator, which must be a separate operator from the one deployed for OpenShift monitoring.
- The **oc** command-line tool is installed and configured to connect to the OpenShift cluster.
- Secret values for session management and authentication within the console.
  You can use the OpenSSL TLS management tool for generating the values as follows:

```
SESSION_SECRET=$(LC_CTYPE=C openssl rand -base64 32)
echo "Generated SESSION_SECRET: $SESSION_SECRET"

NEXTAUTH_SECRET=$(LC_CTYPE=C openssl rand -base64 32)
echo "Generated NEXTAUTH_SECRET: $NEXTAUTH_SECRET"
```

Use **openssl help** for command-line descriptions of the options used.

In addition to the files to install the Streams for Apache Kafka Console, Streams for Apache Kafka Console also provides pre-configured files to install the Streams for Apache Kafka Operator, the Prometheus Operator, and a Kafka cluster. In this procedure, we assume the operators are installed.

The installation files offer the quickest way to set up and try the console, though you can use your own deployments of Streams for Apache Kafka and Prometheus.

**Procedure**

1. Download and extract the Streams for Apache Kafka Console installation artifacts.
   The console is available from Streams for Apache Kafka software downloads page .

   The files contain the deployment configuration required for the console, the Kafka cluster, and Prometheus.

2. Create a Prometheus instance with the configuration required by the console by applying the Prometheus installation files:

   a. Edit **${NAMESPACE}** in the **console-prometheus-server.clusterrolebinding.yaml** file to use the namespace the Prometheus instance is going to be installed into:

   ```
   sed -i 's/${NAMESPACE}/'"my-project"'/g' <resource_path>/console-prometheus-server.clusterrolebinding.yaml
   ```

   For example, in this procedure we are installing to the **my-project** namespace. The configuration binds the role for Prometheus with its service account.

   b. Create the Prometheus instance by applying the installation files in this order:

   ```
   # Prometheus security resources
   oc apply -n my-project -f <resource_path>/prometheus/console-prometheus-server.clusterrole.yaml
   oc apply -n my-project -f <resource_path>/prometheus/console-prometheus-server.serviceaccount.yaml
   oc apply -n my-project -f <resource_path>/prometheus/console-prometheus-server.clusterrolebinding.yaml

   # Prometheus PodMonitor and Kubernetes scrape configurations
   oc apply -n my-project -f <resource_path>/prometheus/kafka-resources.podmonitor.yaml
   oc apply -n my-project -f <resource_path>/prometheus/kubernetes-scrape-configs.secret.yaml

   # Prometheus instance
   oc apply -n my-project -f <resource_path>/prometheus/console-prometheus.prometheus.yaml
   ```

   The instance is named **console-prometheus** and the URL of the service for connecting the console is **http://prometheus-operated.my-project.svc.cluster.local:9090**, with **my-project** taken from the namespace name.

NOTE

No route is deployed for the **console-prometheus** instance as it does not need to be accessible from outside the OpenShift cluster.

3. Create and deploy a Kafka cluster.

   a. If you are using the console with a Kafka cluster operating in KRaft mode, update the metrics configuration for the cluster in the **console-kafka-metrics.configmap.yaml** file:

      - Uncomment the KRaft-related metrics configuration.
      - Comment out the ZooKeeper related metrics.

      This file contains the metrics configuration required by the console.

   b. Edit the **KafkaUser** custom resource in the **console-kafka-user1.kafkauser.yaml** file by adding ACL types to provide authorized access for the console to the Kafka cluster.
      At a minimum, the Kafka user requires the following ACL rules:

      - **Describe**, **DescribeConfigs** permissions for the **cluster** resource
      - **Read**, **Describe**, **DescribeConfigs** permissions for all **topic** resources
      - **Read**, **Describe** permissions for all **group** resources

      #### Example user authorization settings

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: console-cluster-user1
  labels:
    strimzi.io/cluster: console-kafka
spec:
  authentication:
    type: scram-sha-512
  authorization:
    type: simple
    acls:
      - resource:
          type: cluster
          name: ""
          patternType: literal
        operations:
          - Describe
          - DescribeConfigs
      - resource:
          type: topic
          name: "*"
          patternType: literal
        operations:
          - Read
          - Describe
          - DescribeConfigs
      - resource:
```

```
      type: group
      name: "*"
      patternType: literal
    operations:
      - Read
      - Describe
```

c. Edit the **console-kafka.kafka.yaml** file to replace the placeholders:

```
sed -i 's/type: ${LISTENER_TYPE}/type: route/g' console-kafka.kafka.yaml
sed -i 's/\${CLUSTER_DOMAIN}/'"<my_router_base_domain>"'/g' console-
kafka.kafka.yaml
```

This file contains the **Kafka** custom resource configuration to create the Kafka cluster.

These commands do the following:

- Replace **type: ${LISTENER_TYPE}** with **type: route**.
- Replace **${CLUSTER_DOMAIN}** with the value of the base domain required to specify the route listener hosts used by the bootstrap and per-broker services. By default, **route** listener hosts are automatically assigned by OpenShift. However, you can override the assigned route hosts by specifying hosts.

Alternatively, you can copy the example configuration to your own Kafka deployment .

d. Create the Kafka cluster by applying the installation files in this order:

```
# Metrics configuration
oc apply -n my-project -f console-kafka-metrics.configmap.yaml

# Create the cluster
oc apply -n my-project -f console-kafka.kafka.yaml

# Create a user for the cluster
oc apply -n my-project -f <resource_path>/console-kafka-user1.kafkauser.yaml
```

If you are using your own Kafka cluster, apply the updated **Kafka** resource configuration instead of **console-kafka.kafka.yaml**.

The installation files create a Kafka cluster as well as a Kafka user and the metrics configuration required by the console for connecting to the cluster A Kafka user and metrics configuration are required for each Kafka cluster you want to monitor through the console. Each Kafka user requires a unique name.

4. Check the status of the deployment:

```
oc get pods -n <my_console_namespace>
```

**Output shows the operators and cluster readiness**

```
NAME                     READY   STATUS   RESTARTS
strimzi-cluster-operator   1/1     Running  0
console-kafka-kafka-0      1/1     Running  0
console-kafka-kafka-1      1/1     Running  0
```

```
console-kafka-kafka-2       1/1     Running  0
prometheus-operator-...     1/1     Running  0
console-prometheus          1/1     Running  0
```

Here, **console-kafka** is the name of the cluster.

A pod ID identifies the pods created.

With the default deployment, you install three pods.

READY shows the number of replicas that are ready/expected. The deployment is successful when the STATUS displays as Running.

5. Install the Streams for Apache Kafka Console.

   a. Edit the **console-server.clusterrolebinding.yaml** file to use the namespace the console instance is going to be installed into:

   ```
   sed -i 's/${NAMESPACE}/'"my-project"'/g' /<resource_path>console-
   server.clusterrolebinding.yaml
   ```

   The configuration binds the role for the console with its service account.

   b. Install the console user interface and route to the interface by applying the installation files in this order:

   ```
   # Console security resources
   oc apply -n my-project -f <resource_path>/console-server.clusterrole.yaml
   oc apply -n my-project -f <resource_path>/console-server.serviceaccount.yaml
   oc apply -n my-project -f <resource_path>/console-server.clusterrolebinding.yaml

   # Console user interface service
   oc apply -n my-project -f <resource_path>/console-ui.service.yaml

   # Console route
   oc apply -n my-project -f <resource_path>/console-ui.route.yaml
   ```

   The install creates the role, role binding, service account, services, and route necessary to run the console user interface.

   c. Create a **Secret** called **console-ui-secrets** containing two secret values (as described in the prerequisites) for session management and authentication within the console:

   ```
   oc create secret generic console-ui-secrets -n my-project \
     --from-literal=SESSION_SECRET="<session_secret_value>" \
     --from-literal=NEXTAUTH_SECRET="<next_secret_value>"
   ```

   The secrets are mounted as environment variables when the console is deployed.

   d. Get the hostname for the route created for the console user interface:

   ```
   oc get route console-ui-route -n my-project -o jsonpath='{.spec.host}'
   ```

   The hostname is required for access to the console user interface.

   e. Edit the **console.deployment.yaml** file to replace the placeholders:

```
sed -i 's/${CONSOLE_HOSTNAME}/'"<route_hostname>"'/g' console.deployment.yaml
sed -i 's/${NAMESPACE}/'"my-project"'/g' console.deployment.yaml
```

These commands do the following:

- Replace **https://${CONSOLE_HOSTNAME}** with **https://<route_hostname>**, which is the route used to access the console user interface.
- Replace **${NAMESPACE}** with the **my-project** namespace name in **http://prometheus-operated.${NAMESPACE}.svc.cluster.local:9090**, which is the URL of the Prometheus instance used by the console.

If you are using your own Kafka cluster, ensure that the environment variables are configured with the correct values. The values enable the console to connect with the cluster and retrieve metrics.

f. Install the console API:

```
oc apply -n my-project -f <resource_path>/console.deployment.yaml
```

**Output shows the console readiness**

```
NAME                    READY   STATUS   RESTARTS
strimzi-cluster-operator    1/1     Running  0
console-kafka-kafka-0       1/1     Running  0
console-kafka-kafka-0       1/1     Running  0
console-kafka-kafka-0       1/1     Running  0
prometheus-operator-...     1/1     Running  0
console-prometheus          1/1     Running  0
console-api             1/1     Running  0
console-ui              1/1     Running  0
```

### Adding the example configuration to your own Kafka cluster

If you already have a Kafka cluster installed, you can update the **Kafka** resource with the required configuration. When applying the cluster configuration files, use the updated **Kafka** resource rather than using the **Kafka** resource provided with the Streams for Apache Kafka Console installation files.

The **Kafka** resource requires the following configuration:

- A **route** listener to expose the cluster for console connection
- Prometheus metrics enabled for retrieving metrics on the cluster. Add the same configuration for ZooKeeper if you are using ZooKeeper for metadata management.

The Prometheus metrics configuration must reference the **ConfigMap** that provides the metrics configuration required by the console. The metrics configuration is provided in the **console-cluster-metrics.configmap.yaml** resource configuration file.

### Example Kafka cluster configuration for console connection

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: console-kafka
  namespace: my-project
spec:
  entityOperator:
    topicOperator: {}
    userOperator: {}
  kafka:
    authorization:
      type: simple
    config:
      allow.everyone.if.no.acl.found: 'true'
      default.replication.factor: 3
      inter.broker.protocol.version: '3.6'
      min.insync.replicas: 2
      offsets.topic.replication.factor: 3
      transaction.state.log.min.isr: 2
      transaction.state.log.replication.factor: 3
    listeners: (1)
      - name: route1
        port: 9094
        tls: true
        type: route
        authentication:
          type: scram-sha-512
    replicas: 3
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 10Gi
        deleteClaim: false
    metricsConfig: (2)
      type: jmxPrometheusExporter
      valueFrom:
        configMapKeyRef:
          name: console-cluster-metrics
          key: kafka-metrics-config.yml
    version: 3.6.0
  zookeeper:
    replicas: 3
    storage:
      deleteClaim: false
      size: 10Gi
      type: persistent-claim
    metricsConfig: (3)
      type: jmxPrometheusExporter
      valueFrom:
        configMapKeyRef:
          name: console-cluster-metrics
          key: zookeeper-metrics-config.yml
```

(1) Listener to expose the cluster for console connection. In this example, a route listener is configured.

(2) Prometheus metrics, which are enabled by referencing a **ConfigMap** containing configuration for the Prometheus JMX exporter.

3 Add ZooKeeper configuration only if you are using Streams for Apache Kafka with ZooKeeper for cluster management. It is not required in KRaft mode.

### Checking the console deployment environment variables

If you are using your own Kafka cluster, check the deployment configuration for the console has the required environment variables.

The following prefixes determine the scope of the environment variable values:

- **KAFKA** represents configuration for all Kafka clusters.
- **CONSOLE_KAFKA_<UNIQUE_NAME_ID_FOR_CLUSTER>** represents configuration for each specific cluster.

### Example console deployment configuration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: console
spec:
  replicas: 1
  # ...
  template:
    metadata:
      labels:
        app: console
    spec:
      # ...
      containers:
      - name: console-api
        # ...
        env:
        - name: KAFKA_SECURITY_PROTOCOL (1)
          value: SASL_SSL
        - name: KAFKA_SASL_MECHANISM (2)
          value: SCRAM-SHA-512
        - name: CONSOLE_KAFKA_CLUSTER1 (3)
          value: my-project/console-kafka
        - name: CONSOLE_KAFKA_CLUSTER1_BOOTSTRAP_SERVERS (4)
          value: console-kafka-route1-bootstrap-my-project.router.com:443
        - name: CONSOLE_KAFKA_CLUSTER1_SASL_JAAS_CONFIG (5)
          valueFrom:
            secretKeyRef:
              name: console-cluster-user1
              key: sasl.jaas.config
      - name: console-ui
        # ...
        env:
        - name: NEXTAUTH_SECRET (6)
          valueFrom:
            secretKeyRef:
              name: console-ui-secrets
              key: NEXTAUTH_SECRET
      - name: SESSION_SECRET (7)
        valueFrom:
          secretKeyRef:
            name: console-ui-secrets
            key: SESSION_SECRET
      - name: NEXTAUTH_URL (8)
        value: 'https://console-ui-route-my-project.router.com'
      - name: BACKEND_URL  (9)
        value: 'http://127.0.0.1:8080'
      - name: CONSOLE_METRICS_PROMETHEUS_URL (10)
        value: 'http://prometheus-operated.my-project.svc.cluster.local:9090'
```

1. The security protocol used for communication with Kafka brokers.
2. The SASL mechanism for console (client) authentication to the Kafka brokers.
3. The namespace and the name specified for the cluster in its **Kafka** resource configuration.
4. The host and port pair of the bootstrap broker address to discover and connect to all brokers in the Kafka cluster. In this example, a route listener address is being used. The listener was configured in the **Kafka** resource.
5. Authentication credentials for the Kafka user mounted as a **Secret** to the Streams for Apache Kafka Console deployment. The **sasl.jaas.config** property within the secret contains the SASL credentials, such as usernames and passwords. Here, the credentials are mounted as a **Secret**.
6. Secret for authentication within the console.
7. Secret for session management within the console
8. The URL to connect to the Streams for Apache Kafka user interface and for users to access the console.
9. The backend server that the console user interface communicates with for data retrieval.
10. The URL to connect to the Prometheus instance, which includes the namespace (**my-project**) of the **Kafka** resource.

# CHAPTER 3. NAVIGATING THE STREAMS FOR APACHE KAFKA CONSOLE

When you open the Streams for Apache Kafka Console, the homepage presents a list of connected Kafka clusters. By clicking on a Kafka cluster name on the homepage or from the side menu, you can find information on the following components:

**Kafka clusters**

A group of Kafka brokers and management components.

**Brokers**

A broker contains topics and orchestrates the storage and passing of messages.

**Topics**

A topic provides a destination for the storage of data. Kafka splits each topic into one or more partitions.

**Partitions**

A subset of a topic used for data sharding and replication. The number of partitions is defined in the topic configuration.

**Consumer groups**

Kafka groups consumers with the same group ID and distributes messages across group members. Consumers within a group receive data from one or more partitions.

For example, you can view the status of a Kafka cluster before navigating to view information on the cluster’s brokers and topics, or the consumer groups connected to the Kafka cluster.

NOTE

If the side menu is not visible, click the hamburger menu (three horizontal lines) in the console header.

# CHAPTER 4. HOME: CHECKING CONNECTED CLUSTERS

The homepage offers a snapshot of connected Kafka clusters, providing the status of brokers and a count of associated consumer groups. As you explore topics, the homepage conveniently presents details about your recent topic views.

To find more information:

- Click on a cluster name to find cluster metrics in the **Cluster overview** page.
- Click on a recently viewed topic to retrieve details about that particular topic.

# CHAPTER 5. CLUSTER OVERVIEW PAGE

The **Cluster overview** page shows the status of a Kafka cluster. Here, you can assess the readiness of Kafka brokers, identify any cluster errors or warnings, and gain crucial insights into the cluster's health. At a glance, the page provides information on the number of topics and partitions within the cluster, along with their replication status. Explore cluster metrics through charts displaying used disk space, CPU utilization, and memory usage. Additionally, topic metrics offer a comprehensive view of total incoming and outgoing byte rates for all topics in the Kafka cluster.

## 5.1. ACCESSING CLUSTER CONNECTION DETAILS FOR CLIENT ACCESS

When connecting a client to a Kafka cluster, retrieve the necessary connection details from the **Cluster overview** page by following these steps.

### Procedure

1. From the Streams for Apache Kafka Console, click the name of the Kafka cluster that you want to connect to, then click **Cluster overview** and **Cluster connection details**.

2. Copy and add bootstrap address and connection properties to your Kafka client configuration to establish a connection with the Kafka cluster.

**NOTE**

Ensure that the authentication type used by the client matches the authentication type configured for the Kafka cluster.

# CHAPTER 6. TOPICS PAGE

The **Topics** page shows all the topics created for a Kafka cluster. Use this page to check information on topics.

The **Topics** page shows the overall replication status for partitions in the topic, as well as counts for the partitions in the topic and the number of associated consumer groups. The overall storage used by the topic is also shown.

**WARNING**

Internal topics must not be modified. You can choose to hide internal topics from the list of topics returned on the **Topics** page.

By clicking on a topic name, additional topic information is presented on a series of tabs:

**Messages**

Messages shows the message log for a topic.

**Partitions**

Partitions shows the replication status of each partition in a topic.

**Consumer groups**

Consumer groups lists the names and status of the consumer groups and group members connected to a topic.

**Configuration**

Configuration shows the configuration of a topic.

If a topic is shown as **Managed**, it means that is managed using the Streams for Apache Kafka Topic Operator and was not created directly in the Kafka cluster.

Use the information provided on the tabs to check and modify the configuration of your topics.

## 6.1. CHECKING TOPIC MESSAGES

Track the flow of messages for a specific topic from the **Messages** tab. The **Messages** tab presents a chronological list of messages for a topic.

**Procedure**

1. From the Streams for Apache Kafka Console, click the name of the Kafka cluster, then click **Topics**.
2. Click the name of the topic you want to check.
3. Check the information on the **Messages** tab.
   For each message, you can see its timestamp (in UTC), offset, key, and value.

   By clicking on a message, you can see the full message details.

Click the **Manage columns** icon (represented as two columns) to choose the information to display.

4. Click the search dropdown and select the advanced search options to refine your search. Choose to display the latest messages or messages from a specified time or offset. You can display messages for all partitions or a specified partition.

   When you are done, you can click the CSV icon (represented as a CSV file) to download the information on the returned messages.

## Refining your search

In this example, search terms, and message, retrieval, and partition options are combined:

- **messages=timestamp:2024-03-01T00:00:00Z retrieve=50 partition=1 Error on page load where=value**

The filter searches for the text "Error on page load" in partition 1 as a message value, starting from March 1, 2024, and retrieves up to 50 messages.

### Search terms

Enter search terms as text (*has the words*) to find specific matches and define *where* in a message to look for the term. You can search anywhere in the message or narrow the search to the key, header, or value.
For example:

- **messages=latest retrieve=100 642-26-1594 where=key**

This example searches the latest 100 messages on message key **642-26-1594**.

### Message options

Set the starting point for returning messages.

- **Latest** to start from the latest message.
  - **messages=latest**
- **Timestamp** to start from an exact time and date in ISO 8601 format.
  - **messages=timestamp:2024-03-14T00:00:00Z**
- **Offset** to start from an offset in a partition. In some cases, you may want to specify an offset without a partition. However, the most common scenario is to search by offset within a specific partition.
  - **messages=offset:5600253 partition=0**
- **Unix Timestamp** to start from a time and date in Unix format.
  - **messages=epoch:1**

### Retrieval options

Set a retrieval option.

- **Number of messages** to return a specified number of messages.
  - **messages=latest retrieve=50**

- **Continuously** to return the latest messages in real-time. Click the pause button (represented by two vertical lines) to pause the refresh. Unpause to continue the refresh.
    - **retrieve=continuously**

**Partition options**

Choose to run a search against all partitions or a specific partition.

## 6.2. CHECKING TOPIC PARTITIONS

Check the partitions for a specific topic from the **Partitions** tab. The **Partitions** tab presents a list of partitions belonging to a topic.

**Procedure**

1. From the Streams for Apache Kafka Console, click the name of the Kafka cluster, then click **Topics**.
2. Click the name of the topic you want to check from the **Topics** page.
3. Check the information on the **Partitions** tab.

For each partition, you can see its replication status, as well as information on designated partition leaders, replica brokers, and the amount of data stored by the partition.

You can view partitions by replication status:

**In-sync**

All partitions in the topic are fully replicated. A partition is fully-replicated when its replicas (followers) are 'in-sync' with the designated partition leader. Replicas are 'in-sync' if they have fetched records up to the log end offset of the leader partition within an allowable lag time, as determined by **replica.lag.time.max.ms**.

**Under-replicated**

A partition is under-replicated if some of its replicas (followers) are not in-sync. An under-replicated status signals potential issues in data replication.

**Offline**

Some or all partitions in the topic are currently unavailable. This may be due to issues such as broker failures or network problems, which need investigating and addressing.

You can also check information on the broker designated as partition leader and the brokers that contain the replicas:

**Leader**

The leader handles all produce requests. Followers on other brokers replicate the leader’s data. A follower is considered in-sync if it catches up with the leader’s latest committed message.

**Preferred leader**

When creating a new topic, Kafka’s leader election algorithm assigns a leader from the list of replicas for each partition. The algorithm aims for a balanced spread of leadership assignments. A "Yes" value indicates the current leader is the preferred leader, suggesting a balanced leadership distribution. A "No" value may suggest imbalances in the leadership assignments, requiring further investigation. If the leadership assignments of partitions are not well-balanced, it can contribute to size discrepancies. A well-balanced Kafka cluster should distribute leadership roles across brokers evenly.

**Replicas**

Followers that replicate the leader's data. Replicas provide fault tolerance and data availability.

NOTE

Discrepancies in the distribution of data across brokers may indicate balancing issues in the Kafka cluster. If certain brokers are consistently handling larger amounts of data, it may indicate that partitions are not evenly distributed across the brokers. This could lead to uneven resource utilization and potentially impact the performance of those brokers.

## 6.3. CHECKING TOPIC CONSUMER GROUPS

Check the consumer groups for a specific topic from the **Consumer groups** tab. The **Consumer groups** tab presents a list of consumer groups associated with a topic.

**Procedure**

1. From the Streams for Apache Kafka Console, click the name of the Kafka cluster, then click **Topics**.
2. Click the name of the topic you want to check from the **Topics** page.
3. Check the information on the **Consumer groups** tab.
4. To check consumer group members, click the consumer group name.

For each consumer group, you can see its status, the overall consumer lag across all partitions, and the number of members. For more information on checking consumer groups, see Chapter 8, *Consumer groups page*.

For each group member, you see the unique (consumer) client ID assigned to the consumer within the consumer group, overall consumer lag, and the number of assigned partitions. For more information on checking consumer group members, see Section 8.1, "Checking consumer group members".

NOTE

Monitoring consumer group behavior is essential for ensuring optimal distribution of messages between consumers.

## 6.4. CHECKING TOPIC CONFIGURATION

Check the configuration of a specific topic from the **Configuration** tab. The **Configuration** tab presents a list of configuration values for the topic.

**Procedure**

1. From the Streams for Apache Kafka Console, click the name of the Kafka cluster, then click **Topics**.
2. Click the name of the topic you want to check from the **Topics** page.
3. Check the information on the **Configuration** tab.

You can filter for the properties you wish to check, including selecting by data source:

- **DEFAULT_CONFIG** properties have a predefined default value. This value is used when there are no user-defined values for those properties.
- **STATIC_BROKER_CONFIG** properties have predefined values that apply to the entire broker and, by extension, to all topics managed by that broker. This value is used when there are no user-defined values for those properties.
- **DYNAMIC_TOPIC_CONFIG** property values have been configured for a specific topic and override the default configuration values.

TIP

The Streams for Apache Kafka Topic Operator simplifies the process of creating managing Kafka topics using **KafkaTopic** resources.

# CHAPTER 7. BROKERS PAGE

The **Brokers** page shows all the brokers created for a Kafka cluster. For each broker, you can see its status, as well as the distribution of partitions across the brokers, including the number of partition leaders and replicas.

The broker status is shown as one of the following:

**Stable**
: A stable broker is operating normally without significant issues.

**Unstable**
: An unstable broker may be experiencing issues, such as high resource usage or network problems.

If the broker has a rack ID, this is the ID of the rack or datacenter in which the broker resides.

Click on the right arrow (>) next to a broker name to see more information about the broker, including its hostname and disk usage.

**NOTE**

Consider rebalancing if the distribution is uneven to ensure efficient resource utilization.

# CHAPTER 8. CONSUMER GROUPS PAGE

The **Consumer groups** page shows all the consumer groups associated with a Kafka cluster. For each consumer group, you can see its status, the overall consumer lag across all partitions, and the number of members. Click on associated topics to show the topic information available from the **Topics** page tabs.

Consumer group status can be one of the following:

- **Stable** indicates normal functioning
- **Rebalancing** indicates ongoing adjustments to the consumer group's members.
- **Empty** suggests no active members. If in the empty state, consider adding members to the group.

Check group members by clicking on a consumer group name. For more information on checking consumer group members, see Section 8.1, "Checking consumer group members".

## 8.1. CHECKING CONSUMER GROUP MEMBERS

Check the members of a specific consumer group from the **Consumer groups** page.

**Procedure**

1. From the Streams for Apache Kafka Console, click the name of the Kafka cluster, then click **Consumer groups**.
2. Click the name of the consumer group you want to check from the **Consumer groups** page.
3. Click on the right arrow (>) next to a member ID to see the topic partitions a member is associated with, as well as any possible consumer lag.

For each group member, you see the unique (consumer) client ID assigned to the consumer within the consumer group, overall consumer lag, and the number of assigned partitions.

Any consumer lag for a specific topic partition reflects the gap between the last message a consumer has picked up (committed offset position) and the latest message written by the producer (end offset position).

# APPENDIX A. USING YOUR SUBSCRIPTION

Streams for Apache Kafka is provided through a software subscription. To manage your subscriptions, access your account at the Red Hat Customer Portal.

## Accessing Your Account

1. Go to access.redhat.com.
2. If you do not already have an account, create one.
3. Log in to your account.

## Activating a Subscription

1. Go to access.redhat.com.
2. Navigate to **My Subscriptions**.
3. Navigate to **Activate a subscription** and enter your 16-digit activation number.

## Downloading Zip and Tar Files

To access zip or tar files, use the customer portal to find the relevant files for download. If you are using RPM packages, this step is not required.

1. Open a browser and log in to the Red Hat Customer Portal **Product Downloads** page at access.redhat.com/downloads.
2. Locate the **Streams for Apache Kafka for Apache Kafka**entries in the **INTEGRATION AND AUTOMATION** category.
3. Select the desired Streams for Apache Kafka product. The **Software Downloads** page opens.
4. Click the **Download** link for your component.

## Installing packages with DNF

To install a package and all the package dependencies, use:

```
dnf install <package_name>
```

To install a previously-downloaded package from a local directory, use:

```
dnf install <path_to_download_package>
```

*Revised on 2024-05-30 17:23:28 UTC*